From NIZK *Arguments* to ZAPs, Generically

Anish Banerjee
UT Austin
anish@cs.utexas.edu

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

David J. Wu
UT Austin
dwu4@cs.utexas.edu

Abstract

Dwork and Naor (FOCS 2000) showed a generic transformation to construct a ZAP (a two-round public-coin witness-indistinguishable proof) from any non-interactive zero-knowledge (NIZK) proof with statistical soundness in the common random string model. In recent years, a number of works have shown how to construct NIZK *arguments* in the common random string model from a broad range of assumptions including decisional Diffie-Hellman (DDH), learning with errors (LWE), or combinations of multiple assumptions. While a number of previous works have developed specialized tools to build ZAPs using these same assumptions (through a non-trivial adaptation of the underlying NIZK), a natural question is whether we can *generically* obtain a ZAP from these NIZK arguments à la Dwork-Naor.

In this work, we introduce the notion of a *sometimes-constricting generator* and show how to use it to *generically* upgrade any computational (resp., statistical) NIZK argument in the common random string model into a computational (resp., statistical) ZAP argument. We then show how to build sometimes-constricting generators from either the DDH assumption (over pairing-free groups) or the LWE assumption. Our transformation immediately allows us to recover constructions of ZAPs from assumptions like DDH or LWE, as well as enables new constructions from different combinations of cryptographic assumptions with properties that were not previously attainable. More broadly, our compiler provides a general mechanism to convert any future NIZK construction in the common random string model into a ZAP.

1 Introduction

Zero-knowledge proofs for NP [GMR85] allow a prover to convince a verifier that an NP statement is true without revealing anything more than the fact that the statement is true. While zero-knowledge proofs are a cornerstone of modern cryptography, constructing them either requires assuming some kind of setup (to sample a common reference string) or relies on interaction between a prover and a verifier. In the plain model, at least three rounds of interaction are needed [GO94, BLV03]. One way to overcome these limitations is to relax zero-knowledge to witness indistinguishability, which asks that a proof for an NP statement x generated using a witness w_0 should be indistinguishable from a proof for the same statement generated using a witness w_1. Unlike the case of zero-knowledge, there are no lower bounds on the round complexity of witness indistinguishable proofs.

ZAPs. In this work, we focus on ZAPs [DN00], which are two-message public-coin witness indistinguishable proofs in the plain model. The seminal work of Dwork and Naor introduced the notion of a ZAP and then showed how to generically construct it from any non-interactive zero-knowledge (NIZK) proof whose common reference string (CRS) is uniformly random. The classic NIZK based on factoring by Feige, Lapidot, and Shamir [FLS90] for instance satisfies this property. The Dwork-Naor transformation applies as long as the underlying NIZK satisfies the following properties: (1) the NIZK is statistically sound; and (2) the CRS is a uniform random string. Today, NIZKs for NP are known from numerous cryptographic assumptions including quadratic residuosity [BFM88], factoring [FLS90], pairing-based assumptions [CHK03, GOS06, LPWW20], assumptions over pairing-free groups [CKU20, JJ21, CJJQ23], lattice-based assumptions [CCH+19, PS19, Wat24, WWW25, BCD+25], as well as the learning parity with noise assumption in conjunction with another algebraic assumption [BKM20, DJJ24]. However, in many of these settings, the underlying NIZK only satisfies one of the two requirements of the Dwork-Naor transformation. For instance, many of these aforementioned constructions like [CCH+19, BKM20, JJ21, PS19, DJJ24, WWW25, BCD+25] are only

computationally sound with a uniform random string. As a result, we cannot generically apply the Dwork-Naor transformation to obtain ZAPs from the same underlying assumption. Indeed, recent ZAPs based on assumptions like the decisional Diffie-Hellman (DDH) assumption over pairing-free groups [JJ21], pairing-based assumptions [CKSU21], or lattice-based assumptions [LVW19, BFJ+20, GJJM20, BLN+25] have all taken a specialized approach that starts from a specific NIZK scheme (e.g., [JJ21, LPWW20, PS19, WWW25]) and then modifies the structure to obtain a ZAP.

This work: generically constructing ZAPs from NIZK arguments. The beauty of the Dwork-Naor transformation is that it provides a generic approach to take any NIZK proof in the uniform random string model and convert it into a ZAP. There is no need to understand the internal details of the NIZK proof to construct the ZAP. A natural question then is whether we can achieve an analogous transformation starting from a weaker primitive (say, a NIZK *argument* in the uniform random string model). In this work, we show that using a new tool called a sometimes-constricting generator, we can generically compile *any* NIZK argument in the common random string model into a computational ZAP argument. Specifically, we prove the following theorem:

Theorem 1.1 (Informal). *Let Π_{NIZK} be a sub-exponentially-secure computational (resp., statistical) NIZK argument for* NP *in the common random string model and let Π_{SCG} be a sub-exponentially-secure sometimes-constricting generator. Then, we obtain a sub-exponentially-secure computational (resp., statistical) ZAP argument for* NP.

We also note that we can replace sub-exponential security of *either* the NIZK or the sometimes-constricting generator (but not both) in Theorem 1.1 with quasi-polynomial security (i.e., $2^{-\log^c \lambda}$-security for a constant $c > 1$ where λ is the security parameter). In return, we obtain a quasi-polynomially-secure computational (resp., statistical) ZAP argument.

Constructing sometimes-constricting generators. We then show how to build sometimes-constricting generator from the standard decisional Diffie-Hellman (DDH) assumption over pairing-free groups or the learning with errors (LWE) assumption. Our techniques are inspired by techniques used to build lossy functions from the same underlying assumptions [PW08]. In conjunction with Theorem 1.1, our results allow us to upgrade any NIZK argument (with super-polynomial security) in the common random string model to a computational ZAP argument from either the DDH or the LWE assumption. For instance, we immediately recover several existing implications by applying our transformation to existing NIZK schemes:

- **ZAPs from DDH:** If we apply our transformation to the statistical NIZK argument based on sub-exponentially-secure DDH from [JJ21], then we obtain a statistical ZAP argument from sub-exponentially-secure DDH. The resulting ZAP has quasi-polynomial security (since the underlying NIZK only provides quasi-polynomial security). This yields a ZAP with the same properties as the specialized construction from [JJ21] (which they obtain by adapting their NIZK construction).
- **ZAPs from LWE:** If we apply our transformation to a statistical NIZK argument from learning with errors (e.g., any of [PS19, WWW25, BCD+25]), then we obtain a (sub-exponentially-secure) statistical ZAP argument from the sub-exponentially-secure learning with errors assumption. This is comparable to previous lattice-based statistical ZAP arguments [LVW19, BFJ+20, GJJM20, BLN+25]. An advantage of previous constructions is that they can also be instantiated with quasi-polynomially-secure LWE to get a quasi-polynomially-secure statistical ZAP. However, prior constructions were tailored either to the specific structure of the [PS19] NIZK based on correlation-intractable hash functions [LVW19, BFJ+20, GJJM20] or to the specific structure of a hidden-bits model NIZK [WWW25, BLN+25]. Our approach applies irrespective of the structure of the underlying NIZK.

Using our generic compiler, we can also obtain several new implications that were previously not known. We list two examples below:

- **Sub-exponentially-secure ZAPs from DDH and LPN:** If we apply our transformation to the computational NIZK argument based on DDH and LPN from [BCD+25],[1] then we obtain a sub-exponentially-secure computational ZAP argument from sub-exponential DDH and sub-exponential LPN. Previously, it was not known how to get a ZAP with sub-exponential security from this combination of assumptions.

[1] The NIZK is a computational NIZK proof with a pseudorandom CRS. If we instantiate the scheme with a uniform random string, then the scheme becomes a computational NIZK argument.

- **Quasi-polynomially-secure ZAPs from DDH, LPN, and MQ:** We can also apply our transformation to the computational NIZK argument based on LPN and the multivariate quadratic assumption (MQ) from [DJJ24]. If we assume sub-exponential hardness of LPN and exponentially-hard MQ (to get a sub-exponentially-secure NIZK), and *quasi-polynomial* hardness of DDH (to get a quasi-polynomially-secure sometimes-constricting generator), then we obtain a quasi-polynomially-secure ZAP. Once again, this combination of assumptions was not previously known to yield a ZAP.

1.1 Technical Overview

We start by recalling the classic Dwork-Naor template [DN00] of constructing a ZAP proof from a NIZK proof. Recall first that in a NIZK for NP, there are three algorithms: (1) a setup algorithm that samples a common reference string (CRS); (2) a prover algorithm that takes as input the CRS, the statement x, an NP witness w, and outputs a proof π; and (3) a verification algorithm that takes as input the CRS, the statement x, and the proof π and decides whether to accept or reject. Critically, both soundness and zero-knowledge of the NIZK assume that the CRS was generated honestly. When designing a ZAP from a NIZK, a natural strategy is to have the prover and the verifier *jointly* sample a CRS for the NIZK and then take the proof of the statement to be the NIZK proof with respect to the jointly-sampled CRS. The question is how to implement this joint sampling procedure. The general template of Dwork and Naor works as follows:

- The verifier picks an initial common reference string crs_V and sends it as its initial message.
- The prover then picks a "tweak" crs_P and the overall common reference string $\mathsf{crs}_{\mathsf{NIZK}}$ for the NIZK is derived as a deterministic function of $(\mathsf{crs}_P, \mathsf{crs}_V)$. The prover generates a proof π with respect to $\mathsf{crs}_{\mathsf{NIZK}}$ and sends (crs_P, π) to the verifier as the proof.
- The verifier derives $\mathsf{crs}_{\mathsf{NIZK}}$ from $(\mathsf{crs}_V, \mathsf{crs}_P)$ and then checks the proof π with respect to $\mathsf{crs}_{\mathsf{NIZK}}$.

When designing a ZAP, we must strike a balance between two competing goals:

- For soundness, the prover's tweak crs_P must be sufficiently constrained. Namely, the prover must not be able to efficiently find a tweak crs_P that induces a bad crs where it is possible to break soundness. In particular, the prover must not be able to unilaterally choose $\mathsf{crs}_{\mathsf{NIZK}}$, as this would allow it to trivially break soundness.
- For witness indistinguishability, the prover's tweak must have sufficient entropy. Namely, the verifier should not be able to choose a crs_V where for most tweaks crs_P available to the prover, the induced CRS $\mathsf{crs}_{\mathsf{NIZK}}$ from $(\mathsf{crs}_P, \mathsf{crs}_V)$ fails to hide the witness.

The work of Dwork-Naor shows how to satisfy both properties by relying on a reverse randomization technique and *statistical* soundness of the underlying NIZK. In this work, our starting point is a NIZK *argument* which only provides computational soundness. To rely on computational soundness, we instead follow an alternative approach based on complexity leveraging inspired by the works of [LVW19, BFJ+20, GJJM20, CKSU21, BLN+25]. This will enable us to balance the two competing objectives. Conceptually, these previous constructions essentially do the following:

- With probability $\varepsilon = \varepsilon(\lambda)$ over the choice of crs_V, the verifier is able to "program" $\mathsf{crs}_{\mathsf{NIZK}}$ to a value of its choosing. Observe that such a verifier is able to break witness indistinguishability with probability *at least* ε (i.e., we can assume that the verifier would always try to program $\mathsf{crs}_{\mathsf{NIZK}}$ into one that would allow it to break witness indistinguishability). Thus, it must be the case that $\varepsilon(\lambda)$ is a *negligible* function. In this setting, this property does not inherently contradict witness indistinguishability.

  At the same time, this property allows us to argue soundness provided that the underlying NIZK satisfies a notion of ε-soundness where the probability that an efficient adversary breaks soundness (with respect to a uniform random reference string) is at most $\varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$. Namely, suppose an adversary breaks soundness of the ZAP with non-negligible probability $\delta = \delta(\lambda)$. Suppose moreover that over the choice of crs_V, the adversary succeeds in breaking soundness with advantage $\varepsilon(\lambda) \cdot \delta(\lambda)$ for a target $\mathsf{crs}_{\mathsf{NIZK}}$ chosen by the verifier (in this case, for an honestly-chosen CRS). Note that this latter property is *stronger* than just being able to target a $\mathsf{crs}_{\mathsf{NIZK}}$ with probability ε. We require the stronger property that the success probability of the adversary

breaking soundness with respect to the target reference string $\mathsf{crs}_{\mathsf{NIZK}}$ is δ scaled down by exactly ε. (Intuitively, we want the success of the adversary to be independent of whether the verifier can program the CRS of its choice.) If this were the case, then we have found an adversary that breaks soundness of the underlying NIZK with probability $\varepsilon(\lambda) \cdot \delta(\lambda)$ for some non-negligible δ. This contradicts ε-soundness of the NIZK. Because we need to set ε to be inverse super-polynomial, this step requires the NIZK to be super-polynomially-secure.

- On the flip side, we also require that for *any* choice of the verifier message crs_V, with all but negligible probability over the choice of crs_P, the induced CRS $\mathsf{crs}_{\mathsf{NIZK}}$ associated with $(\mathsf{crs}_P, \mathsf{crs}_V)$ is statistically close to a uniform random string. This is enough to guarantee witness indistinguishability, since with overwhelming probability over the choice of crs_P, the prover is generating a proof with respect to a uniform random string. This allows us to appeal to the fact that the NIZK is zero knowledge, and thus, witness indistinguishable. Note that the statistical distance with the uniform distribution must be *at least* ε (since with probability at least ε, the verifier has complete control over the value of $\mathsf{crs}_{\mathsf{NIZK}}$).

The works of [LVW19, BFJ+20, GJJM20, CKSU21, BLN+25] developed various algebraic mechanisms to integrate this general template with the algebraic structure of an existing NIZK construction [PS19, LPWW20, WWW25]. In this work, we take a different strategy and introduce a cryptographic primitive that captures this template and enables a generic compiler that is agnostic to the specific structure of the underlying NIZK argument.

Sometimes-constricting generator. The main cryptographic notion we introduce in this work is that of a sometimes-constricting generator. The properties of the sometimes-constricting generator capture the general template described above for constructing a ZAP from a NIZK argument. Specifically, a sometimes-constricting generator consists of three algorithms $(\mathsf{Setup}, \mathsf{Sample}, \mathsf{Verify})$ with the following properties:

- The setup algorithm Setup takes the security parameter λ and an output length ℓ_{out} and outputs a set of public parameters pp together with a "guess" for the output string $t^* \in \{0,1\}^{\ell_{\mathsf{out}}}$.
- The Sample algorithm is a randomized algorithm that takes the public parameters pp and outputs a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ together with a proof π attesting to the value of y.
- Finally, the verification algorithm Verify takes the public parameters pp, an output string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ and a proof π and outputs a bit indicating whether the string y is valid or not.

The main requirements are as follows:

- **Completeness:** The first property simply says that Verify accepts all pairs (y, π) output by Sample.
- **Mode indistinguishability:** The second property requires that the public parameters pp output by Setup be computationally indistinguishable from a uniform random string. This is important for obtaining a public-coin ZAP.
- **Target randomness:** The next property essentially asserts that for *all* public parameters pp, the strings $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ output by Sample are statistically close to uniform. In fact, we impose a stronger requirement by requiring an efficient algorithm Target such that for all public parameters pp, the following distributions are statistically indistinguishable (i.e., have statistical distance at most $\mathsf{negl}(\lambda)$):

$$\{(y,\pi) : (y,\pi) \leftarrow \mathsf{Sample}(\mathsf{pp})\} \quad \text{and} \quad \left\{(y,\pi) : \begin{array}{c} y \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}} \\ \pi \leftarrow \mathsf{Target}(\mathsf{td}, y) \end{array}\right\}. \tag{1.1}$$

  Here td denotes some trapdoor information associated with the public parameters pp that is used for targeting. We separately assume that there is an inefficient algorithm Preprocess that takes any collection of public parameters pp and outputs an associated targeting trapdoor associated with it.
- **μ-guessing:** The final property essentially says that with probability at least $\mu = \mu(\lambda)$ over the choice of $(\mathsf{pp}, t^*) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}})$, any efficient algorithm $\mathcal{A}$ that successfully outputs (y, π) where $\mathsf{Verify}(\mathsf{pp}, y, \pi) = 1$

will output $y = t^*$ with probability that is negligibly close to μ.[2] Importantly for our application, we require that μ is a function of the security parameter (and does *not* depend on the output length ℓ_{out}). Note that this is only feasible for values of μ that are smaller than the statistical distance between the distributions in Eq. (1.1).

ZAPs from any NIZK argument and a sometimes-constricting generator. Given a sometimes-constricting generator, it is straightforward to construct a ZAP from a NIZK argument. Suppose the underlying NIZK argument has a uniform random string of length σ. We will use a sometimes-constricting generator to sample strings of length σ. In the scheme, the verifier chooses a uniform random string $z \xleftarrow{R} \{0,1\}^\sigma$ and the prover derives a string y using the sometimes-constricting generator. The joint CRS for the NIZK is then $y \oplus z$. As we see below, the properties of the sometimes-constricting generator will enable us to argue soundness and witness indistinguishability. We now give a sketch of our construction:

- **Verifier's message:** The verifier's initial message consists of a random $\mathsf{pp}_{\mathsf{SCG}} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$ for the sometimes-constricting generator and a random string $z \xleftarrow{R} \{0,1\}^\sigma$. Here, ℓ_{pp} is the length of the public parameters for the sometimes-constricting generator.
- **Prover's message:** Given an NP statement x and the associated witness w, the prover first runs $(y, \pi_{\mathsf{SCG}}) \leftarrow \mathsf{Sample}(\mathsf{pp}_{\mathsf{SCG}})$ and then sets $\mathsf{crs}_{\mathsf{NIZK}} = y \oplus z$. Then it uses the NIZK to generate a proof π_{NIZK} of x with respect to $\mathsf{crs}_{\mathsf{NIZK}}$. The overall proof is then $(y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$.
- **Verification:** To check the proof, the verifier first checks that $\mathsf{Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$. If so, it constructs $\mathsf{crs}_{\mathsf{NIZK}} = y \oplus z$ and then checks that π_{NIZK} is a valid proof for x with respect to the $\mathsf{crs}_{\mathsf{NIZK}}$.

We refer to Construction 3.2 for the formal details. We now provide a brief sketch of the security analysis.

Soundness. Soundness relies on mode indistinguishability and the μ-guessing property of the sometimes-constricting generator together with soundness of the NIZK. In particular, we need to assume that the underlying NIZK is μ-sound, which means a polynomial-time adversary can break soundness with advantage at most $\mu(\lambda) \cdot \mathsf{negl}(\lambda)$. Then, our analysis proceeds as follows:

- First, by mode indistinguishability of the sometimes-constricting generator, we can replace the public parameters $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$ with the public parameters pp output by Setup. Let $t^* \in \{0,1\}^\sigma$ be the associated guess output by Setup.
- Let $\mathsf{crs}_{\mathsf{NIZK}} \xleftarrow{R} \{0,1\}^\sigma$ be an honestly-generated CRS for the NIZK. We program $\mathsf{crs}_{\mathsf{NIZK}}$ into the verifier's initial message by setting $z = \mathsf{crs}_{\mathsf{NIZK}} \oplus t^*$. Since $\mathsf{crs}_{\mathsf{NIZK}}$ is uniformly random, this does not change the distribution of z.
- At this point, we can rely on μ-guessing security of π_{SCG}. This property ensures that if a prover outputs a false statement x and a proof $(y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ where $\mathsf{Verify}(y, \pi_{\mathsf{SCG}}) = 1$ and π_{NIZK} is a valid proof of x with respect to the CRS $y \oplus z$, then with probability at least $\mu(\lambda)$, it will additionally be the case that $y = t^*$. However, since we programmed $z = \mathsf{crs}_{\mathsf{NIZK}} \oplus t^*$ into the verifier's message, this means π_{NIZK} is a valid proof with respect to the CRS $y \oplus z = t^* \oplus \mathsf{crs}_{\mathsf{NIZK}} \oplus t^* = \mathsf{crs}_{\mathsf{NIZK}}$. Since $\mathsf{crs}_{\mathsf{NIZK}}$ is honestly-generated, this means the prover broke soundness with advantage $\mu(\lambda)$, which contradicts μ-soundness of the original NIZK.

As noted above, the sometimes-constricting generator can only satisfy μ-guessing security for a choice of μ that is a negligible function (i.e., bounded by an inverse super-polynomial function). In turn, this means we need to assume that an efficient adversary can only break soundness with advantage at most $\mu(\lambda) \cdot \mathsf{negl}(\lambda)$, which is a stronger requirement than standard polynomial hardness. One way to satisfy our requirements is to assume sub-exponential hardness of the NIZK and complexity leverage (see Lemma 2.9). We refer to the proof of Theorem 3.4 for the full details.

[2] The formal definition is more intricate and we refer to Definition 3.1 for the precise definition.

Witness indistinguishability. Witness indistinguishability follows by the target randomness property of the sometimes-constricting generator. Specifically, the target randomness property says that for any choice of pp, the distribution of y output by Sample is statistically close to uniform. In the construction, the prover constructs its proof with respect to $\mathsf{crs}_{\mathsf{NIZK}} = y \oplus z$. Thus, if the string $y \in \{0,1\}^\sigma$ output by Sample is statistically close to uniform, the distribution of $\mathsf{crs}_{\mathsf{NIZK}}$ is also statistically close to uniform. This allows us to appeal to zero-knowledge of the underlying NIZK to argue that π_{NIZK} hides the witness. Formally, in the security analysis, the reduction would get $\mathsf{crs}_{\mathsf{NIZK}} \in \{0,1\}^\sigma$ from the NIZK challenger and the reduction would set $y = \mathsf{crs}_{\mathsf{NIZK}} \oplus z$ and then use the Target algorithm to obtain the proof $\pi_{\mathsf{SCG}} \leftarrow \mathsf{Target}(\mathsf{td}, y)$. In this case, the trapdoor td is a function of the public parameters and could be provided to the reduction algorithm as *non-uniform* advice; correspondingly, witness indistinguishability relies on security of the underlying NIZK against non-uniform adversaries.[3] Finally, we note that if the underlying NIZK argument satisfies statistical zero-knowledge, then our analysis yields a statistical ZAP argument. We refer to Theorem 3.9 for the formal analysis.

Constructing a sometimes-constricting generator. It remains to show how to construct a sometimes-constricting generator. To do so, we first introduce an intermediate notion called a branch-constricting generator, which can be more directly built from algebraic assumptions such as DDH or LWE. A branch-constricting generator is syntactically very similar to a sometimes-constricting generator except we introduce an additional "branch parameter" with the following properties:

- The setup, sample, and verification algorithms all take an additional branch parameter $\mathsf{br} \in \mathfrak{B}$ as input.
- Completeness is similar to before. Namely, if we sample a pair (y, π) with respect to a branch br, then the pair verifies with respect to the same branch. Mode indistinguishability says that the public parameters associated with *any* branch br are computationally indistinguishable from a random string. In particular, this means the public parameters pp must computationally hide the associated branch br.
- Target randomness is defined similarly to before and says that for *any* choice of pp, if we sample a *random* branch $\mathsf{br} \xleftarrow{R} \mathfrak{B}$, then the string y output by $\mathsf{Sample}(\mathsf{pp}, \mathsf{br})$ is statistically close to uniform. Formally, it is defined with respect to an explicit Target algorithm exactly as before.
- The final property (in place of μ-guessing) says that if the public parameters pp are generated for a particular branch $\mathsf{br}^* \in \mathfrak{B}$, then the set of possible values $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ where $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}^*, y, \pi) = 1$ is small (i.e., has size $2^{\mathsf{poly}(\lambda)}$ that only depends on the security parameter and not the output length ℓ_{out}). Technically, we impose a stronger requirement that says there is an efficient algorithm SampleGuess to sample from the set of possible outputs y that $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}^*, y, \cdot)$ may accept.

It is straightforward to build a sometimes-constricting generator from a branch-constricting generator:

- The setup algorithm for the sometimes-constricting generator would sample a random branch $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$ and output the public parameters for br^*. The guess $t^* \in \{0,1\}^{\ell_{\mathsf{out}}}$ is obtained via SampleGuess.
- The sampling algorithm for the sometimes-constricting generator would simply run the sampling algorithm for the branch-constricting generator with a *random* branch $\mathsf{br} \xleftarrow{R} \mathfrak{B}$ to obtain a pair (y, π_{BCG}). The chosen branch br is included as part of the proof $\pi_{\mathsf{SCG}} = (\mathsf{br}, \pi_{\mathsf{BCG}})$. The verification algorithm simply invokes the corresponding algorithm for the branch-constricting generator with branch br and proof π_{SCG}.

Completeness, mode indistinguishability, and target randomness now follow via the corresponding properties for the underlying scheme. For the μ-guessing property, the argument proceeds as follows:

- For ease of exposition, suppose the public parameters pp are associated with a branch br^* and let $S^* \subset \{0,1\}^{\ell_{\mathsf{out}}}$ be the set of values $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ for which $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}^*, y, \cdot)$ accepts. Recall that the size of S^* depends only on λ and not on ℓ_{out}. Suppose also that SampleGuess samples uniformly from S^*. In Section 4, we show how to avoid these simplifying assumptions.

[3] Alternatively, if td can be computed in $2^{\mathsf{poly}(\lambda)}$ time (independent of ℓ_{out}), we could also consider a super-polynomial-time reduction algorithm that runs both Preprocess and Target. This would be fine as long as the NIZK is secure against super-polynomial-time adversaries. If the underlying NIZK satisfies *statistical* zero-knowledge, then we can also consider an inefficient reduction that simply runs both Preprocess and Target. We provide more discussion in Theorem 3.9.

- Consider an adversary $\mathcal{A}$ for the μ-guessing security game. Let $(y, (\mathsf{br}, \pi_{\mathsf{BCG}}))$ be the value output by this adversary. Our goal is to argue that with probability at least μ, it will be the case that $y = t^*$.
- First, suppose that no efficient adversary can break mode indistinguishability of the branch-constricting generator with probability better than $1/|\mathfrak{B}| \cdot \mathsf{negl}(\lambda)$. Since Setup samples the target branch $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$, this means that $\mathsf{br} = \mathsf{br}^*$ with probability at least $1/|\mathfrak{B}| \cdot (1 - \mathsf{negl}(\lambda))$. Namely, this level of mode indistinguishability ensures that adversary $\mathcal{A}$ cannot consistently avoid choosing $\mathsf{br} = \mathsf{br}^*$.
- Suppose $(\mathsf{br}, \pi_{\mathsf{BCG}})$ is a valid proof for y. When $\mathsf{br} = \mathsf{br}^*$, this means that $y \in S^*$. If SampleGuess outputs $t^* \leftarrow S^*$, then $t^* = y$ with probability $1/|S^*|$.
- We conclude that the guess t^* is correct if we guessed the correct branch br^* and the correct target within the constrained set S^* associated with br^*. This occurs with probability $1/(|\mathfrak{B}| \cdot |S^*|)$. If we set $\mathfrak{B}$ to be a function of only the security parameter, then the resulting scheme satisfies the μ-guessing property with $\mu = 1/(|\mathfrak{B}| \cdot |S^*|)$, which depends only on the security parameter.

We refer to Theorem 4.5 for the full details.

Comparison with all-but-one trapdoor functions. A branch-constricting generator shares a similar flavor as the notion of an all-but-one trapdoor function introduced by Peikert and Waters [PW08]. An all-but-one trapdoor function has several branches, where almost all of the branches implement injective trapdoor functions (with a common trapdoor), except for one branch which is lossy. Analogously, we can think of a branch-constricting generator as a function with several branches, where almost all of the branches are *surjective* functions, except for a *few* that have a much smaller range.

Constructing a branch-constricting generator from DDH. We now sketch how to construct a branch-constricting generator from the DDH assumption. Let $\mathbb{G}$ be a prime-order group of order p and generated by g. For a matrix $\mathbf{M} \in \mathbb{Z}_p^{n \times n}$, we write $[\![\mathbf{M}]\!]$ to denote $g^{\mathbf{M}}$, where exponentiation is defined component-wise. Our construction shares a similar structure to the DDH-based all-but-one function from [PW08]:

- We can take the branch set $\mathfrak{B}$ to be any subset $\mathfrak{B} \subset \mathbb{Z}_p$.
- To sample the public parameters associated with a branch $\mathsf{br}^* \in \mathfrak{B}$, and output length ℓ_{out}, we first sample $\mathbf{u}, \mathbf{v} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$ and then compute $\mathbf{M} = \mathbf{u}\mathbf{v}^\intercal - \mathsf{br}^* \cdot \mathbf{I}$, where $\mathbf{I}$ denotes the identity matrix. The public parameters are $\mathsf{pp} = [\![\mathbf{u}\mathbf{v}^\intercal - \mathsf{br}^* \cdot \mathbf{I}]\!]$.
- Given the public parameters $[\![\mathbf{M}]\!]$ and a branch $\mathsf{br} \in \mathfrak{B}$, the sample algorithm samples a random $\mathbf{r} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$ and computes $[\![(\mathbf{M} + \mathsf{br} \cdot \mathbf{I})\mathbf{r}]\!]$. It then applies a (universal) hash function $h \colon \mathbb{G} \to \{0,1\}$ to each group element to obtain an ℓ_{out}-bit string $y = h([\![(\mathbf{M} + \mathsf{br} \cdot \mathbf{I})\mathbf{r}]\!])$. The proof is the vector $\mathbf{r}$.
- The verification algorithm simply checks that $y = h([\![(\mathbf{M} + \mathsf{br} \cdot \mathbf{I})\mathbf{r}]\!])$.

It is straightforward to check each of the required properties:

- Mode indistinguishability follows from DDH. Specifically, under the DDH assumption, an encoding of a rank-1 matrix $[\![\mathbf{u}\mathbf{v}^\intercal]\!]$ is computationally indistinguishable from the encoding of a random matrix $[\![\mathbf{M}]\!]$ where $\mathbf{M} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$. Thus, the public parameters are computationally indistinguishable from a matrix of uniform random group elements.
- For target randomness, take *any* matrix $\mathbf{M} \in \mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$. Target randomness follows whenever the matrix $(\mathbf{M} + \mathsf{br} \cdot \mathbf{I})$ is full rank. In this case, $(\mathbf{M} + \mathsf{br} \cdot \mathbf{I}) \cdot \mathbf{r}$ is uniform over $\mathbb{Z}_p^{\ell_{\mathsf{out}}}$ when $\mathbf{r} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$. Now, if $\mathbf{M} + \mathsf{br} \cdot \mathbf{I}$ is not full rank, then br must be an eigenvalue of $\mathbf{M}$. Since $\mathbf{M}$ has dimension $\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}$, it can have at most ℓ_{out} distinct eigenvalues. Thus, as long as $\ell_{\mathsf{out}}/|\mathfrak{B}| = \mathsf{negl}(\lambda)$, then the output of Sample with a random branch $\mathsf{br} \xleftarrow{R} \mathfrak{B}$ will yield a uniform random string in $\{0,1\}^{\ell_{\mathsf{out}}}$. (It is straightforward to extend this argument to give an explicit Target algorithm; see Theorem 5.11). We note that this rank argument also featured prominently in the construction of pairing-based ZAPs from the work of [CKSU21].

- Finally, we need to argue that if $\mathsf{pp} = [\![\mathbf{M}]\!]$ is output by Setup for branch br^*, then there are a small number of values $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ where $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}^*, y, \cdot)$ accepts. On the target branch, the verification algorithm only accepts if $y = h([\![(\mathbf{M}+\mathsf{br}^*\cdot\mathbf{I})\mathbf{r}]\!])$. In this case, the setup algorithm sets $\mathbf{M} = \mathbf{u}\mathbf{v}^\intercal - \mathsf{br}^*\cdot\mathbf{I}$ so $\mathbf{M}+\mathsf{br}^*\cdot\mathbf{I} = \mathbf{u}\mathbf{v}^\intercal$, and correspondingly, the verification algorithm accepts only if $y = h(\mathbf{u}\mathbf{v}^\intercal\mathbf{r}) = h(\alpha\mathbf{u})$, for some $\alpha \in \mathbb{Z}_p$. By construction, there are only $p = 2^{\Theta(\lambda)}$ possible values of α. Observe that this only depends on the group order (which is a function of the security parameter), and importantly, independent of the output length. Essentially, when $\mathsf{br} = \mathsf{br}^*$, the sampling algorithm is sampling values from a rank 1 subspace (as opposed to a rank ℓ_{out} vector space). This constrains the number of outputs to depend only on the security parameter λ rather than the output dimension ℓ_{out}.

We refer to Construction 5.6 for the formal description and Section 5 for the analysis.

Constructing a branch-constricting generator from LWE. Next, we sketch how to construct a branch-constricting generator from the LWE assumption. Here, we rely on a similar type of matrix re-randomization that was previously used in the work of [BLN+25] for building a ZAP via a hidden-bits approach.

- We take the branch set $\mathfrak{B}$ to be a collection of tuples of the form $(i_1, \ldots, i_N, z_1, \ldots, z_N)$ where $i_1, \ldots, i_N \in [k]$ are a collection of indices (over a range $[k]$) and $z_1, \ldots, z_N \in \mathbb{Z}_q$ are scalars. We view the branch as a *sparse* encoding of a row vector $\mathbf{b}\mathbf{r}^\intercal \in \mathbb{Z}_q^{kN}$ with N non-zero values. The indices of the non-zero elements in $\mathbf{b}\mathbf{r}^\intercal$ are determined by $i_1, \ldots, i_N \in [k]$ (see Construction 6.7 for the precise mapping) and the values at the non-zero indices are given by $z_1, \ldots, z_N \in \mathbb{Z}_q$. We map a branch br to a block diagonal matrix $\mathbf{H}_{\mathsf{br}}$ of the form

$$\mathbf{H}_{\mathsf{br}} = \begin{bmatrix} \mathbf{b}\mathbf{r}^\intercal & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{b}\mathbf{r}^\intercal & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{b}\mathbf{r}^\intercal \end{bmatrix} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m},$$

  where $m = kN\ell_{\mathsf{out}}$.

- To sample the public parameters associated with a branch $\mathsf{br}^* \in \mathfrak{B}$ and output length ℓ_{out}, we first sample matrices $\mathbf{S} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{\ell_{\mathsf{out}} \times n}$, $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$ as well as an error term $\mathbf{E} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$ from a discrete Gaussian distribution. We then compute the public parameters as $\mathsf{pp} = \mathbf{C} = \mathbf{S}\mathbf{A} + \mathbf{E} - \mathbf{H}_{\mathsf{br}^*}$. Here, n is the lattice dimension (for the LWE assumption).

- Given the public parameters $\mathsf{pp} = \mathbf{C}$ and a branch $\mathsf{br} \in \mathfrak{B}$, the sample algorithm samples a vector $\mathbf{r} \in \mathbb{Z}_q^m$ from a discrete Gaussian distribution and then computes $(\mathbf{C} + \mathbf{H}_{\mathsf{br}})\mathbf{r}$. It then rounds each element of the vector to a bit to obtain an ℓ_{out}-length bit string $y = \mathsf{Round}((\mathbf{C} + \mathbf{H}_{\mathsf{br}})\mathbf{r})$. The proof is the vector $\mathbf{r}$.

- The verification algorithm simply checks that $\mathbf{r}$ is low-norm and that $y = \mathsf{Round}((\mathbf{C} + \mathbf{H}_{\mathsf{br}})\mathbf{r})$.

We briefly sketch why the construction satisfies each of the required properties:

- Mode indistinguishability follows from LWE. Specifically, under the LWE assumption, the matrix $\mathbf{S}\mathbf{A} + \mathbf{E}$ is indistinguishable from a random matrix $\mathbf{C}$ where $\mathbf{C} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. Thus, the public parameters are computationally indistinguishable from a matrix of uniform random elements.

- For target randomness, take *any* matrix $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. We show the target randomness property in two main steps:

  - First, we show that for all but a negligible fraction of branches $\mathsf{br} \in \mathfrak{B}$ the matrix $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$ has a low-norm gadget trapdoor [MP12]. This is the main technical part of the proof.
  - Then, by appealing to [GPV08], we can argue that for any matrix $\mathbf{C}_{\mathsf{br}}$ with a low-norm gadget trapdoor, the distribution of $\mathbf{C}_{\mathsf{br}}\mathbf{r}$ is statistically close to uniform when $\mathbf{r}$ is sampled from a (sufficiently-wide) discrete Gaussian.

  This argument extends straightforwardly to an explicit Target algorithm that uses the gadget trapdoor to sample a preimage; we refer to Theorem 6.13 for the full details.

- Finally, we need to argue that if $\mathsf{pp} = \mathbf{C}$ is output by Setup for branch br^*, then there are a small number of values $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ where $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}^*, y, \cdot)$ accepts. On the target branch, the verification algorithm only accepts if $y = \mathsf{Round}((\mathbf{C}+\mathbf{H}_{\mathsf{br}^*})\mathbf{r})$. Since the setup algorithm sets $\mathbf{C} = \mathbf{SA}+\mathbf{E}-\mathbf{H}_{\mathsf{br}^*}$, this means $\mathbf{C}+\mathbf{H}_{\mathsf{br}^*} = \mathbf{SA}+\mathbf{E}$, and correspondingly, the verification algorithm accepts only if $y = \mathsf{Round}((\mathbf{SA}+\mathbf{E})\mathbf{r})$.

  For simplicity, ignore the error term $\mathbf{E}$. The key observation is that the mapping $\mathbf{r} \in \mathbb{Z}_q^m \mapsto \mathbf{Ar} \in \mathbb{Z}_q^n$ is compressing. Specifically since $\mathbf{Ar} \in \mathbb{Z}_q^n$, the maximum number of values of $\mathsf{Round}(\mathbf{SAr})$ is $q^n = 2^{n \log q}$. Since $n \log q = \mathsf{poly}(\lambda)$, this means the set of possible values y is also bounded by $2^{\mathsf{poly}(\lambda)}$. This is independent of the output length ℓ_{out}, as required. In the proof of Theorem 6.24, we give a more rigorous analysis that also takes the error $\mathbf{E}$ into account. There, we rely on the fact that the error $\mathbf{E}$ and the opening $\mathbf{r}$ are low-norm, and therefore, can be accounted for using a simple rounding mechanism.

We refer to Construction 6.7 for the formal description and Section 6 for the analysis.

2 Preliminaries

We write λ to denote the security parameter. We write $\mathsf{poly}(\lambda)$ to denote a polynomial in the security parameter λ. We say a function $f(\lambda)$ is negligible in λ if $f(\lambda) = o(\lambda^{-c})$ for all $c \in \mathbb{N}$ and denote this by writing $f(\lambda) = \mathsf{negl}(\lambda)$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We say two distribution ensembles $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{D}_{0,\lambda}] - \Pr[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{D}_{1,\lambda}]| = \mathsf{negl}(\lambda).$$

We say that $\mathcal{D}_0$ and $\mathcal{D}_1$ are statistically indistinguishable if their statistical distance is $\mathsf{negl}(\lambda)$ and that they are identical if their statistical distance is identically zero. Finally, we recall Hoeffding's inequality:

Lemma 2.1 (Hoeffding's Inequality)**.** *Let $X_1, \ldots, X_n \in \{0,1\}$ be independent random variables where $\Pr[X_i = 1] \geq p$ for all $i \in [n]$. Then for every $\varepsilon > 0$,*

$$\Pr\left[\sum_{i\in[n]} X_i < n(p-\varepsilon)\right] \leq e^{-2\varepsilon^2 n}.$$

Non-uniform algorithms. We model an *efficient non-uniform algorithm* $\mathcal{A}$ for inputs of length $n = n(\lambda)$ as a pair of algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ is a (possibly unbounded) algorithm that takes as input 1^λ and outputs an advice string $a_{\lambda,\mathcal{A}}$ of length $\mathsf{poly}(\lambda)$, and $\mathcal{A}_2$ is an *efficient* algorithm that takes as input the advice $a_{\lambda,\mathcal{A}}$ and the input x. Specifically, for all $\lambda \in \mathbb{N}$ and all inputs $x \in \{0,1\}^{n(\lambda)}$, we define the output $\mathcal{A}(1^\lambda, x)$ to be $\mathcal{A}(1^\lambda, x) \coloneqq \mathcal{A}_2(\mathcal{A}_1(1^\lambda), x)$. We often refer to $\mathcal{A}_1$ as the *preprocessing* algorithm and $\mathcal{A}_2$ as the *online* algorithm.

Super-polynomial hardness. Our construction will rely on super-polynomial hardness assumptions, so we will formulate some of our security definitions using (t,ε)-notation. Generally, we say that a primitive is (t,ε)-secure, if for all adversaries $\mathcal{A}$ running in time at most $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the adversary's advantage is bounded by $\varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$. We say a primitive is polynomially-secure if it is $(1, \mathsf{negl}(\lambda))$-secure for some negligible function $\mathsf{negl}(\cdot)$. We say it is quasi-polynomially-secure if it is $(1, 2^{-\log^c \lambda})$-secure for some constant $c > 1$. We say that it is sub-exponentially-secure if it is $(1, 2^{-\lambda^\varepsilon})$-secure for some constant $\varepsilon \in (0,1)$. Finally, we say that it is exponentially-secure if it is $(1, 2^{-c\lambda})$-secure for some constant $c > 0$.

2.1 Hash Functions and Randomness Extraction

Next, we recall some properties on hash functions and randomness extractors that we use in this work. Let $\mathcal{D}$ be a distribution over a finite set $\mathcal{X}$. The min-entropy of $\mathcal{D}$ is defined to be $H_\infty(\mathcal{D}) = -\log \max_{x\in\mathcal{X}} \mathcal{D}(x)$.

Definition 2.2 (Uniformity of Hash Functions)**.** Let $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of hash functions indexed by a security parameter where $\mathcal{H}_\lambda = \{H_\lambda : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}$. Then $\mathcal{H}$ satisfies statistical uniformity if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the statistical distance between the following two distribution ensembles is $\mathsf{negl}(\lambda)$:

$$\left\{(H_\lambda, H_\lambda(x)) : H_\lambda \xleftarrow{R} \mathcal{H}_\lambda, x \xleftarrow{R} \mathcal{X}_\lambda\right\}_{\lambda\in\mathbb{N}} \quad \text{and} \quad \left\{(H_\lambda, y_\lambda) : H_\lambda \xleftarrow{R} \mathcal{H}_\lambda, y_\lambda \xleftarrow{R} \mathcal{Y}_\lambda\right\}_{\lambda\in\mathbb{N}}.$$

When these two distributions are identical, then $\mathcal{H}$ satisfies perfect uniformity.

Definition 2.3 (Universal Hash Function)**.** A family of hash functions $\mathcal{H} = \left\{H : \mathcal{X} \to \{0,1\}^\ell\right\}$ is universal if for all $x_1 \neq x_2 \in \mathcal{X}$, we have that $\Pr[H(x_1) = H(x_2) : H \xleftarrow{R} \mathcal{H}] \leq 2^{-\ell}$.

Lemma 2.4 (Leftover Hash Lemma [HILL99])**.** *Let λ be a security parameter. Let $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of hash functions indexed by a security parameter where $\mathcal{H}_\lambda = \left\{H_\lambda : \mathcal{X}_\lambda \to \{0,1\}^{\ell(\lambda)}\right\}$ where each $\mathcal{H}_\lambda$ is a universal family of hash functions. Let $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of distributions where $\mathcal{D}_\lambda$ is a distribution over $\mathcal{X}_\lambda$ and moreover, there exists a function $\delta(\lambda) = \omega(\log\lambda)$ such that for all $\lambda \in \mathbb{N}$, $H_\infty(\mathcal{D}_\lambda) \geq \ell(\lambda) + \delta(\lambda)$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the statistical distance between the following distributions is $\mathsf{negl}(\lambda)$:*

$$\left\{(H_\lambda, H_\lambda(x)) : H_\lambda \xleftarrow{R} \mathcal{H}_\lambda, x \leftarrow \mathcal{D}_\lambda\right\}_{\lambda\in\mathbb{N}} \quad \textit{and} \quad \left\{(H_\lambda, u) : H_\lambda \xleftarrow{R} \mathcal{H}_\lambda, u \xleftarrow{R} \{0,1\}^{\ell(\lambda)}\right\}_{\lambda\in\mathbb{N}}.$$

Next, we state two simple corollaries of the leftover hash lemma (Lemma 2.4) that will be useful in our analysis. The first corresponds to the case where the same hash function is used to hash polynomially-many independent instances. This follows from Lemma 2.4 via a standard hybrid argument. The second is that universal hash families over a sufficiently large domain $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ satisfy statistical uniformity (Definition 2.2). This follows by taking the distribution $\mathcal{D}_\lambda$ in Lemma 2.4 to be the uniform distribution over $\mathcal{X}_\lambda$.

Corollary 2.5 (Leftover Hash Lemma for Multiple Instances)**.** *Let λ be a security parameter. Let $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of hash functions indexed by a security parameter where $\mathcal{H}_\lambda = \left\{H_\lambda : \mathcal{X}_\lambda \to \{0,1\}^{\ell(\lambda)}\right\}$ where each $\mathcal{H}_\lambda$ is a universal family of hash functions. Let $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of distributions where $\mathcal{D}_\lambda$ is a distribution over $\mathcal{X}_\lambda$ and moreover, there exists a function $\delta(\lambda) = \omega(\log\lambda)$ such that for all $\lambda \in \mathbb{N}$, $H_\infty(\mathcal{D}_\lambda) \geq \ell(\lambda) + \delta(\lambda)$. Then, for all polynomials $n = n(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the statistical distance between the following distributions is $\mathsf{negl}(\lambda)$:*

$$\left\{(H_\lambda, H_\lambda(x_1), \ldots, H_\lambda(x_n)) : \begin{array}{c} H_\lambda \xleftarrow{R} \mathcal{H}_\lambda, \\ x_1, \ldots, x_n \leftarrow \mathcal{D}_\lambda \end{array}\right\}_{\lambda\in\mathbb{N}} \quad \textit{and} \quad \left\{(H_\lambda, u_1, \ldots, u_n) : \begin{array}{c} H_\lambda \xleftarrow{R} \mathcal{H}_\lambda, \\ u_1 \ldots, u_n \xleftarrow{R} \{0,1\}^{\ell(\lambda)} \end{array}\right\}_{\lambda\in\mathbb{N}}.$$

Corollary 2.6 (Universal Hash Functions are Statistically Uniform)**.** *Let $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of hash functions where each $\mathcal{H}_\lambda = \left\{H_\lambda : \mathcal{X}_\lambda \to \{0,1\}^{\ell(\lambda)}\right\}$ is universal. Suppose there exists a function $\delta(\lambda) = \omega(\log\lambda)$ such that for all $\lambda \in \mathbb{N}$, $\log|\mathcal{X}_\lambda| \geq \ell(\lambda) + \delta(\lambda)$. Then $\mathcal{H}$ satisfies statistical uniformity.*

2.2 Cryptographic Primitives

We now recall the main cryptographic primitives we use in this work.

Non-interactive zero-knowledge arguments. First, we recall the notion of a non-interactive zero-knowledge argument for NP [BFM88, FLS90]. In this work, we only need to rely on the weaker property of witness-indistinguishability (which is implied by the standard definition of zero knowledge [FLS90, Lemma 3.4]). For ease of exposition, we focus exclusively on witness-indistinguishability in the following definition (and in this paper as a whole).

Definition 2.7 (Non-Interactive Zero-Knowledge Argument)**.** Let λ be a security parameter. Let $\mathcal{R}$ be an NP relation defined by a family of polynomial-size circuits $C = \left\{C_n : \{0,1\}^n \times \{0,1\}^{h(n)} \to \{0,1\}\right\}_{n\in\mathbb{N}}$, where n is the instance length and $h = h(n)$ is the witness length. Let $\mathcal{L}$ be the associated NP language. A non-interactive zero-knowledge argument for $\mathcal{R}$ in the uniform random string model with random string length $\sigma = \sigma(\lambda, n)$ is a tuple of efficient algorithms $\Pi_{\mathsf{NIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^n) \to \mathsf{crs}$: On input the security parameter λ and the statement length n, the setup algorithm outputs a common reference string $\mathsf{crs} \in \{0,1\}^\sigma$.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: On input the common reference string $\mathsf{crs} \in \{0,1\}^\sigma$, a statement $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^h$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: On input the common reference string $\mathsf{crs} \in \{0,1\}^\sigma$, a statement $x \in \{0,1\}^n$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

We require that Π_{NIZK} satisfies the following properties:

- **Uniform random string:** For all $\lambda, n \in \mathbb{N}$ the following distributions are identical:

$$\left\{\mathsf{crs} : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)\right\} \quad \text{and} \quad \left\{\mathsf{crs} : \mathsf{crs} \xleftarrow{R} \{0,1\}^{\sigma(\lambda,n)}\right\}$$

- **Completeness:** For all $\lambda \in \mathbb{N}$, and all $(x, w) \in \mathcal{R}$, it holds that

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 : \begin{array}{c}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|x|}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\end{array}\right] = 1.$$

- **Soundness:** For a security parameter λ and an adversary $\mathcal{A}$, we define the adaptive soundness game as follows:

  1. On input the security parameter 1^λ, the adversary sends a statement length 1^n to the challenger.
  2. The challenger runs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ and sends crs to the adversary.
  3. The adversary responds with a statement $x \in \{0,1\}^n$ and a proof π.
  4. The challenger outputs 1 if $x \notin \mathcal{L}$ and $\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1$. Otherwise it outputs 0.

  We say that Π_{NIZK} satisfies (t, ε)-*adaptive soundness* if for all adversaries $\mathcal{A}$ running in time at most $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[b = 1] = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

  in the adaptive soundness game. We also define a non-adaptive soundness game where the adversary outputs the statement $x \in \{0,1\}^n$ at the beginning of the security game (*before* it sees the common reference string). We then say that Π_{NIZK} satisfies (t, ε)-*non-adaptive soundness* if for all adversaries $\mathcal{A}$ running in time at most $t(\lambda) \cdot \mathsf{poly}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[b = 1] = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

  in the non-adaptive soundness game.

- **Non-adaptive witness indistinguishability:** For a security parameter λ, an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the non-adaptive witness-indistinguishability game as follows:

  1. On input the security parameter 1^λ, the adversary outputs a statement $x \in \{0,1\}^n$ and two witnesses $w_0, w_1 \in \{0,1\}^h$ to the challenger.
  2. The challenger checks that $\mathcal{R}(x, w_0) = 1 = \mathcal{R}(x, w_1)$. If not, the challenger outputs 0. Otherwise, the challenger runs $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ and responds with $(\mathsf{crs}, \mathsf{Prove}(\mathsf{crs}, x, w_b))$.
  3. Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.

  We say Π_{NIZK} satisfies *non-adaptive witness-indistinguishability* if for all polynomials $n = n(\lambda)$, and every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda) \tag{2.1}$$

  in the non-adaptive witness-indistinguishability game. We say Π_{NIZK} satisfies *statistical* non-adaptive witness-indistinguishability if Eq. (2.1) holds for *all* adversaries $\mathcal{A}$.

Remark 2.8 (Non-Adaptive Hardness against Non-Uniform Adversaries)**.** In the non-adaptive soundness (resp., witness indistinguishability) security game for a NIZK, the adversary has to commit to the statement x (resp., the statement x and the witnesses w_0, w_1) at the beginning of the security game. When we consider hardness against a non-uniform non-adaptive adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we can assume without loss of generality that the preprocessing algorithm $\mathcal{A}_1$ includes the statement x (resp., the statement x and the witnesses w_0, w_1) as part of its advice string $\mathsf{st}_{\mathcal{A}}$ and the online algorithm $\mathcal{A}_2$ would then be invoked on $\mathsf{st}_{\mathcal{A}}$ and crs (resp., $\mathsf{st}_{\mathcal{A}}$, crs, and π). We adopt this convention throughout this paper.

Complexity leveraging. Next, we state a standard security-parameter-scaling lemma (i.e., complexity leveraging) that we will use to simplify the exposition. We give the proof in Appendix A.

Lemma 2.9 (Complexity Leveraging)**.** *Suppose Π_{NIZK} is a NIZK that satisfies completeness, witness indistinguishability, and $(1, 2^{-\lambda^{\varepsilon_s}})$-non-adaptive soundness for constant $\varepsilon_s \in (0,1)$ in the common random string model. Then for every polynomial $p(\lambda)$, there exists a NIZK argument Π'_{NIZK} that satisfies completeness, witness-indistinguishability and $(1, 2^{-p(\lambda)})$-non-adaptive soundness in the common random string model.*

ZAPs. We now recall the formal notion of a ZAP [DN00]. A ZAP is a two-message witness-indistinguishable proof (or argument system). By default, a ZAP is *public coin*, which means the verifier's initial message is a uniform random string. Moreover, we require witness-indistinguishability to hold even if the verifier chooses its message maliciously.

Definition 2.10 (ZAP Argument)**.** Let λ be a security parameter. Let $\mathcal{R}$ be an NP relation defined by a family of polynomial-size circuits $C = \left\{C_n \colon \{0,1\}^n \times \{0,1\}^{h(n)} \to \{0,1\}\right\}_{n \in \mathbb{N}}$, where n is the instance length and $h = h(n)$ is the witness length. Let $\mathcal{L}$ be the associated NP language. A ZAP argument for $\mathcal{R}$ with public parameter size $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, n)$ is a pair of efficient algorithms $\Pi_{\mathsf{ZAP}} = (\mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Prove}(1^\lambda, \mathsf{pp}, x, w) \to \pi$: On input the security parameter λ, the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, a statement $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^h$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{pp}, x, \pi) \to b$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, a statement $x \in \{0,1\}^n$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

We require that Π_{ZAP} satisfies the following properties:

- **Completeness:** For all $\lambda \in \mathbb{N}$, and all $(x, w) \in \mathcal{R}$, it holds that

$$\Pr\left[\mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 : \begin{array}{c} \mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}(\lambda, |x|)} \\ \pi \leftarrow \mathsf{Prove}(1^\lambda, \mathsf{pp}, x, w) \end{array}\right] = 1.$$

- **Soundness:** For a security parameter λ and an adversary $\mathcal{A}$, we define the adaptive soundness game as follows:
  1. On input the security parameter 1^λ, the adversary sends the statement length 1^n to the challenger.
  2. The challenger responds with the public parameters $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}(\lambda, n)}$
  3. The adversary responds with a statement x and proof π.
  4. The challenger outputs 1 if $x \notin \mathcal{L}$ and $\mathsf{Verify}(\mathsf{pp}, x, \pi) = 1$. Otherwise it outputs 0.

  We say that Π_{ZAP} satisfies *adaptive soundness* if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[b = 1] = \mathsf{negl}(\lambda)$$

  in the adaptive soundness game. We also define a non-adaptive soundness game where the adversary outputs the statement $x \in \{0,1\}^n$ at the beginning of the security game (*before* it sees the public parameters). We say that Π_{ZAP} satisfies *non-adaptive soundness* if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the non-adaptive soundness game.

- **Witness-indistinguishability:** For a security parameter λ, an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the witness-indistinguishability game as follows:

  1. On input the security parameter 1^λ, the adversary outputs a statement $x \in \{0,1\}^n$, two witnesses $w_0, w_1 \in \{0,1\}^h$, and the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}(\lambda,n)}$.
  2. The challenger checks that $\mathcal{R}(x, w_0) = 1 = \mathcal{R}(x, w_1)$. If not, the challenger outputs 0. Otherwise, the challenger responds with $\mathsf{Prove}(1^\lambda, \mathsf{pp}, x, w_b)$.
  3. Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.

  We say Π_{ZAP} satisfies witness-indistinguishability if for all polynomials $n = n(\lambda)$, and every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda) \tag{2.2}$$

  in the witness-indistinguishability game. We say Π_{ZAP} satisfies statistical witness indistinguishability if Eq. (2.2) holds for *all* adversaries $\mathcal{A}$.

Remark 2.11 (Non-Adaptive Hardness against Non-Uniform Adversaries)**.** We follow the same conventions as Remark 2.8 when defining non-adaptive soundness against a non-uniform adversary (i.e., where we assume the preprocessing algorithm chooses the statement). Similarly, when defining witness indistinguishability against a non-uniform adversary, we assume without loss of generality that the preprocessing algorithm includes the statement x, the witnesses w_0, w_1, and the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ as part of the advice string $\mathsf{st}_{\mathcal{A}}$ it outputs.

3 ZAPs from Sometimes-Constricting Generators

In this section, we introduce the notion of a sometimes-constricting generator and then show how a sometimes-constricting generator can be used to generically compile a NIZK argument (in the common random string model) to a ZAP. We refer to Section 1.1 for an overview of the syntax of a sometimes-constricting generator and how it can be used to generically lift any NIZK argument in the common random string model to a ZAP.

Definition 3.1 (Sometimes-Constricting Generator)**.** Let λ be a security parameter and ℓ_{out} be an output length. A public-coin sometimes-constricting generator with public parameter size $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}})$ is a triple of efficient algorithms $\Pi_{\mathsf{SCG}} = (\mathsf{Setup}, \mathsf{Sample}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}) \to (\mathsf{pp}, t^*)$: On input the security parameter λ and the output length ℓ_{out}, the setup algorithm outputs a set of public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and a guess $t^* \in \{0,1\}^{\ell_{\mathsf{out}}}$. Throughout, we will assume that the public parameters pp always implicitly include a description of 1^λ and $1^{\ell_{\mathsf{out}}}$.
- $\mathsf{Sample}(\mathsf{pp}) \to (y, \pi)$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, the sampling algorithm outputs a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ and a proof π.
- $\mathsf{Verify}(\mathsf{pp}, y, \pi) \to \{0,1\}$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.

We require Π_{SCG} to satisfy the following properties:

- **Completeness:** For all $\lambda, \ell_{\mathsf{out}} \in \mathbb{N}$, all $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, and all (y, π) in the support of $\mathsf{Sample}(\mathsf{pp})$, it holds that $\mathsf{Verify}(\mathsf{pp}, y, \pi) = 1$.
- **Mode indistinguishability:** For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the mode-indistinguishability game as follows:
  - On input 1^λ, the adversary sends $1^{\ell_{\mathsf{out}}}$ to the challenger.
  - If $b = 0$, the challenger samples $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$. If $b = 1$, the challenger samples $(\mathsf{pp}, t^*) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}})$. The challenger sends pp to the adversary.

– Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.

We say that Π_{SCG} satisfies mode indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda) \tag{3.1}$$

in the mode-indistinguishability game.

- **μ-Guessing security:** For a security parameter λ and an adversary $\mathcal{A}$, we define the following game:
  - On input 1^λ, the adversary sends $1^{\ell_{\mathsf{out}}}$ to the challenger.
  - The challenger samples $(\mathsf{pp}, t^*) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}})$ and sends pp to the adversary $\mathcal{A}$.
  - The adversary $\mathcal{A}$ either aborts with output $\perp$ or it outputs a pair (y, π). The output of the experiment is 1 if $y = t^*$ and $\mathsf{Verify}(\mathsf{pp}, y, \pi) = 1$ and 0 otherwise.

  Let $E_\perp$ be the event where $\mathcal{A}$ either outputs $\perp$ or outputs a pair (y, π) where $\mathsf{Verify}(\mathsf{pp}, y, \pi) = 0$. Let E_{guessed} be the event that the output of the experiment is 1. We say Π_{SCG} satisfies μ-guessing security if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\lambda \in \mathbb{N}$,

$$\Pr[E_{\mathsf{guessed}}] \geq \mu(\lambda) \cdot (\Pr[\neg E_\perp] - \mathsf{negl}(\lambda)) \,. \tag{3.2}$$

- **Target randomness:** There exists a pair of algorithms $(\mathsf{Preprocess}, \mathsf{Target})$ with the following syntax:
  - $\mathsf{Preprocess}(\mathsf{pp}) \to \mathsf{td}$: On input the public parameter $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ (which includes an implicit description of 1^λ and $1^{\ell_{\mathsf{out}}}$), the preprocess algorithm outputs a trapdoor td.
  - $\mathsf{Target}(\mathsf{td}, y) \to \pi$: On input a trapdoor td and a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$, the target algorithm outputs a proof π.

  We require that $(\mathsf{Preprocess}, \mathsf{Target})$ satisfy the following two properties:
  - **Efficiency:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, \ell_{\mathsf{out}} \in \mathbb{N}$, all $\mathsf{pp} \in \{0,1\}^*$, and all td in the support of $\mathsf{Preprocess}(\mathsf{pp})$, we have that $|\mathsf{td}| \leq \mathsf{poly}(\lambda + \ell_{\mathsf{out}} + |\mathsf{pp}|)$. In addition, we require that Target be efficiently-computable. Note that we do *not* require that Preprocess be efficiently-computable.
  - **Target randomness:** For a security parameter λ, an adversary $\mathcal{A}$, an output length ℓ_{out}, and a bit $b \in \{0,1\}$, we define the target randomness game as follows:
    * On input the security parameter 1^λ and the output length $1^{\ell_{\mathsf{out}}}$, the adversary $\mathcal{A}$ sends pp to the challenger.
    * The challenger computes $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}})$ and checks that $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$. Otherwise, it halts with output 0.
      · If $b = 0$, the challenger samples $(y, \pi) \leftarrow \mathsf{Sample}(\mathsf{pp})$.
      · If $b = 1$, the challenger samples $y \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$ and computes $\mathsf{td} \leftarrow \mathsf{Preprocess}(\mathsf{pp})$ and $\pi \leftarrow \mathsf{Target}(\mathsf{td}, y)$.

      The challenger sends (y, π) to the adversary.
    * Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.

    We say that Π_{SCG} satisfies target randomness if for all (possibly unbounded) adversaries $\mathcal{A}$ and all polynomials $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$

    in the target randomness game.

ZAP from NIZK using a sometimes-constricting generator. We now show how to generically compile any NIZK argument in the common random string model into a ZAP. We give our construction below and refer to Section 1.1 for an overview of the transformation.

Construction 3.2 (ZAP from NIZK Argument and Sometimes-Constricting Generator)**.** Let λ be a security parameter. Let $\mathcal{R}$ be an NP relation, and let $C = \{C_n : \{0,1\}^n \times \{0,1\}^{h(n)} \to \{0,1\}\}_{n\in\mathbb{N}}$ be the family of polynomial-size circuits that compute $\mathcal{R}$. Let $\mathcal{L}$ be the associated NP language. Our construction relies on the following building blocks:

- Let $\Pi_{\mathsf{SCG}} = (\mathsf{SCG.Setup}, \mathsf{SCG.Sample}, \mathsf{SCG.Verify})$ be a public-coin sometimes-constricting generator with μ-guessing advantage. Let $\ell_{\text{scg-pp}}(\lambda, \ell_{\text{out}})$ be the public parameter size for Π_{SCG}.
- Let $\Pi_{\mathsf{NIZK}} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ be a NIZK argument for $\mathcal{R}$ in the common random string model satisfying witness-indistinguishability and $(1, \mu(\lambda))$-non-adaptive soundness (where μ is the same function as that for the μ-guessing advantage for Π_{SCG}). Let $\sigma(\lambda, n)$ be the length of the common random string associated with Π_{NIZK}.

We construct a ZAP $\Pi_{\mathsf{ZAP}} = (\mathsf{Prove}, \mathsf{Verify})$ with public parameter size $\ell_{\mathsf{pp}}(\lambda, n) = \sigma(\lambda, n) + \ell_{\text{scg-pp}}(\lambda, \sigma(\lambda, n))$ as follows:

- $\mathsf{Prove}(1^\lambda, \mathsf{pp}, x, w)$: On input the security parameter λ, the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, the statement $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^h$, the prove algorithm proceeds as follows:
  - Let $\sigma = \sigma(\lambda, n)$ and $\ell_{\text{scg-pp}} = \ell_{\text{scg-pp}}(\lambda, \sigma)$.
  - Parse $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$ where $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\text{scg-pp}}}$ and $z \in \{0,1\}^\sigma$.
  - Compute $(y, \pi_{\mathsf{SCG}}) \leftarrow \mathsf{SCG.Sample}(\mathsf{pp}_{\mathsf{SCG}})$ and $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(y \oplus z, x, w)$.
  - Output $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$.
- $\mathsf{Verify}(\mathsf{pp}, x, \pi)$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, the statement $x \in \{0,1\}^n$, and the proof $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$, the verification algorithm parses $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$ where $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\text{scg-pp}}}$ and $z \in \{0,1\}^\sigma$. Then, it checks that $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$ and that $\mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}})$. If the checks pass, it outputs 1; otherwise, it outputs 0.

Theorem 3.3 (Completeness)**.** *If Π_{SCG} is complete and Π_{NIZK} is complete in the common random string model, then Construction 3.2 is complete.*

Proof. Take any $\lambda \in \mathbb{N}$ and any $(x, w) \in \mathcal{R}$ where $|x| = n$. Take any $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and let $\pi \leftarrow \mathsf{Prove}(1^\lambda, \mathsf{pp}, x, w)$. Parse $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$ where $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\text{scg-pp}}}$ and $z \in \{0,1\}^\sigma$. Next, by construction of Prove, we can write $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ where $(y, \pi_{\mathsf{SCG}}) \leftarrow \mathsf{SCG.Sample}(\mathsf{pp}_{\mathsf{SCG}})$ and $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(y \oplus z, x, w)$. We now consider the two verification checks:

- By completeness of Π_{SCG}, $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$ so the first verification check passes.
- Since the CRS output by $\mathsf{NIZK.Setup}(1^\lambda, 1^n)$ is uniformly random, the value $y \oplus z$ lies in the support of $\mathsf{NIZK.Setup}(1^\lambda, 1^n)$. Then, by completeness of Π_{NIZK}, this means $\mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1$ and the second verification check passes.

Since both verification checks pass, completeness holds. □

Theorem 3.4 (Non-Adaptive Soundness)**.** *Suppose Π_{NIZK} is $(1, \mu(\lambda))$-non-adaptively sound against non-uniform adversaries and Π_{SCG} satisfies mode-indistinguishability and $\mu(\lambda)$-guessing security against non-uniform adversaries. Then Construction 3.2 is non-adaptively sound against non-uniform adversaries.*

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a non-uniform adversary for the non-adaptive soundness game, where $\mathcal{A}_1$ is the offline algorithm that takes as input the security parameter 1^λ and outputs a polynomial-size advice string $\mathsf{st}_{\mathcal{A}}$. We assume that $\mathsf{st}_{\mathcal{A}}$ includes a description of the statement $x \in \{0,1\}^n$ (see Remark 2.11). Algorithm $\mathcal{A}_2$ is the online algorithm that takes as input the advice string $\mathsf{st}_{\mathcal{A}}$, the public parameters pp, and outputs the proof π (for the statement x). We now define a sequence of hybrid experiments:

- Hyb_1: This is the non-adaptive soundness game for Construction 3.2.
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}_1$ outputs an advice string $\mathsf{st}_{\mathcal{A}}$, which includes a description of the statement $x \in \{0,1\}^n$.
  2. The challenger samples $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$ and invokes $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{pp})$. Algorithm $\mathcal{A}_2$ outputs $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$.
  3. The challenger parses $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$ where $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\mathsf{scg\text{-}pp}}}$ and $z \in \{0,1\}^\sigma$. The challenger checks that $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$ and outputs 0 if not. The output of the experiment is 1 if
  $$x \notin \mathcal{L} \quad \text{and} \quad \mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1 \quad \text{and} \quad \mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1.$$
- Hyb_2: Same as Hyb_1 except the challenger samples $\mathsf{pp}_{\mathsf{SCG}}$ using $\mathsf{SCG.Setup}$.
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}_1$ outputs an advice string $\mathsf{st}_{\mathcal{A}}$, which includes a description of the statement $x \in \{0,1\}^n$.
  2. The challenger samples $(\mathsf{pp}_{\mathsf{SCG}}, t^*) \leftarrow \mathsf{SCG.Setup}(1^\lambda, 1^\sigma)$. Next, it samples $z \xleftarrow{R} \{0,1\}^\sigma$ and sets $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$. Then, it invokes $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{pp})$. Algorithm $\mathcal{A}_2$ outputs $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$.
  3. The output of the experiment is 1 if
  $$x \notin \mathcal{L} \quad \text{and} \quad \mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1 \quad \text{and} \quad \mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1.$$
- Hyb_3: Same as Hyb_2, except the challenger only outputs 1 if $y = t^*$.
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}_1$ outputs an advice string $\mathsf{st}_{\mathcal{A}}$, which includes a description of the statement $x \in \{0,1\}^n$.
  2. The challenger samples $(\mathsf{pp}_{\mathsf{SCG}}, t^*) \leftarrow \mathsf{SCG.Setup}(1^\lambda, 1^\sigma)$. Next, it samples $z \xleftarrow{R} \{0,1\}^\sigma$ and sets $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$. Then, it invokes $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{pp})$. Algorithm $\mathcal{A}_2$ outputs $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$.
  3. The output of the experiment is 1 if
  $$x \notin \mathcal{L} \quad \text{and} \quad \mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1 \quad \text{and} \quad \mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1 \quad \text{and} \quad t^* = y.$$
- Hyb_4: Same as Hyb_3, except the challenger programs t^* into the CRS as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}_1$ outputs an advice string $\mathsf{st}_{\mathcal{A}}$, which includes a description of the statement $x \in \{0,1\}^n$.
  2. The challenger samples $(\mathsf{pp}_{\mathsf{SCG}}, t^*) \leftarrow \mathsf{SCG.Setup}(1^\lambda, 1^\sigma)$. Next, it samples $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^n)$ and sets $z = \mathsf{crs} \oplus t^*$. Finally, it sets $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$. Then, it invokes $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{pp})$. Algorithm $\mathcal{A}_2$ outputs $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$.
  3. The output of the experiment is 1 if
  $$x \notin \mathcal{L} \quad \text{and} \quad \mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1 \quad \text{and} \quad \mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1 \quad \text{and} \quad t^* = y.$$

Let $\mathsf{Hyb}_i(\mathcal{A})$ be the random variable denoting the output of an execution of experiment Hyb_i with adversary $\mathcal{A}$. We now analyze each pair of adjacent hybrid experiments. Our goal is to show $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] = \mathsf{negl}(\lambda)$.

Claim 3.5. *Suppose Π_{SCG} satisfies mode indistinguishability against non-uniform adversaries. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all security parameters $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct an efficient non-uniform adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that breaks mode indistinguishability. The preprocessing algorithm $\mathcal{B}_1$ works as follows:

1. On input the security parameter 1^λ, run $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$ and let x be the statement associated with $\mathsf{st}_{\mathcal{A}}$.

2. If $x \in \mathcal{L}$, then let $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, 1^\sigma, x, 0)$. Otherwise, let $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, 1^\sigma, x, 1)$, where $\sigma = \sigma(\lambda, |x|)$. Output $\mathsf{st}_{\mathcal{B}}$.

The online algorithm $\mathcal{B}_2$ now works as follows:

1. On input the state $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, 1^\sigma, x, b)$, output 1^σ.
2. The challenger responds with the public parameters $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\text{scg-pp}}}$.
3. Sample $z \xleftarrow{R} \{0,1\}^\sigma$ and set $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$. Run $\pi \leftarrow \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{pp})$ and parse $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$.
4. Output 1 if $b = 1$, $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$, and $\mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1$. Otherwise, output 0.

By construction, since $\mathcal{A}_2$ is efficient, algorithm $\mathcal{B}_2$ is also efficient. By construction of $\mathcal{B}$, we have $b = 0$ if $x \in \mathcal{L}$ and $b = 1$ if $x \notin \mathcal{L}$. Thus, the output of $\mathcal{B}_2$ is 1 if and only if $x \notin \mathcal{L}$, $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$, and $\mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1$. Observe that these conditions coincide with those in Hyb_1 and Hyb_2. We now consider the two possible distributions of $\mathsf{pp}_{\mathsf{SCG}}$:

- Suppose the challenger samples $\mathsf{pp}_{\mathsf{SCG}} \xleftarrow{R} \{0,1\}^{\ell_{\text{scg-pp}}}$. Since $\mathcal{B}_2$ samples $z \xleftarrow{R} \{0,1\}^\sigma$, the overall distribution of pp is uniform over $\{0,1\}^{\ell_{\mathsf{pp}}}$. This matches the distribution in Hyb_1 and correspondingly, algorithm $\mathcal{B}_2$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.
- Suppose the challenger samples $(\mathsf{pp}_{\mathsf{SCG}}, t^*) \leftarrow \mathsf{SCG.Setup}(1^\lambda, 1^\sigma)$. In this case, the distribution of pp coincides with the distribution in Hyb_2. In this case, algorithm $\mathcal{B}_2$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ breaks mode indistinguishability with the same advantage ε, and the claim holds. □

Claim 3.6. *Suppose Π_{SCG} satisfies μ-guessing security against non-uniform adversaries. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all security parameters $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \geq \mu(\lambda) \cdot (\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \mathsf{negl}(\lambda)).$$

Proof. We first use $\mathcal{A}$ to construct a non-uniform μ-guessing adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for Π_{SCG}. The preprocessing algorithm $\mathcal{B}_1$ works as follows:

1. On input the security parameter 1^λ, run $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$ and let x be the statement associated with $\mathsf{st}_{\mathcal{A}}$.
2. If $x \in \mathcal{L}$, then let $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, 1^\sigma, x, 0)$. Otherwise, let $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, 1^\sigma, x, 1)$, where $\sigma = \sigma(\lambda, |x|)$. Output $\mathsf{st}_{\mathcal{B}}$.

The online algorithm $\mathcal{B}_2$ now works as follows:

1. On input the state $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, 1^\sigma, x, b)$, output 1^σ.
2. The challenger replies with the public parameters $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\text{scg-pp}}}$.
3. Sample $z \xleftarrow{R} \{0,1\}^\sigma$ and set $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$. Run $\pi \leftarrow \mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, \mathsf{pp})$ and parse $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$.
4. Then check if $b = 1$, $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$, and $\mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1$. If so, output (y, π_{SCG}). Otherwise, output $\perp$.

By construction, since $\mathcal{A}_2$ is efficient, algorithm $\mathcal{B}_2$ is also efficient. We now analyze the μ-guessing advantage of algorithm $\mathcal{B}$. In the μ-guessing security game, the challenger samples $(\mathsf{pp}_{\mathsf{SCG}}, t^*) \leftarrow \mathsf{SCG.Setup}(1^\lambda, 1^\sigma)$. Let $E_\perp$ be the event that algorithm $\mathcal{B}_2$ outputs $\perp$ or that it outputs (y, π_{SCG}) where $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 0$. Let E_{guessed} be the event that algorithm $\mathcal{B}_2$ outputs (y, π_{SCG}) where $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$ and $y = t^*$. By definition, the distribution of pp simulated by $\mathcal{B}_2$ perfectly coincides with the distribution in Hyb_2 and Hyb_3. Thus, the following relations hold:

- Consider the event $\neg E_\perp$. First, algorithm $\mathcal{B}_2$ will never output a pair (y, π_{SCG}) where $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 0$. In addition, algorithm $\mathcal{B}_2$ will not output $\perp$ if $b = 1$, $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$, and $\mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1$. By construction of $\mathcal{B}_1$, $b = 1$ if and only if $x \notin \mathcal{L}$. Thus, event $\neg E_\perp$ occurs as long as

  $$x \notin \mathcal{L} \quad \text{and} \quad \mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1 \quad \text{and} \quad \mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1.$$

  By definition, this is precisely $\Pr[\neg E_\perp] = \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

- Next, the event E_{guessed} occurs only if

$$x \notin \mathcal{L} \quad \text{and} \quad \mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1 \quad \text{and} \quad \mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1 \quad \text{and} \quad t^* = y.$$

  By definition, this is precisely $\Pr[E_{\text{guessed}}] = \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

Since Π_{SCG} satisfies μ-guessing security, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[E_{\text{guessed}}] \geq \mu(\lambda) \cdot (\Pr[\neg E_{\perp}] - \mathsf{negl}(\lambda)).$$

Since $\Pr[E_{\text{guessed}}] = \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$ and $\Pr[\neg E_{\perp}] = \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$, we conclude that

$$\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \geq \mu(\lambda) \cdot (\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \mathsf{negl}(\lambda)).$$

The claim follows. □

Claim 3.7. *Suppose Π_{NIZK} has a uniform random string. Then, $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.*

Proof. The only difference between the two hybrids is in how $z \in \{0,1\}^{\sigma}$ is sampled. In Hyb_3, the challenger samples $z \xleftarrow{R} \{0,1\}^{\sigma}$ whereas in Hyb_4, the challenger samples $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^{\lambda}, 1^{n})$ and sets $z = \mathsf{crs} \oplus t^*$. Since Π_{NIZK} has a uniform random string, the distribution crs in Hyb_4 is uniform over $\{0,1\}^{\sigma}$ and independent of t^*. Thus, in both cases, the distribution of z is uniform over $\{0,1\}^{\sigma}$ and the two experiments are identically distributed. □

Claim 3.8. *Suppose Π_{NIZK} satisfies $(1, \mu(\lambda))$-non-adaptive soundness against non-uniform adversaries. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = \mu(\lambda) \cdot \mathsf{negl}(\lambda)$.*

Proof. Suppose $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] \geq \mu(\lambda) \cdot \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a non-adaptive non-uniform soundness adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ for Π_{NIZK}. The preprocessing algorithm $\mathcal{B}_1$ simply runs $\mathcal{A}_1$ to get the state $\mathsf{st}_{\mathcal{A}}$ and the associated statement x. Algorithm $\mathcal{B}_1$ outputs the same statement x and the advice string $\mathsf{st}_{\mathcal{B}} = \mathsf{st}_{\mathcal{A}}$. We now describe the online algorithm $\mathcal{B}_2$:

1. On input the advice string $\mathsf{st}_{\mathcal{B}} = \mathsf{st}_{\mathcal{A}}$ and the common reference string $\mathsf{crs} \in \{0,1\}^{\sigma}$, algorithm $\mathcal{B}_2$ runs $(\mathsf{pp}_{\mathsf{SCG}}, t^*) \leftarrow \mathsf{SCG.Setup}(1^{\lambda}, 1^{\sigma})$. It computes $z = \mathsf{crs} \oplus t^*$, sets $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$, and runs $\mathcal{A}_2$ on $\mathsf{st}_{\mathcal{A}}$ and the public parameters pp.
2. Algorithm $\mathcal{A}_2$ responds with a proof $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$. Algorithm $\mathcal{B}_2$ checks $\mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1$ and $y = t^*$. If both checks pass, it sends π_{NIZK} to the challenger. Otherwise it halts with output $\perp$.

First, if $\mathcal{A}_2$ is efficient, then so is $\mathcal{B}_2$. We argue that $\mathcal{B}$ perfectly simulates an execution of Hyb_4 for $\mathcal{A}$. In the reduction, the challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{|x|})$, which coincides with the distribution of crs in Hyb_4. Similarly, algorithm $\mathcal{B}$ samples $(\mathsf{pp}_{\mathsf{SCG}}, t^*)$ and constructs pp exactly as described in Hyb_4. Thus, with probability $\mu(\lambda) \cdot \varepsilon(\lambda)$, algorithm $\mathcal{A}$ will output $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ such that $\mathsf{Hyb}_4(\mathcal{A}) = 1$, which means

$$x \notin \mathcal{L},\ \mathsf{SCG.Verify}(\mathsf{pp}_{\mathsf{SCG}}, y, \pi_{\mathsf{SCG}}) = 1,\ \mathsf{NIZK.Verify}(y \oplus z, x, \pi_{\mathsf{NIZK}}) = 1 \text{ and } t^* = y. \tag{3.3}$$

In particular, since algorithm $\mathcal{B}$ sets $z = \mathsf{crs} \oplus t^*$, this means that

$$y \oplus z = t^* \oplus \mathsf{crs} \oplus t^* = \mathsf{crs}.$$

In this case, Eq. (3.3) means that algorithm $\mathcal{B}$ outputs (x, π_{NIZK}) where $x \notin \mathcal{L}$ and $\mathsf{NIZK.Verify}(\mathsf{crs}, x, \pi_{\mathsf{NIZK}}) = 1$, which means it breaks non-adaptive soundness of Π_{NIZK} with the same advantage. □

Proof of Theorem 3.4. Returning now to the proof of Theorem 3.4, we appeal to Claims 3.6 and 3.7 to conclude that there exists a negligible function $\mathsf{negl}_1(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\mu(\lambda) \cdot (\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \mathsf{negl}_1(\lambda)) \leq \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1].$$

Combined with Claim 3.8, we see that there exists a negligible function $\mathsf{negl}_2(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\mu(\lambda) \cdot (\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \mathsf{negl}_1(\lambda)) \leq \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = \mu(\lambda) \cdot \mathsf{negl}_2(\lambda).$$

This means

$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \leq \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda).$$

Finally, by Claim 3.5, there exists a negligible function $\mathsf{negl}_3(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \leq \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda) + \mathsf{negl}_3(\lambda),$$

which proves non-adaptive soundness. □

Theorem 3.9 (Witness Indistinguishability)**.** *Suppose Π_{NIZK} satisfies non-adaptive witness indistinguishability against efficient non-uniform (resp., unbounded) adversaries and Π_{SCG} satisfies target randomness. Then Construction 3.2 satisfies witness indistinguishability against efficient non-uniform (resp., unbounded) adversaries.*

Proof. Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an efficient non-uniform (resp. unbounded) adversary for the witness indistinguishability game (see Remark 2.11). Let $(\mathsf{SCG.Preprocess}, \mathsf{SCG.Target})$ be the target randomness algorithms associated with Π_{SCG}. We now define a sequence of hybrid arguments, each parameterized by a bit $b \in \{0,1\}$.

- $\mathsf{Hyb}_1^{(b)}$: This is the standard witness-indistinguishability game with bit b:
  1. On input the security parameter 1^λ, the preprocessing algorithm $\mathcal{A}_1$ outputs the advice string $\mathsf{st}_{\mathcal{A}}$. Let $x \in \{0,1\}^n$ be the statement, $w_0, w_1 \in \{0,1\}^h$ be the witnesses and $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ be the public parameters associated with $\mathsf{st}_{\mathcal{A}}$.
  2. The challenger checks that $\mathcal{R}(x, w_0) = 1 = \mathcal{R}(x, w_1)$ and that $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, |x|)$. If any check fails, the challenger halts with output 0.
  3. The challenger parses $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$ where $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\text{scg-pp}}}$ and $z \in \{0,1\}^\sigma$.
  4. The challenger constructs the proof $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ as follows:
     - Sample $(y, \pi_{\mathsf{SCG}}) \leftarrow \mathsf{SCG.Sample}(\mathsf{pp}_{\mathsf{SCG}})$.
     - Compute $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(z \oplus y, x, w_b)$.

     The challenger sends $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ to $\mathcal{A}$.
  5. Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.
- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$, except the challenger changes how it samples (y, π_{SCG}) in Step 4:
  4. The challenger constructs the proof $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ as follows:
     - Sample $y \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$.
     - Compute $\mathsf{td}_{\mathsf{SCG}} \leftarrow \mathsf{SCG.Preprocess}(\mathsf{pp}_{\mathsf{SCG}})$ and $\pi_{\mathsf{SCG}} \leftarrow \mathsf{SCG.Target}(\mathsf{td}_{\mathsf{SCG}}, y)$.
     - Compute $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(z \oplus y, x, w_b)$.

     The challenger sends $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ to $\mathcal{A}$.
- $\mathsf{Hyb}_3^{(b)}$: Same as $\mathsf{Hyb}_2^{(b)}$, except the challenger changes the distribution of y in Step 4:
  4. The challenger constructs the proof $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ as follows:
     - Sample $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^n)$ and set $y = \mathsf{crs} \oplus z$.

- Compute $\mathsf{td}_{\mathsf{SCG}} \leftarrow \mathsf{SCG.Preprocess}(\mathsf{pp}_{\mathsf{SCG}})$ and $\pi_{\mathsf{SCG}} \leftarrow \mathsf{SCG.Target}(\mathsf{td}_{\mathsf{SCG}}, y)$.
- Compute $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(z \oplus y, x, w_b)$.

The challenger sends $\pi = (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}})$ to $\mathcal{A}$.

Let $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ be the random variable denoting the output of the experiment $\mathsf{Hyb}_i^{(b)}$ with adversary $\mathcal{A}$. We now analyze each adjacent pair of hybrid experiments.

Claim 3.10. *Suppose Π_{SCG} satisfies target randomness. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Take any $b \in \{0,1\}$ and let $|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a (inefficient) reduction algorithm $\mathcal{B}$ that breaks target randomness of Π_{SCG} with the same advantage. Algorithm $\mathcal{B}$ works as follows:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$. Let $x \in \{0,1\}^n$ be the statement, $w_0, w_1 \in \{0,1\}^h$ be the witnesses and $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ be the public parameters associated with $\mathsf{st}_{\mathcal{A}}$.
2. If $\mathcal{R}(x, w_0) = 0$ or $\mathcal{R}(x, w_1) = 0$ or $\ell_{\mathsf{pp}} \neq \ell_{\mathsf{pp}}(\lambda, |x|)$, then algorithm $\mathcal{B}$ outputs 0. Otherwise, it parses $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$ where $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\text{scg-pp}}}$, $z \in \{0,1\}^\sigma$, $\sigma = \sigma(\lambda, |x|)$, and $\ell_{\text{scg-pp}} = \ell_{\text{scg-pp}}(\lambda, \sigma)$ and sends $(1^\sigma, \mathsf{pp}_{\mathsf{SCG}})$ to the challenger.
3. The challenger responds with a pair (y, π_{SCG}).
4. Algorithm $\mathcal{B}$ computes $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(z \oplus y, x, w_b)$ and sends $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}}))$ to the challenger.

Let $n = n(\lambda)$ be the length of the statement output by $\mathcal{A}_1$. By definition, this is polynomially-bounded. Consider now the target randomness security experiment with algorithm $\mathcal{B}$ and output length $\ell_{\mathsf{out}}(\lambda) = \sigma(\lambda, n(\lambda))$.

We bound the probability that algorithm $\mathcal{B}$ outputs 1 in the target randomness security experiment. First, if $\mathcal{R}(x, w_0) = 0$ or $\mathcal{R}(x, w_1) = 0$ or $\ell_{\mathsf{pp}} \neq \ell_{\mathsf{pp}}(\lambda, |x|)$, the challenger's output in $\mathsf{Hyb}_1^{(b)}$ or $\mathsf{Hyb}_2^{(b)}$ is always 0. Algorithm $\mathcal{B}$ also outputs 0 in this case. It suffices to consider the case where $\mathcal{R}(x, w_0) = 1 = \mathcal{R}(x, w_1)$ and $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, |x|)$. We argue that depending on the distribution of the tuple (y, π_{SCG}), algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(b)}$ or $\mathsf{Hyb}_2^{(b)}$ for $\mathcal{A}$:

- Suppose the challenger samples $(y, \pi_{\mathsf{SCG}}) \leftarrow \mathsf{SCG.Sample}(\mathsf{pp}_{\mathsf{SCG}})$. This is precisely how the challenger samples (y, π_{SCG}) in $\mathsf{Hyb}_1^{(b)}$. Moreover, algorithm $\mathcal{B}$ constructs π_{NIZK} using the same procedure as in $\mathsf{Hyb}_1^{(b)}$. Thus, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(b)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]$.
- Suppose the challenger samples $y \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$ and then computes π_{SCG} as $\pi_{\mathsf{SCG}} \leftarrow \mathsf{SCG.Target}(\mathsf{td}_{\mathsf{SCG}}, y)$ where $\mathsf{td}_{\mathsf{SCG}} \leftarrow \mathsf{SCG.Preprocess}(\mathsf{pp}_{\mathsf{SCG}})$. In this case, the distribution of (y, π_{SCG}) precisely coincides with its distribution in $\mathsf{Hyb}_2^{(b)}$. Moreover, algorithm $\mathcal{B}$ constructs the proof π_{NIZK} using the same procedure as in $\mathsf{Hyb}_2^{(b)}$. Thus, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_2^{(b)}$ and outputs 1 with probability $\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]$.

We conclude that the algorithm $\mathcal{B}$ breaks target randomness with the same advantage ε. □

Claim 3.11. *Suppose Π_{NIZK} has a uniform random string. Then for all $b \in \{0,1\}$,*

$$\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1].$$

Proof. Same as the proof of Claim 3.7. □

Claim 3.12. *Suppose Π_{SCG} satisfies target randomness and Π_{NIZK} satisfies non-adaptive witness indistinguishability against non-uniform (resp. unbounded) adversaries. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_3^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(1)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Recall that $\mathcal{A}$ is an efficient non-uniform (resp., unbounded) adversary for the witness indistinguishability game.

Proof. Suppose $|\Pr[\mathsf{Hyb}_3^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(1)}(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ where ε is non-negligible. We use $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct a non-uniform (resp. unbounded) algorithm $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that breaks witness indistinguishability of Π_{NIZK} with the same advantage. We show the non-uniform case below and remark about how to adapt it for unbounded adversaries in the end. The preprocessing algorithm $\mathcal{B}_1$ works as follows:

1. On input the security parameter 1^λ, run $\mathsf{st}_{\mathcal{A}} \leftarrow \mathcal{A}_1(1^\lambda)$. Let $x \in \{0,1\}^n$ be the statement, $w_0, w_1 \in \{0,1\}^h$ be the witnesses and $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ be the public parameters associated with $\mathsf{st}_{\mathcal{A}}$.
2. If $\ell_{\mathsf{pp}} \neq \ell_{\mathsf{pp}}(\lambda, |x|)$, then output $\mathsf{st}_{\mathcal{B}} = \bot$. Otherwise, parse $\mathsf{pp} = \mathsf{pp}_{\mathsf{SCG}} \| z$ where $\mathsf{pp}_{\mathsf{SCG}} \in \{0,1\}^{\ell_{\text{scg-pp}}}$, $z \in \{0,1\}^\sigma$, $\sigma = \sigma(\lambda, |x|)$, and $\ell_{\text{scg-pp}} = \ell_{\text{scg-pp}}(\lambda, \sigma)$. Compute $\mathsf{td}_{\mathsf{SCG}} \leftarrow \mathsf{SCG.Preprocess}(\mathsf{pp}_{\mathsf{SCG}})$.
3. Output $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, \mathsf{pp}_{\mathsf{SCG}}, \mathsf{td}_{\mathsf{SCG}}, z, x, w_0, w_1)$.

The online algorithm $\mathcal{B}_2$ works as follows:

1. On input $\mathsf{st}_{\mathcal{B}}$, if $\mathsf{st}_{\mathcal{B}} = \bot$, then output $\bot$. Otherwise, parse $\mathsf{st}_{\mathcal{B}} = (\mathsf{st}_{\mathcal{A}}, \mathsf{pp}_{\mathsf{SCG}}, \mathsf{td}_{\mathsf{SCG}}, z, x, w_0, w_1)$ and output (x, w_0, w_1).
2. The challenger responds with crs and a proof π_{NIZK}.
3. Let $y = \mathsf{crs} \oplus z$ and compute $\pi_{\mathsf{SCG}} \leftarrow \mathsf{SCG.Target}(\mathsf{td}_{\mathsf{SCG}}, y)$. Output $\mathcal{A}_2(\mathsf{st}_{\mathcal{A}}, (y, \pi_{\mathsf{SCG}}, \pi_{\mathsf{NIZK}}))$.

Length of advice string. First, we need to argue that $|\mathsf{st}_{\mathcal{B}}| = \mathsf{poly}(\lambda)$. Since $\mathcal{A}$ is an efficient adversary, this means $|\mathsf{st}_{\mathcal{A}}| = \mathsf{poly}(\lambda)$. As such, the lengths of pp, x, w_0, w_1 associated with $\mathsf{st}_{\mathcal{A}}$ are all polynomially-bounded. Since Π_{SCG} satisfies target randomness, the size of the trapdoor $\mathsf{td}_{\mathsf{SCG}}$ is at most $\mathsf{poly}(|\mathsf{pp}_{\mathsf{SCG}}|) = \mathsf{poly}(\ell_{\text{scg-pp}}(\lambda, \sigma(\lambda, n)))$. Since $n = \mathsf{poly}(\lambda)$, and $\sigma(\cdot,\cdot)$ and $\ell_{\text{scg-pp}}(\cdot,\cdot)$ are polynomially-bounded, this means the length of $\mathsf{td}_{\mathsf{SCG}}$ is $\mathsf{poly}(\lambda)$.

Running time of online algorithm. By target randomness of Π_{SCG}, the target algorithm Target is efficient (i.e., runs in time $\mathsf{poly}(|\mathsf{td}_{\mathsf{SCG}}| + \sigma(\lambda, n))$). Since $\sigma(\cdot,\cdot)$ is a polynomial, and n is polynomially-bounded, the running time of Target is $\mathsf{poly}(\lambda)$. Since $\mathcal{A}_2$ is efficient, we conclude that $\mathcal{B}_2$ is efficient.

Advantage analysis. To conclude the proof, we compute the advantage of $\mathcal{B}$. First, the challenger samples $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{|x|})$ which coincides with the distribution in $\mathsf{Hyb}_3^{(0)}$ and $\mathsf{Hyb}_3^{(1)}$. Next, algorithm $\mathcal{B}$ computes z, the trapdoor $\mathsf{td}_{\mathsf{SCG}}$, the string y, and the proof π_{SCG} using the exact same procedure as described in $\mathsf{Hyb}_3^{(0)}$ and $\mathsf{Hyb}_3^{(1)}$. Moreover, in $\mathsf{Hyb}_3^{(0)}$ and $\mathsf{Hyb}_3^{(1)}$ (as well as in the reduction), we have

$$z \oplus y = z \oplus (\mathsf{crs} \oplus z) = \mathsf{crs}.$$

It remains to consider the distribution of π_{NIZK}:

- Suppose $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w_0)$. This coincides with the distribution in $\mathsf{Hyb}_3^{(0)}$ so algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_3^{(0)}(\mathcal{A}) = 1]$.
- Suppose $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, x, w_1)$. This coincides with the distribution in $\mathsf{Hyb}_3^{(1)}$ so algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_3^{(1)}(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ distinguishes with the same advantage ε.

The case of unbounded $\mathcal{A}$. The above argument extends in a straightforward manner to the case when our adversary $\mathcal{A}$ is unbounded. The only differences are that we now allow the online algorithms to be inefficient and the advice string length to be unbounded. No other part of the proof relies on computational restrictions, and therefore the remainder of the argument proceeds unchanged. □

Proof of Theorem 3.9. Combining Claims 3.10 to 3.12, we conclude via a hybrid argument that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| \leq \mathsf{negl}(\lambda).$$

The claim follows. □

Remark 3.13 (On the Need for Non-Uniform Hardness)**.** Both the proofs of Theorem 3.4 and Theorem 3.9 rely on non-uniform hardness. We briefly describe how we could alternatively avoid non-uniformity by relying on sub-exponential hardness.

- **Soundness.** In our soundness analysis (Theorem 3.4), we rely on non-uniformity so the reduction algorithm can decide whether the underlying adversary outputs a true or a false statement. Alternatively, if the underlying NIZK scheme was sound against adversaries running in *sub-exponential time*, then we can complexity leverage and scale up the security parameter (as a function of the statement size) so that the reduction algorithm can decide the language itself. Effectively, the reduction algorithm can simply compute the advice string itself, and so there is no need to rely on the non-uniform advice bit for indicating whether the statement is true or not.
- **Witness indistinguishability.** In our witness indistinguishability analysis (Theorem 3.9), we run the (inefficient) Preprocess algorithm in the offline phase and provide the trapdoor to the online algorithm as the non-uniform advice in Claim 3.12. This means our analysis assumes the underlying NIZK provides witness-indistinguishability against non-uniform adversaries.

  First, we note that if the NIZK satisfies *statistical* witness indistinguishability, then it is of course also secure against non-uniform adversaries.

  Alternatively, if the NIZK only provides computational witness indistinguishability *and* if the Preprocess algorithm can be implemented in time $2^{\mathsf{poly}(\lambda)}$ which is independent of the output length ℓ_{out}, then we can again rely on complexity leveraging to avoid the need for non-uniform hardness. Specifically, if the underlying NIZK provided witness indistinguishability against adversaries running in sub-exponential time, then we can again complexity leverage and scale up the security parameter for the NIZK so that the reduction algorithm (in the proof of Claim 3.12) can simply run the Preprocess algorithm itself in the online phase. With this modification, the online algorithm no longer needs the non-uniform advice. Our sometimes-constricting generator based on the DDH assumption (Section 5 and Corollary 5.21) has the property where the running time of Preprocess is a function of the security parameter λ (and *not* the output length ℓ_{out}).

  For this latter implication, it is crucial that the running time of Preprocess be independent of ℓ_{out}.[4] This is because scaling up the security parameter for the NIZK could lead to a NIZK with a longer CRS, which in turn translates to a larger value of ℓ_{out} in Construction 3.2. Now, if the running time of Preprocess depends on ℓ_{out}, this means the NIZK would now need to be secure against an adversary whose running time is large enough to run Preprocess. This leads to a circular dependency. In contrast, if the running time of Preprocess only depends on the security parameter, then we can simply fix a security parameter λ, and then choose the NIZK security parameter such that the NIZK remains secure against an adversary whose running time is large enough to run Preprocess.

3.1 Extending to Super-Polynomial Security

We briefly remark about how to extend Construction 3.2 to obtain a $(1, \varepsilon)$-secure ZAP for inverse-super-polynomial ε. As noted in Section 2, sub-exponential security corresponds to the case where $\varepsilon(\lambda) = 2^{-\lambda^c}$ for some constant $c > 0$ and quasi-polynomial security corresponds to the case where $\varepsilon(\lambda) = 2^{-\log^c \lambda}$ for some constant $c > 1$. To do so, we require $(1, \varepsilon)$-security for the mode indistinguishability, the μ-guessing, and the target randomness properties in Definition 3.1. Concretely, we would make the following changes:

[4] Technically, we do allow a polylogarithmic dependence, but since we only consider settings where $\ell_{\mathsf{out}} \leq 2^{\lambda}$, we can always absorb polylogarithmic dependencies into the security parameter λ.

- **Mode indistinguishability:** We now require $(1, \varepsilon)$-mode indistinguishability. Namely, for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

  in the mode indistinguishability game.

- **μ-Guessing security:** We now require $(1, \varepsilon)$-μ-guessing security. Namely, we modify Eq. (3.2) to require for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[E_{\mathsf{guessed}}] \geq \mu(\lambda) \cdot (\Pr[\neg E_{\perp}] - \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)) \,. \tag{3.4}$$

- **Target randomness:** We now require $(1, \varepsilon)$-target randomness. Namely, for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

  in the target randomness game.

We can update the proofs of Theorem 3.4 and Theorem 3.9 to obtain a $(1, \varepsilon)$-secure ZAP as follows:

- **Soundness:** To prove soundness, we use the same sequence of hybrid experiments from the proof of Theorem 3.4.
  - Under the strengthened notion of mode indistinguishability, Claim 3.5 now says that

$$|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| \leq \varepsilon(\lambda) \cdot \mathsf{negl}_1(\lambda).$$

  - Under the strengthened notion of μ-guessing security, Claim 3.6 now says that

$$\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] \geq \mu(\lambda) \cdot (\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \varepsilon(\lambda) \cdot \mathsf{negl}_2(\lambda)).$$

  - Claim 3.7 is unaffected so we still have $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.
  - Finally, if the underlying NIZK satisfies $(1, \mu(\lambda) \cdot \varepsilon(\lambda))$-non-adaptive soundness, then Claim 3.8 shows that

$$\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = \mu(\lambda) \cdot \varepsilon(\lambda) \cdot \mathsf{negl}_3(\lambda).$$

    Recall from Lemma 2.9 that if the underlying NIZK is sub-exponentially sound, then we can complexity leverage to obtain a NIZK that is $(1, 2^{-p(\lambda)})$-sound for any polynomial $p(\lambda)$.
  - By the same hybrid argument and calculation as in the proof of Theorem 3.4, we now conclude that there is a negligible function $\mathsf{negl}(\cdot)$ such that $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$, which means the ZAP satisfies $(1, \varepsilon)$-soundness.

- **Witness indistinguishability:** For witness indistinguishability, we use the same sequence of hybrid experiments from the proof of Theorem 3.9. Take a NIZK that satisfies $(1, \varepsilon(\lambda))$-non-adaptive witness indistinguishability against non-uniform adversaries.
  - Under the strengthened notion of target randomness, Claim 3.10 now says that

$$|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| \leq \varepsilon(\lambda) \cdot \mathsf{negl}_1(\lambda).$$

  - Claim 3.11 is unaffected so we still have

$$\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1].$$

  - Finally, if the underlying NIZK satisfies $(1, \varepsilon(\lambda))$-non-adaptive soundness, then Claim 3.12 shows that

$$|\Pr[\mathsf{Hyb}_3^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(1)}(\mathcal{A}) = 1]| = \varepsilon(\lambda) \cdot \mathsf{negl}_2(\lambda).$$

  - By the same hybrid argument and calculation as in the proof of Theorem 3.9, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$ and the ZAP satisfies $(1, \varepsilon)$-witness indistinguishability.

4 Branch-Constricting Generator

In this section, we define an intermediate primitive called a branch-constricting generator. We then show how to use a branch-constricting generator to construct a sometimes-constricting generator. The notion of a branch-constricting generator provides a more convenient intermediate abstraction when constructing it from algebraic assumptions. Then, in Section 5, we show how to construct a branch-constricting generator from the DDH assumption (over pairing-free groups) and in Section 6, we show how to construct one from the plain LWE assumption.

Definition 4.1 (Branch-Constricting Generator)**.** Let λ be a security parameter and ℓ_{out} be an output length. A public-coin branch-constricting generator with branch set $\mathfrak{B} = \mathfrak{B}(\lambda)$ and public parameter size $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}})$ is a tuple of efficient algorithms $\Pi_{\mathsf{BCG}} = (\mathsf{Setup}, \mathsf{Sample}, \mathsf{Verify}, \mathsf{SampleGuess})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*) \to (\mathsf{pp}, \mathsf{aux})$: On input the security parameter 1^λ, the output length ℓ_{out}, and a branch $\mathsf{br}^* \in \mathfrak{B}$, the setup algorithm outputs a set of public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and auxiliary information aux. Throughout, we will assume that the public parameters pp always implicitly include a description of 1^λ and $1^{\ell_{\mathsf{out}}}$.
- $\mathsf{Sample}(\mathsf{pp}, \mathsf{br}) \to (y, \pi)$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and a branch $\mathsf{br} \in \mathfrak{B}$, the evaluation algorithm outputs a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ and a proof π.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}, y, \pi) \to b$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, a branch $\mathsf{br} \in \mathfrak{B}$, a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$. This algorithm is deterministic.
- $\mathsf{SampleGuess}(\mathsf{aux}) \to t^*$: On input the auxiliary information aux, the sample-guess algorithm outputs a string $t^* \in \{0,1\}^{\ell_{\mathsf{out}}}$.

We require Π_{BCG} to satisfy the following properties:

- **Completeness:** For all $\lambda, \ell_{\mathsf{out}} \in \mathbb{N}$, all branches $\mathsf{br} \in \mathfrak{B}$, all $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, all (y, π) in the support of $\mathsf{Sample}(\mathsf{pp}, \mathsf{br})$, it holds that $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}, y, \pi) = 1$.
- **Mode indistinguishability:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the mode-indistinguishability game as follows:
  - On input the security parameter 1^λ, the adversary $\mathcal{A}$ sends the output length $1^{\ell_{\mathsf{out}}}$ and a branch $\mathsf{br}^* \in \mathfrak{B}$ to the challenger.
  - If $b = 0$, the challenger samples $\mathsf{pp} \xleftarrow{\mathrm{R}} \{0,1\}^{\ell_{\mathsf{pp}}}$, where $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}})$. If $b = 1$, the challenger samples $(\mathsf{pp}, \mathsf{aux}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$. The challenger gives pp to $\mathcal{A}$.
  - Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.

  We say Π_{BCG} satisfies ε-mode-indistinguishability if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$

  $$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

  in the mode-indistinguishability game.
- **Most-branches target randomness.** There exists a pair of algorithms $(\mathsf{Preprocess}, \mathsf{Target})$ with the following syntax:
  - $\mathsf{Preprocess}(\mathsf{pp}, \mathsf{br}) \to \mathsf{td}$: On input the public parameters (which include an implicit description of 1^λ and $1^{\ell_{\mathsf{out}}}$) $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, and a branch $\mathsf{br} \in \mathfrak{B}$, the preprocess algorithm outputs a trapdoor td.
  - $\mathsf{Target}(\mathsf{td}, y) \to \pi$: On input a trapdoor td and a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$, the target algorithm outputs a proof π.

  We require that $(\mathsf{Preprocess}, \mathsf{Target})$ satisfy the following two properties:

– **Efficiency:** There exists a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $\mathsf{pp} \in \{0,1\}^*$ and all $\mathsf{br} \in \mathfrak{B}$, and all td in the support of $\mathsf{Preprocess}(\mathsf{pp}, \mathsf{br})$, we require that $|\mathsf{td}| \leq \mathsf{poly}(\lambda + \ell_{\mathsf{out}} + |\mathsf{pp}| + |\mathsf{br}|)$. In addition, we require that Target be efficiently-computable. Note that we do *not* require that Preprocess be efficiently-computable.

– **Most-branches target randomness:** For a security parameter λ, an output length ℓ_{out}, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the most-branches target randomness game as follows:

  * On input the security parameter 1^λ and the output length $1^{\ell_{\mathsf{out}}}$, the adversary $\mathcal{A}$ sends pp to the challenger.
  * The challenger computes $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}})$ and checks that $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$. Otherwise, it halts with output 0.
    · If $b = 0$, the challenger samples $\mathsf{br} \xleftarrow{R} \mathfrak{B}$ and computes $(y, \pi) \leftarrow \mathsf{Sample}(\mathsf{pp}, \mathsf{br})$.
    · If $b = 1$, the challenger samples $\mathsf{br} \xleftarrow{R} \mathfrak{B}$ and $y \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$. Next, it computes the trapdoor $\mathsf{td} \leftarrow \mathsf{Preprocess}(\mathsf{pp}, \mathsf{br})$ and $\pi \leftarrow \mathsf{Target}(\mathsf{td}, y)$.

    The challenger sends (br, y, π) to the adversary.
  * Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.

  We say that Π_{BCG} satisfies target randomness if for all (possibly unbounded) adversaries $\mathcal{A}$ and all polynomials $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$

  in the most-branches target randomness game.

- **Sampling on constricted branch:** There exists a polynomial $p = p(\lambda)$ such that for all $\lambda \in \mathbb{N}$, all $\ell_{\mathsf{out}} \in \mathbb{N}$, all $\mathsf{br}^* \in \mathfrak{B}$, all $(\mathsf{pp}, \mathsf{aux})$ in the support of $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$, all strings $y \in \{0,1\}^{\ell_{\mathsf{out}}}$, and all proofs $\pi \in \{0,1\}^*$ where $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}^*, y, \pi) = 1$, it holds that

$$\Pr[y = t^* : t^* \leftarrow \mathsf{SampleGuess}(\mathsf{aux})] \geq 2^{-p(\lambda)}. \tag{4.1}$$

Constructing a sometimes-constricting generator. We will now show how to construct a sometimes-constricting generator using a branch-constricting generator.

Construction 4.2 (Sometimes-Constricting Generator from Branch-Constricting Generator)**.** Let λ be a security parameter and ℓ_{out} be an output length. Let $\Pi_{\mathsf{BCG}} = (\mathsf{BCG.Setup}, \mathsf{BCG.Sample}, \mathsf{BCG.Verify}, \mathsf{BCG.SampleGuess})$ be a branch-constricting generator with branch set $\mathfrak{B} = \mathfrak{B}(\lambda)$, branch element size $\ell_{\mathsf{br}} = \ell_{\mathsf{br}}(\lambda, \ell_{\mathsf{out}})$ and public parameter size $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}})$. We construct a sometimes-constricting generator $\Pi_{\mathsf{SCG}} = (\mathsf{Setup}, \mathsf{Sample}, \mathsf{Verify})$ with public parameter size ℓ_{pp} as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}})$: On input the security parameter 1^λ and the output length $1^{\ell_{\mathsf{out}}}$, the setup algorithm samples $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$ and runs $(\mathsf{pp}, \mathsf{aux}) \leftarrow \mathsf{BCG.Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$. Next it computes $t^* \leftarrow \mathsf{BCG.SampleGuess}(\mathsf{aux})$ and outputs pp and the guess t^*.
- $\mathsf{Sample}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, sample $\mathsf{br} \xleftarrow{R} \mathfrak{B}$ and compute $(y, \pi_{\mathsf{BCG}}) \leftarrow \mathsf{BCG.Sample}(\mathsf{pp}, \mathsf{br})$. Output y and $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$.
- $\mathsf{Verify}(\mathsf{pp}, y, \pi)$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$, and a proof $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$, the verification algorithm outputs $\mathsf{BCG.Verify}(\mathsf{pp}, \mathsf{br}, y, \pi_{\mathsf{BCG}})$.

Theorem 4.3 (Completeness)**.** *If Π_{BCG} is complete, then Construction 4.2 is complete.*

Proof. Take any $\lambda, \ell_{\mathsf{out}} \in \mathbb{N}$, any $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, and any (y, π) in the support of $\mathsf{Sample}(\mathsf{pp})$. By construction, we can write $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$ and moreover, (y, π_{BCG}) is in the support of $\mathsf{BCG.Sample}(\mathsf{pp}, \mathsf{br})$. By completeness of Π_{BCG}, this means $\mathsf{BCG.Verify}(\mathsf{pp}, \mathsf{br}, y, \pi_{\mathsf{BCG}})$ outputs 1. This means $\mathsf{Verify}(\mathsf{pp}, y, \pi) = 1$, as required. □

Theorem 4.4 (Mode Indistinguishability)**.** *Suppose* Π_{BCG} *satisfies* ε*-mode indistinguishability for some* $\varepsilon(\lambda) \leq 1$*. Then Construction 4.2 satisfies mode indistinguishability.*

Proof. Take any efficient adversary $\mathcal{A}$ that breaks mode indistinguishability of the sometimes-constricting generator with advantage $\delta = \delta(\lambda)$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks mode indistinguishability of Π_{BCG}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $1^{\ell_{\mathsf{out}}} \leftarrow \mathcal{A}(1^\lambda)$.
2. Algorithm $\mathcal{B}$ samples $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$ and gives $(1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$ to the challenger. The challenger replies with pp.
3. Algorithm $\mathcal{B}$ gives pp to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs.

If $\mathcal{A}$ is efficient, then $\mathcal{B}$ is efficient by construction. We now consider the distribution of pp.

- Suppose the challenger samples $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$. This corresponds to the distribution in an execution of the mode indistinguishability experiment where $b = 0$.
- Suppose the challenger samples $(\mathsf{pp}, \mathsf{aux}) \leftarrow \mathsf{BCG.Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$. Since algorithm $\mathcal{B}$ samples $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$, the distribution of pp is precisely that of $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}})$, which coincides with the distribution in an execution of the mode indistinguishability experiment where $b = 1$.

We conclude that $\mathcal{B}$ breaks mode indistinguishability of Π_{BCG} with the same advantage $\delta = \delta(\lambda)$. Since Π_{BCG} satisfies ε-mode indistinguishability and $\varepsilon(\lambda) \leq 1$, we conclude that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\delta(\lambda) \leq \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda) \leq \mathsf{negl}(\lambda). \tag{4.2}$$

Thus, Construction 4.2 satisfies mode indistinguishability. □

Theorem 4.5 (μ-Guessing Security)**.** *Suppose* Π_{BCG} *satisfies the sampling on constricted branch property, and let* $p = p(\lambda)$ *be the associated polynomial from Eq. (4.1). Let* $\varepsilon(\lambda) \leq 1/|\mathfrak{B}(\lambda)|$*. If* Π_{BCG} *satisfies* ε*-mode indistinguishability, then Construction 4.2 satisfies* μ*-guessing security for* $\mu(\lambda) = 2^{-p(\lambda)} \cdot \varepsilon(\lambda)$*.*

Proof. Take any efficient adversary $\mathcal{A}$ for the μ-guessing security game. Then the μ-guessing security game for Construction 4.2 with security parameter λ and adversary $\mathcal{A}$ proceeds as follows:

1. On input 1^λ, the adversary $\mathcal{A}$ sends $1^{\ell_{\mathsf{out}}}$ to the challenger.
2. The challenger samples $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$, $(\mathsf{pp}, \mathsf{aux}) \leftarrow \mathsf{BCG.Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$, and $t^* \leftarrow \mathsf{BCG.SampleGuess}(\mathsf{aux})$. The challenger gives pp to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ either aborts with output $\perp$ or it outputs a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ and a proof π. The output of the experiment is $b = 1$ if $y = t^*$ and $\mathsf{Verify}(\mathsf{pp}, y, \pi) = 1$. In particular, the output is 1 if $y = t^*$ and $\mathsf{BCG.Verify}(\mathsf{pp}, \mathsf{br}, y, \pi_{\mathsf{BCG}}) = 1$ where $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$. Otherwise, the output is 0.

Let $\mathsf{Guess}(\mathcal{A})$ be the random variable denoting the output of the above experiment and $E_\perp$ be the event that adversary $\mathcal{A}$ aborts or that it outputs (y, π) where $\mathsf{Verify}(\mathsf{pp}, y, \pi) = 0$. Notably, if $\neg E_\perp$ occurs, then the adversary $\mathcal{A}$ must have output (y, π) where $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$ and $\mathsf{Verify}(\mathsf{pp}, y, \pi) = 1$, and correspondingly, $\mathsf{BCG.Verify}(\mathsf{pp}, \mathsf{br}, y, \pi_{\mathsf{BCG}}) = 1$. We need to show that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for $\lambda \in \mathbb{N}$,

$$\Pr[\mathsf{Guess}(\mathcal{A}) = 1] \geq \mu(\lambda) \cdot (\Pr[\neg E_\perp] - \mathsf{negl}(\lambda)).$$

By definition,

$$\begin{aligned} \Pr[\mathsf{Guess}(\mathcal{A}) = 1] &= \Pr[\neg E_\perp \wedge y = t^*] \\ &\geq \Pr[\neg E_\perp \wedge y = t^* \wedge \mathsf{br} = \mathsf{br}^*] \\ &= \Pr[y = t^* \mid (\mathsf{br} = \mathsf{br}^* \wedge \neg E_\perp)] \cdot \Pr[\mathsf{br} = \mathsf{br}^* \wedge \neg E_\perp]. \end{aligned} \tag{4.3}$$

Conditioned on $\neg E_\perp$, this means adversary $\mathcal{A}$ outputs (y, π) where $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$ and $\mathsf{BCG.Verify}(\mathsf{pp}, \mathsf{br}, y, \pi_{\mathsf{BCG}}) = 1$. Since Π_{BCG} satisfies sampling on constricted branch and $(\mathsf{pp}, \mathsf{aux})$ is in the support of $\mathsf{BCG.Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$, we have

$$\Pr[y = t^* \mid (\mathsf{br} = \mathsf{br}^* \wedge \neg E_\perp)] \geq 2^{-p(\lambda)}, \tag{4.4}$$

where the probability is taken over the choice of $t^* \leftarrow \mathsf{BCG.SampleGuess}(\mathsf{aux})$. To complete the proof, we now bound the probability of $\Pr[\mathsf{br} = \mathsf{br}^* \wedge \neg E_\perp]$.

Claim 4.6. *Let* $\mathsf{Match}(\mathcal{A})$ *be the indicator random variable for the event* $(\mathsf{br} = \mathsf{br}^* \wedge \neg E_\perp)$ *in the* μ*-guessing security experiment. If* Π_{BCG} *satisfies* ε*-mode indistinguishability, then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\Pr[\mathsf{Match}(\mathcal{A}) = 1] \geq \varepsilon(\lambda) \cdot (\Pr[\neg E_\perp] - \mathsf{negl}(\lambda)).$$

Proof. To prove the claim, we first define an intermediate hybrid Hyb where the challenger samples $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$ (independently of br^*):

1. On input 1^λ, the adversary $\mathcal{A}$ sends $1^{\ell_{\mathsf{out}}}$ to the challenger.
2. The challenger samples $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$ and $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$. The challenger gives pp to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ either aborts with output $\perp$ or it outputs a pair (y, π). The output of the experiment is 1 if $\mathsf{br} = \mathsf{br}^*$, $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$, and $\mathsf{BCG.Verify}(\mathsf{pp}, \mathsf{br}, y, \pi_{\mathsf{BCG}}) = 1$. Otherwise, the output of the experiment is 0.

Let $\mathsf{Match}'(\mathcal{A})$ be the random variable corresponding to the output of an execution of Hyb. Suppose

$$|\Pr[\mathsf{Match}(\mathcal{A}) = 1] - \Pr[\mathsf{Match}'(\mathcal{A}) = 1]| \geq \varepsilon(\lambda) \cdot \delta(\lambda)$$

for some non-negligible function $\delta = \delta(\lambda)$. We use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ that breaks ε-mode-indistinguishability of Π_{BCG}:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $1^{\ell_{\mathsf{out}}} \leftarrow \mathcal{A}(1^\lambda)$.
2. Algorithm $\mathcal{B}$ samples $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$ and gives $(1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$ to the challenger. The challenger replies with pp.
3. Algorithm $\mathcal{B}$ gives pp to $\mathcal{A}$. Algorithm $\mathcal{A}$ either outputs $\perp$ or it outputs a pair (y, π).
4. Output 1 if $\mathcal{A}$ outputs a pair (y, π) where $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$, $\mathsf{br} = \mathsf{br}^*$, and $\mathsf{BCG.Verify}(\mathsf{pp}, \mathsf{br}, y, \pi_{\mathsf{BCG}}) = 1$. Otherwise, output 0.

We consider the two possibilities depending on the distribution of pp:

- If the challenger samples $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$, then algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb. In this case, algorithm $\mathcal{B}$ outputs 1 if and only if $\mathsf{Match}'(\mathcal{A}) = 1$.
- If the challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$, then algorithm $\mathcal{B}$ perfectly simulates the public parameter distribution in the original μ-guessing security game. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Match}(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ has advantage

$$|\Pr[\mathsf{Match}(\mathcal{A}) = 1] - \Pr[\mathsf{Match}'(\mathcal{A}) = 1]| \geq \varepsilon(\lambda) \cdot \delta(\lambda),$$

for some non-negligible $\delta = \delta(\lambda)$. This contradicts the ε-mode-indistinguishability of Π_{BCG}. Thus, we conclude that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Match}(\mathcal{A}) = 1] - \Pr[\mathsf{Match}'(\mathcal{A}) = 1]| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda). \tag{4.5}$$

To complete the proof, we now bound the probability $\Pr[\mathsf{Match}'(\mathcal{A}) = 1]$. By construction, in Hyb, the distribution of pp is independent of br^*, and in particular, the challenger can defer the sampling of br^* until after the adversary $\mathcal{A}$ outputs (y, π). Since the challenger samples $\mathsf{br}^* \xleftarrow{R} \mathfrak{B}$ and independently of all other quantities, we have

$$\begin{aligned}\Pr[\mathsf{Match}'(\mathcal{A}) = 1] &= \Pr[\mathsf{br} = \mathsf{br}^* \wedge \neg E_\perp] \\ &= \Pr[\neg E_\perp] \cdot \Pr[\mathsf{br} = \mathsf{br}^* \mid \neg E_\perp] \\ &= \Pr[\neg E_\perp] \cdot \frac{1}{|\mathfrak{B}(\lambda)|} \\ &\geq \varepsilon(\lambda) \cdot \Pr[\neg E_\perp].\end{aligned} \tag{4.6}$$

Combined with Eq. (4.5), we conclude that

$$\begin{aligned}\Pr[\mathsf{Match}(\mathcal{A}) = 1] &\geq \Pr[\mathsf{Match}'(\mathcal{A}) = 1] - \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda) \\ &\geq \varepsilon(\lambda) \cdot (\Pr[\neg E_\perp] - \mathsf{negl}(\lambda)).\end{aligned}$$

The claim follows. □

Combining Eqs. (4.3) and (4.4) and Claim 4.6, we now have

$$\begin{aligned}\Pr[\mathsf{Guess}(\mathcal{A}) = 1] &= \Pr[y = t^* \mid (\neg E_\perp \wedge \mathsf{br} = \mathsf{br}^*)] \cdot \Pr[\mathsf{br} = \mathsf{br}^* \wedge \neg E_\perp] \\ &= 2^{-p(\lambda)} \cdot \Pr[\mathsf{Match}(\mathcal{A}) = 1] \\ &\geq 2^{-p(\lambda)} \cdot \varepsilon(\lambda) \cdot (\Pr[\neg E_\perp] - \mathsf{negl}(\lambda)).\end{aligned}$$

Thus, Construction 4.2 satisfies μ-guessing security where $\mu(\lambda) = 2^{-p(\lambda)} \cdot \varepsilon(\lambda)$. □

Theorem 4.7 (Target Randomness)**.** *Suppose Π_{BCG} satisfies most-branches target randomness. Then for all polynomials $\ell_{\mathsf{br}} = \ell_{\mathsf{br}}(\lambda)$, Construction 4.2 satisfies target randomness.*

Proof. Let $(\mathsf{BCG.Preprocess}, \mathsf{BCG.Target})$ be the targeting algorithms associated with the most-branches target randomness property of Π_{BCG}. We define the corresponding targeting algorithms for Construction 4.2 as follows:

- $\mathsf{Preprocess}(\mathsf{pp})$: On input the public parameter pp (which includes an implicit description of 1^λ and $1^{\ell_{\mathsf{out}}}$), sample a branch $\mathsf{br} \xleftarrow{R} \mathfrak{B}$ and a trapdoor $\mathsf{td}_{\mathsf{BCG}} \leftarrow \mathsf{BCG.Preprocess}(\mathsf{pp}, \mathsf{br})$. Output $\mathsf{td} = (\mathsf{br}, \mathsf{td}_{\mathsf{BCG}})$.
- $\mathsf{Target}(\mathsf{td}, y)$: On input $\mathsf{td} = (\mathsf{br}, \mathsf{td}_{\mathsf{BCG}})$, compute $\pi_{\mathsf{BCG}} \leftarrow \mathsf{BCG.Target}(\mathsf{td}_{\mathsf{BCG}}, y)$ and output $\pi_{\mathsf{SCG}} = (\mathsf{br}, \pi_{\mathsf{BCG}})$.

We now show that $(\mathsf{Preprocess}, \mathsf{Target})$ satisfy the required properties.

Efficiency. The size of the trapdoor td output by $\mathsf{Preprocess}(\mathsf{pp})$ is $\mathsf{poly}(\lambda + \ell_{\mathsf{out}} + |\mathsf{pp}| + |\mathsf{br}|)$. This is a polynomial in λ, ℓ_{out}, and $|\mathsf{pp}|$ since $|\mathsf{br}| = \ell_{\mathsf{br}}$ is polynomially-bounded. Moreover, since BCG.Target is efficiently-computable, the same holds for Target.

Most-branches target randomness. Suppose there exists a polynomial $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$ and a (possibly-unbounded) adversary $\mathcal{A}$ that can break target randomness of Construction 4.2 with non-negligible probability. We use $\mathcal{A}$ to construct a (possibly-unbounded) adversary $\mathcal{B}$ that breaks most-branches target randomness of Π_{BCG} (for the same output length $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$):

1. On input the security parameter 1^λ and the output length $1^{\ell_{\mathsf{out}}}$, invoke $\mathcal{A}$ on the same inputs to obtain a set of public parameters pp. Algorithm $\mathcal{B}$ forwards pp to the challenger.
2. The challenger replies with a triple $(\mathsf{br}, y, \pi_{\mathsf{BCG}})$. Algorithm $\mathcal{B}$ sets $\pi = (\mathsf{br}, \pi_{\mathsf{BCG}})$ and gives (y, π) to $\mathcal{A}$.
3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which algorithm $\mathcal{B}$ also outputs.

We now compute the advantage of $\mathcal{B}$:

- Suppose the challenger samples $\mathsf{br} \xleftarrow{R} \mathfrak{B}$ and $(y, \pi_{\mathsf{BCG}}) = \mathsf{BCG.Sample}(\mathsf{pp}, \mathsf{br})$. In this case, the pair (y, π) is sampled exactly as in $\mathsf{Sample}(\mathsf{pp})$. This is the target randomness experiment for Construction 4.2 with $b = 0$.
- Suppose the challenger samples $\mathsf{br} \xleftarrow{R} \mathfrak{B}$, $y \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$, $\mathsf{td}_{\mathsf{BCG}} \leftarrow \mathsf{BCG.Preprocess}(\mathsf{pp}, \mathsf{br})$, and $\pi_{\mathsf{BCG}} \leftarrow \mathsf{BCG.Target}(\mathsf{td}_{\mathsf{BCG}}, y)$. Then, the pair $\mathsf{td} = (\mathsf{br}, \mathsf{td}_{\mathsf{BCG}})$ is distributed exactly as $\mathsf{Preprocess}(\mathsf{pp})$ and π is distributed exactly according to $\mathsf{Target}(\mathsf{td}, y)$. Since y is uniform over $\{0,1\}^{\ell_{\mathsf{out}}}$, the pair (y, π) is distributed exactly according to the specification of the target randomness experiment for Construction 4.2 with $b = 1$.

We conclude that algorithm $\mathcal{B}$ breaks most-branches target randomness of Π_{BCG} with the same advantage as $\mathcal{A}$. □

Corollary 4.8 (Sometimes-Constricting Generator)**.** *Let λ be a security parameter. Suppose there exists a branch-constricting generator with a branch set of size $N = N(\lambda)$. Suppose further that Π_{BCG} satisfies completeness, ε-mode indistinguishability for $\varepsilon(\lambda) \leq 1/N(\lambda)$, sampling on constricted branch and target randomness. Then there exists a polynomial $p = p(\lambda)$ such that the resulting sometimes-constricting generator satisfies completeness, mode-indistinguishability, μ-guessing security, and target randomness, where $\mu(\lambda) = 2^{-p(\lambda)} \cdot \varepsilon(\lambda)$.*

4.1 Extending to Super-Polynomial Security

Similar to Section 3.1, it is straightforward to extend Construction 4.2 to obtain a $(1, \varepsilon)$-secure sometimes-constricting generator for inverse-super-polynomial ε. As discussed in Section 3.1, this is relevant for getting a ZAP with quasi-polynomial or sub-exponential security. To extend Construction 4.2 to obtain a $(1, \varepsilon)$-secure sometimes-constricting generator, we would first require that the branch-constricting generator satisfy the following stronger properties:

- **Mode indistinguishability:** First, we require $(1, \varepsilon^2)$-mode indistinguishability. Namely, for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \varepsilon^2(\lambda) \cdot \mathsf{negl}(\lambda)$$

  in the mode indistinguishability game.

- **Most-branches target randomness:** Second, we require $(1, \varepsilon)$-most-branches target randomness. Specifically, for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

  in the most-branches target randomness game.

Using these properties, we can update the proofs of Theorems 4.4, 4.5 and 4.7 to satisfy the strengthened notions from Section 3.1. The main difference is we now require that Π_{BCG} satisfy $(1, \varepsilon^2)$-mode indistinguishability (instead of $(1, \varepsilon)$-mode indistinguishability).

- **Mode indistinguishability:** The proof of Theorem 4.4 directly applies. Specifically, if Π_{BCG} satisfies $(1, \varepsilon^2)$-mode indistinguishability, then the proof in Theorem 4.4 shows that Construction 4.2 also satisfies $(1, \varepsilon^2)$-mode indistinguishability (which implies $(1, \varepsilon)$-mode indistinguishability since $\varepsilon < 1$).
- **μ-Guessing advantage:** We follow the same template as the proof of Theorem 4.5.
  - We modify Claim 4.6 to argue that

$$\Pr[\mathsf{Match}(\mathcal{A}) = 1] \geq \varepsilon(\lambda) \cdot (\Pr[\neg E_\perp] - \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)).$$

  - If Π_{BCG} satisfies $(1, \varepsilon^2)$-mode indistinguishability, Eq. (4.5) now becomes

$$|\Pr[\mathsf{Match}(\mathcal{A}) = 1] - \Pr[\mathsf{Match}'(\mathcal{A}) = 1]| = \varepsilon^2(\lambda) \cdot \mathsf{negl}(\lambda).$$

    Eq. (4.6) is unchanged and still says that

$$\Pr[\mathsf{Match}'(\mathcal{A}) = 1] \geq \varepsilon(\lambda) \cdot \Pr[\neg E_\perp].$$

– By the same calculation as in the proof of Theorem 4.5,

$$\Pr[\mathsf{Guess}(\mathcal{A}) = 1] \geq 2^{-p(\lambda)} \cdot \Pr[\mathsf{Match}(\mathcal{A}) = 1] \geq \mu(\lambda) \cdot (\Pr[\neg E_\perp] - \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda))$$

where $\mu(\lambda) = 2^{-p(\lambda)} \cdot \varepsilon(\lambda)$. Recall that $p(\lambda)$ is still the same associated polynomial from Eq. (4.1).

- **Target randomness:** Similar to the case with mode indistinguishability, the proof of Theorem 4.7 directly extends.

5 Branch-Constricting Generator from DDH

In this section, we show how to construct a branch-constricting generator from the decisional Diffie-Hellman (DDH) assumption (over a standard pairing-free group). We begin by formally defining the DDH assumption.

Definition 5.1 (Prime-Order Group Generator)**.** A prime-order group generator GroupGen is an efficient algorithm that takes as input the security parameter 1^λ and outputs a description $\mathcal{G} = (\mathbb{G}, p, g)$ of a group $\mathbb{G}$ with prime order $p > 2^\lambda$ and generator g. We require that $p = 2^{\Theta(\lambda)}$ and that the group operation in $\mathbb{G}$ be efficiently computable, and that each element of $\mathbb{G}$ can be represented by a bit-string of length at most $O(\lambda)$.

Definition 5.2 (Public-Coin Prime-Order Group Generator)**.** We say GroupGen is a public-coin prime-order group generator if the algorithm GroupGen is public-coin.

Definition 5.3 (Obliviously-Sampleable Group)**.** Let GroupGen be a prime-order group generator. We say that GroupGen is obliviously sampleable with randomness complexity $\rho = \rho(\cdot)$ if $\mathsf{GroupGen}(1^\lambda)$ outputs the description of a group $(\mathcal{G}, p, g)$ in addition to two efficient algorithms $(\mathsf{Samp}, \mathsf{Samp}^{-1})$ with the following properties:

- Algorithm Samp is deterministic and Samp^{-1} may be randomized.
- For all $\lambda \in \mathbb{N}$ and all $h \in \mathbb{G}$,

$$\Pr[\mathsf{Samp}(1^\lambda, r) = h : r \leftarrow \mathsf{Samp}^{-1}(1^\lambda, h)] = 1.$$

- There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the statistical distance between the following distributions is at most $2^{-\lambda} \cdot \mathsf{negl}(\lambda)$:

$$\left\{(r, \mathsf{Samp}(1^\lambda, r)) : r \xleftarrow{R} \{0,1\}^{\rho(\lambda)}\right\} \quad \text{and} \quad \left\{(\mathsf{Samp}^{-1}(1^\lambda, h), h) : h \xleftarrow{R} \mathbb{G}\right\}.$$

  Note that this can be extended to polynomially many samples via a standard hybrid argument.

Notation. We will use implicit notation to represent group elements [EHK+13]. Specifically, let $\mathcal{G} = (\mathbb{G}, p, g)$ be a prime-order group. For a matrix $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$, we write $[\![\mathbf{A}]\!]$ to denote the matrix of group elements $g^{\mathbf{A}}$ (where exponentiation is defined component-wise). For matrices $\mathbf{A}, \mathbf{B}$ of identical dimension and a scalar c, we write $c \cdot [\![\mathbf{A}]\!] \coloneqq [\![c\mathbf{A}]\!]$ and $[\![\mathbf{A}]\!] + [\![\mathbf{B}]\!] \coloneqq [\![\mathbf{A} + \mathbf{B}]\!]$. We now define the DDH problem.

Assumption 5.4 (Decisional Diffie-Hellman)**.** Let GroupGen be a prime-order group generator. The ε-decisional Diffie-Hellman (ε-DDH) assumption holds with respect to GroupGen if for all efficient (and possibly non-uniform) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathcal{G}, [\![u]\!], [\![v]\!], [\![uv]\!]) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathcal{G}, [\![u]\!], [\![v]\!], [\![w]\!]) = 1\right]\right| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

where the probability is over $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and $u, v, w \xleftarrow{R} \mathbb{Z}_p$. When GroupGen is a public-coin group, the group description $\mathcal{G}$ is replaced by the random coins used to sample $\mathcal{G}$.

Matrix DDH. In our construction, it will be more convenient to use the following matrix version of the DDH assumption, which follows from the plain DDH assumption by a standard hybrid argument (cf. [EHK+13]). We state the version we use here.

Assumption 5.5 (Matrix Decisional Diffie-Hellman)**.** Let GroupGen be a prime-order group generator and $n = n(\lambda)$ be an input dimension. We say that the ε-matrix decisional Diffie-Hellman (ε-MDDH) assumption holds with respect to GroupGen if for all polynomials $n = n(\lambda)$ and all efficient (and possibly non-uniform) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathcal{G}, [\![\mathbf{u}]\!], [\![\mathbf{v}]\!], [\![\mathbf{u}\mathbf{v}^\intercal]\!]) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathcal{G}, [\![\mathbf{u}]\!], [\![\mathbf{v}]\!], [\![\mathbf{M}]\!]) = 1\right]\right| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda),$$

where $\mathbf{u}, \mathbf{v} \xleftarrow{R} \mathbb{Z}_p^n$ and $\mathbf{M} \xleftarrow{R} \mathbb{Z}_p^{n \times n}$. When GroupGen is a public-coin group, the group description $\mathcal{G}$ is replaced by the random coins used to sample $\mathcal{G}$.

Construction 5.6 (Branch-Constricting Generator from DDH)**.** Let λ be a security parameter. Our construction relies on the following ingredients:

- Let the branch set size $N = N(\lambda)$ where $N \leq 2^\lambda$.
- Let GroupGen be an obliviously-sampleable public-coin prime-order group generator. Let $\ell_{\mathcal{G}} = \ell_{\mathcal{G}}(\lambda)$ be the length of the random coins used by GroupGen. Let $\rho = \rho(\lambda)$ be the randomness complexity for the oblivious sampler associated with GroupGen and $\ell_{\mathbb{G}} = \ell_{\mathbb{G}}(\lambda)$ be a bound on the representation size of a single group element.
- Let $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$ where each $\mathcal{H}_\lambda$ is a family of universal hash functions with domain $\{0,1\}^{\ell_{\mathbb{G}}(\lambda)}$, seed space $\{0,1\}^{\ell_s(\lambda)}$ and range $\{0,1\}$.

We construct a branch-constricting generator $\Pi_{\mathsf{BCG}} = (\mathsf{Setup}, \mathsf{Sample}, \mathsf{Verify}, \mathsf{SampleGuess})$ with branch set $\mathfrak{B} = \mathfrak{B}(\lambda) = [N(\lambda)]$ and public parameter size $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}}) = \ell_{\mathcal{G}}(\lambda) + \rho(\lambda) \cdot {\ell_{\mathsf{out}}}^2$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$: On input the security parameter λ, the output length ℓ_{out}, and a branch $\mathsf{br}^* \in \mathfrak{B}$, the setup algorithm proceeds as follows:
  - Sample $\mathsf{pp}_{\mathcal{G}} \xleftarrow{R} \{0,1\}^{\ell_{\mathcal{G}}(\lambda)}$ and let $\mathcal{G} = (\mathbb{G}, p, g) = \mathsf{GroupGen}(1^\lambda; \mathsf{pp}_{\mathcal{G}})$. Let $(\mathsf{Samp}, \mathsf{Samp}^{-1})$ be the oblivious sampling algorithms associated with the group $\mathcal{G}$.
  - Sample $\mathbf{u}, \mathbf{v} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$ and let $\mathbf{M} = \mathbf{u}\mathbf{v}^\intercal - \mathsf{br}^* \cdot \mathbf{I} \in \mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$ where $\mathbf{I} \in \{0,1\}^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$ is the identity matrix.

  Let $\mathsf{pp}_{\mathbf{M}} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, [\![\mathbf{M}]\!])$, where we apply Samp^{-1} on each group element in the input. Output $\mathsf{pp} = \mathsf{pp}_{\mathcal{G}} \| \mathsf{pp}_{\mathbf{M}} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and the auxiliary information $\mathsf{aux} = (\mathcal{G}, [\![\mathbf{u}]\!])$. As usual, we assume the public parameters include an implicit description of the security parameter 1^λ and the output length $1^{\ell_{\mathsf{out}}}$.

  Parsing pp**:** In the following, we will interpret a bit string $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ as encoding the group description $\mathcal{G}$ together with $[\![\mathbf{M}]\!]$. Specifically, we first parse pp as $\mathsf{pp} = \mathsf{pp}_{\mathcal{G}} \| \mathsf{pp}_{\mathbf{M}}$ where $\mathsf{pp}_{\mathcal{G}} \in \{0,1\}^{\ell_{\mathcal{G}}}$ and $\mathsf{pp}_{\mathbf{M}} \in \{0,1\}^{\rho \cdot \ell_{\mathsf{out}}^2}$. Then, we compute $\mathcal{G} = (\mathbb{G}, p, g) = \mathsf{GroupGen}(1^\lambda; \mathsf{pp}_{\mathcal{G}})$ and $[\![\mathbf{M}]\!] = \mathsf{Samp}(1^\lambda, \mathsf{pp}_{\mathbf{M}})$, where we apply Samp block-by-block.
- $\mathsf{Sample}(\mathsf{pp}, \mathsf{br})$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and a branch $\mathsf{br} \in \mathfrak{B}$, the sampling algorithm proceeds as follows:
  - Parse pp as $(\mathcal{G}, [\![\mathbf{M}]\!])$ in the manner described in Setup.
  - Sample $h \xleftarrow{R} \mathcal{H}_\lambda$ and $\mathbf{r} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$.
  - Compute $y = h([\![(\mathbf{M} + \mathsf{br} \cdot \mathbf{I}) \cdot \mathbf{r}]\!])$, where the hash function h is applied component-wise to the binary representation of each group element.

  Output y and $\pi = (\mathbf{r}, h)$.

- $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}, y, \pi)$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, a branch $\mathsf{br} \in \mathfrak{B}$, a string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$, and a proof $\pi = (\mathbf{r}, h)$, the verification algorithm first parses pp as $(\mathcal{G}, [\![\mathbf{M}]\!])$ in the manner described in Setup. Then, it outputs 1 if $h \in \mathcal{H}_\lambda$, $\mathbf{r} \in \mathbb{Z}_p^{\ell_{\mathsf{out}}}$, and $y = h([\![(\mathbf{M} + \mathsf{br} \cdot \mathbf{I}) \cdot \mathbf{r}]\!])$, where the hash function h is applied component-wise to the binary representation of each group element.
- $\mathsf{SampleGuess}(\mathsf{aux})$: On input the auxiliary information $\mathsf{aux} = (\mathcal{G}, [\![\mathbf{u}]\!])$ sample $\alpha \xleftarrow{R} \mathbb{Z}_p$, $h \xleftarrow{R} \mathcal{H}_\lambda$, and output $t = h(\alpha \cdot [\![\mathbf{u}]\!])$.

Theorem 5.7 (Completeness)**.** *Suppose* GroupGen *is obliviously sampleable (Definition 5.3). Then Construction 5.6 is complete.*

Proof. Take any $\lambda, \ell_{\mathsf{out}} \in \mathbb{N}$ and any branches $\mathsf{br} \in \mathfrak{B}$. Take $\mathsf{pp} = \mathsf{pp}_{\mathcal{G}} \| \mathsf{pp}_{\mathbf{M}} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and (y, π) in the support of $\mathsf{Sample}(\mathsf{pp}, \mathsf{br})$. From Definition 5.3, this means $\mathsf{Samp}(1^\lambda, \mathsf{pp}_{\mathbf{M}}) = [\![\mathbf{M}]\!]$ for $[\![\mathbf{M}]\!] \in \mathbb{G}^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$. By construction of Sample, we have that $y = h([\![(\mathbf{M} + \mathsf{br} \cdot \mathbf{I}) \cdot \mathbf{r}]\!])$ and $\pi = (\mathbf{r}, h)$ for some $\mathbf{r} \in \mathbb{Z}_p^{\ell_{\mathsf{out}}}$, $h \in \mathcal{H}_\lambda$. These properties precisely coincide with the check that $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}, y, \pi)$ performs, so completeness holds. □

Theorem 5.8 (Mode Indistinguishability)**.** *Suppose* GroupGen *is obliviously sampleable and that the ε-MDDH assumption holds with respect to* GroupGen *for some $\varepsilon(\lambda)$. Then Construction 5.6 satisfies ε'-mode indistinguishability where $\varepsilon'(\lambda) = \max(\varepsilon(\lambda), 2^{-\lambda})$.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the mode indistinguishability game. We define the following sequence of hybrid experiments:

- Hyb_1: This is the mode indistinguishability game with bit $b = 0$. Specifically, the game proceeds as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the length $1^{\ell_{\mathsf{out}}}$ and a branch $\mathsf{br}^* \in \mathfrak{B}$.
  2. The challenger samples $\mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}$ and gives pp to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- Hyb_2: Same as Hyb_1, except the challenger changes how it samples the public parameters pp:
  2. The challenger samples $\mathsf{pp}_{\mathcal{G}} \xleftarrow{R} \{0,1\}^{\ell_{\mathcal{G}}}$ and sets $\mathcal{G} = (\mathbb{G}, p, g) = \mathsf{GroupGen}(1^\lambda; \mathsf{pp}_{\mathcal{G}})$. Then, it samples $\mathbf{M} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$. Finally, it sets $\mathsf{pp}_{\mathbf{M}} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, [\![\mathbf{M}]\!])$ and gives $\mathsf{pp} = \mathsf{pp}_{\mathcal{G}} \| \mathsf{pp}_{\mathbf{M}}$ to $\mathcal{A}$.
- Hyb_3: Same as Hyb_2, except the challenger changes how it samples the public parameters pp:
  2. The challenger samples $\mathsf{pp}_{\mathcal{G}} \xleftarrow{R} \{0,1\}^{\ell_{\mathcal{G}}}$ and sets $\mathcal{G} = (\mathbb{G}, p, g) = \mathsf{GroupGen}(1^\lambda; \mathsf{pp}_{\mathcal{G}})$. Then, it samples $\mathbf{u} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$, $\mathbf{v} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$ and sets $\mathbf{M} = \mathbf{u}\mathbf{v}^\intercal - \mathsf{br}^* \cdot \mathbf{I}$. Finally, it sets $\mathsf{pp}_{\mathbf{M}} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, [\![\mathbf{M}]\!])$ and gives $\mathsf{pp} = \mathsf{pp}_{\mathcal{G}} \| \mathsf{pp}_{\mathbf{M}}$ to $\mathcal{A}$.

  This is the mode indistinguishability game with bit $b = 1$.

Let $\mathsf{Hyb}_i(\mathcal{A})$ be the random variable denoting the output of an execution of experiment Hyb_i with adversary $\mathcal{A}$. We now analyze each adjacent pair of hybrid experiments.

Claim 5.9. *Suppose* GroupGen *is obliviously-sampleable. Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = 2^{-\lambda} \cdot \mathsf{negl}(\lambda)$.*

Proof. The only difference between the hybrids is in the way the public parameters pp are sampled. In Hyb_1, the public parameters $\mathsf{pp}_{\mathbf{M}}$ are sampled uniformly at random from $\{0,1\}^{\ell_{\mathsf{pp}}}$, while in Hyb_2, they are sampled as $\mathsf{pp}_{\mathbf{M}} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, [\![\mathbf{M}]\!])$ for uniformly random $\mathbf{M} \in \mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$. Since GroupGen is obliviously-sampleable and ℓ_{out} is polynomial in λ (as $\mathcal{A}$ is efficient), there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the statistical distance between the following distributions[5] is at most $2^{-\lambda} \cdot \mathsf{negl}(\lambda)$:

$$\left\{r : r \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}\right\} \quad \text{and} \quad \left\{\mathsf{Samp}^{-1}(1^\lambda, [\![\mathbf{M}]\!]) : \mathbf{M} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}\right\}.$$

These distributions exactly correspond to the distribution of pp in Hyb_1 and Hyb_2 respectively, so the claim follows. □

[5] Strictly speaking, in Definition 5.3 the statistical distance is between joint distributions. However marginalizing out one variable cannot increase the statistical distance, so the property used here also holds.

Claim 5.10. *Suppose* GroupGen *is a public-coin group sampler and the ε-MDDH assumption holds with respect to* GroupGen. *Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| > \varepsilon(\lambda) \cdot \delta(\lambda)$ for some non-negligible δ. Without loss of generality, suppose that for each security parameter λ, algorithm $\mathcal{A}$ chooses a fixed value of $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$.[6] We now use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ that breaks the ε-MDDH assumption with dimension ℓ_{out} with the same advantage. Algorithm $\mathcal{B}$ works as follows:

1. On input the security parameter 1^λ and the challenge $(\mathsf{pp}_{\mathcal{G}}, [\![\mathbf{u}]\!], [\![\mathbf{v}]\!], [\![\mathbf{M}]\!])$, algorithm $\mathcal{B}$ invokes algorithm $\mathcal{A}$ on the same security parameter.
2. Algorithm $\mathcal{A}$ outputs the output length $1^{\ell_{\mathsf{out}}}$ and the branch $\mathsf{br}^* \in \mathfrak{B}$.
3. Algorithm $\mathcal{B}$ computes $\mathsf{pp}_{\mathbf{M}} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, [\![\mathbf{M} - \mathsf{br}^* \cdot \mathbf{I}]\!])$ and gives $\mathsf{pp} = \mathsf{pp}_{\mathcal{G}} \| \mathsf{pp}_{\mathbf{M}}$ to $\mathcal{A}$.
4. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which $\mathcal{B}$ also outputs.

We argue that depending on the distribution of the challenge, algorithm $\mathcal{B}$ either perfectly simulates an execution of Hyb_2 or of Hyb_3 for $\mathcal{A}$:

- Suppose the challenger sends $(\mathsf{pp}_{\mathcal{G}}, [\![\mathbf{u}]\!], [\![\mathbf{v}]\!], [\![\mathbf{M}]\!])$ for uniformly random $\mathbf{M} \in \mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$. Algorithm $\mathcal{B}$ computes $[\![\mathbf{M} - \mathsf{br}^* \cdot \mathbf{I}]\!]$ and sets $\mathsf{pp}_{\mathbf{M}} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, [\![\mathbf{M} - \mathsf{br}^* \cdot \mathbf{I}]\!])$. It then sends $\mathsf{pp} = \mathsf{pp}_{\mathcal{G}} \| \mathsf{pp}_{\mathbf{M}}$ to $\mathcal{A}$. Since $\mathbf{M}$ is uniformly random, the distribution of $\mathbf{M} - \mathsf{br}^* \cdot \mathbf{I}$ is uniform over $\mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$ and independent of br^*. This is consistent with Hyb_2, hence $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.
- Suppose the challenger sends $(\mathsf{pp}_{\mathcal{G}}, [\![\mathbf{u}]\!], [\![\mathbf{v}]\!], [\![\mathbf{M}]\!])$ where $[\![\mathbf{M}]\!] = [\![\mathbf{u}\mathbf{v}^\intercal]\!]$ to $\mathcal{B}$. Algorithm $\mathcal{B}$ computes $[\![\mathbf{M} - \mathsf{br}^* \cdot \mathbf{I}]\!] = [\![\mathbf{u}\mathbf{v}^\intercal - \mathsf{br}^* \cdot \mathbf{I}]\!]$ and sets $\mathsf{pp}_{\mathbf{M}} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, [\![\mathbf{u}\mathbf{v}^\intercal - \mathsf{br}^* \cdot \mathbf{I}]\!])$. It then sends $\mathsf{pp} = \mathsf{pp}_{\mathcal{G}} \| \mathsf{pp}_{\mathbf{M}}$ to $\mathcal{A}$. This is consistent with Hyb_3, hence $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ wins with the same advantage $\varepsilon(\lambda) \cdot \delta(\lambda)$, which breaks the ε-MDDH assumption. □

Proof of Theorem 5.8. From Claims 5.9 and 5.10, there exist negligible functions $\mathsf{negl}_1(\cdot)$, $\mathsf{negl}_2(\cdot)$ such that

$$|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| \leq 2^{-\lambda} \cdot \mathsf{negl}_1(\lambda)$$
$$|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq \varepsilon(\lambda) \cdot \mathsf{negl}_2(\lambda)$$

Taking $\varepsilon' = \max(\varepsilon, 2^{-\lambda})$, $\mathsf{negl}_3(\lambda) = \mathsf{negl}_1(\lambda) + \mathsf{negl}_2(\lambda)$, and applying the triangle inequality, we see that

$$|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq 2^{-\lambda} \cdot \mathsf{negl}_1(\lambda) + \varepsilon(\lambda) \cdot \mathsf{negl}_2(\lambda) \leq \varepsilon'(\lambda) \cdot \mathsf{negl}_3(\lambda).$$

Thus the claim follows. □

Theorem 5.11 (Most-Branches Target Randomness)**.** *Suppose $\mathcal{H}$ is a universal hash function family and* $1/N(\lambda) = \mathsf{negl}(\lambda)$. *For all polynomials $\ell_{out} = \ell_{out}(\lambda)$, Construction 5.6 satisfies most-branches target randomness.*

Proof. We begin by stating a useful lemma from [CKSU21].

Lemma 5.12 ([CKSU21, Lemma 33, adapted])**.** *Let p be a prime and take any matrix $\mathbf{M} \in \mathbb{Z}_p^{n \times n}$. Let $N < p$ be a positive integer. Sample $w \xleftarrow{R} [N]$ and let $\mathbf{A} = \mathbf{M} + w \cdot \mathbf{I}_n$, where $\mathbf{I}_n$ is the n-by-n identity matrix. Then, over the randomness of w, the matrix $\mathbf{A}$ is invertible with probability at least $1 - n/N$.*

[6] For instance, for each security parameter, we could provide the reduction algorithm the value of ℓ_{out} that contributes the most to algorithm $\mathcal{A}$'s advantage as non-uniform advice. Then, the reduction algorithm will only proceed if algorithm $\mathcal{A}$ chooses the particular value of ℓ_{out}. Since $\mathcal{A}$ is polynomially-bounded, there are only polynomially-many possible values for ℓ_{out} that it can possibly choose, so restricting $\mathcal{A}$ to a single value of ℓ_{out} for each security parameter can only reduce the advantage of $\mathcal{A}$ by a polynomial factor.

We define the Preprocess and Target algorithms as follows:

- Preprocess(pp, br): On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and a branch $\mathsf{br} \in \mathfrak{B}$, the preprocessing algorithm proceeds as follows:
  - Parse pp as $(\mathcal{G}, [\![\mathbf{M}]\!])$ in the manner described in Setup.
  - Recover $\mathbf{M} \in \mathbb{Z}_p^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$ from $[\![\mathbf{M}]\!]$ by computing discrete logarithms.

  Output $\mathsf{td} = (\mathbf{M}, \mathsf{br})$.
- Target(td, y): Given a trapdoor $\mathsf{td} = (\mathbf{M}, \mathsf{br})$ and a target string $y \in \{0,1\}^{\ell_{\mathsf{out}}}$:
  - If $\mathbf{M} + \mathsf{br} \cdot \mathbf{I}$ is not invertible, then abort and output $\pi = (\mathbf{0}^{\ell_{\mathsf{out}}}, \mathbf{0}^{\ell_s})$.
  - Sample a hash function $h \xleftarrow{R} \mathcal{H}$.
  - For each $i \in [\ell_{\mathsf{out}}]$, sample an element $t_i \xleftarrow{R} \mathbb{Z}_p$ such that $h([\![t_i]\!]) = y_i$. Concretely, we will use the following procedure: repeatedly sample $t_i \xleftarrow{R} \mathbb{Z}_p$ until finding one that satisfies $h([\![t_i]\!]) = y_i$. If for any index $i \in [\ell_{\mathsf{out}}]$, no such t_i is found after λ iterations, then set $\mathbf{t} = \mathbf{0}^{\ell_{\mathsf{out}}}$.

  Output $\pi = ((\mathbf{M} + \mathsf{br} \cdot \mathbf{I})^{-1}\mathbf{t},\ h)$.

We now show that (Preprocess, Target) satisfy the required properties.

Efficiency. The size of the trapdoor $\mathsf{td} = (\mathbf{M}, \mathsf{br})$ is $O(\ell_{\mathsf{out}}^2 \log p + \lambda) = \mathsf{poly}(\lambda, \ell_{\mathsf{out}})$. Moreover, since all steps of the Target algorithm run in $\mathsf{poly}(\lambda, \ell_{\mathsf{out}})$ time, it is efficient.

Most-branches target randomness. Before proving this property, we introduce some useful notation. Let $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ be the public parameters with $[\![\mathbf{M}]\!] = \mathsf{Samp}(1^\lambda, \mathsf{pp}_{\mathbf{M}})$ for some $[\![\mathbf{M}]\!] \in \mathbb{G}^{\ell_{\mathsf{out}} \times \ell_{\mathsf{out}}}$. We interpret each branch $\mathsf{br} \in \mathfrak{B}$ as an element of $\mathbb{Z}_p$ and define

$$\mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}} \coloneqq \{\mathsf{br} : \text{the matrix } \mathbf{M} + \mathsf{br} \cdot \mathbf{I} \text{ is not full rank}\}.$$

Next, for each hash function $h \in \mathcal{H}_\lambda$, value $y \in \{0,1\}^{\ell_{\mathsf{out}}}$, and branch $\mathsf{br} \in \mathfrak{B}$, define the preimage sets to be

$$R_{y,h,\mathsf{br}} \coloneqq \left\{\mathbf{r} : \mathbf{r} \in \mathbb{Z}_p^{\ell_{\mathsf{out}}}, h([\![(\mathbf{M} + \mathsf{br} \cdot \mathbf{I})\mathbf{r}]\!]) = y\right\}$$
$$T_{y,h} \coloneqq \left\{\mathbf{t} : \mathbf{t} \in \mathbb{Z}_p^{\ell_{\mathsf{out}}}, h([\![\mathbf{t}]\!]) = y\right\}$$

Suppose there exists a polynomial $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$ and a possibly unbounded adversary $\mathcal{A}$ that can break the target randomness of Construction 5.6. We define a sequence of hybrid experiments as follows:

- Hyb_1: This is the target randomness game for $b = 0$.
  1. On input the security parameter 1^λ and the output length $1^{\ell_{\mathsf{out}}}$, the adversary $\mathcal{A}$ sends pp to the challenger.
  2. The challenger computes $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}})$ and checks that $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$. Otherwise, it halts with output 0.
  3. The challenger parses pp as $(\mathcal{G}, [\![\mathbf{M}]\!])$ in the manner described in Setup. Then it works as follows:
     - Sample $\mathsf{br} \xleftarrow{R} \mathfrak{B}$, $h \xleftarrow{R} \mathcal{H}_\lambda$, and $\mathbf{r} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$.
     - Compute $y = h([\![(\mathbf{M} + \mathsf{br} \cdot \mathbf{I}) \cdot \mathbf{r}]\!])$.

     The challenger sends the branch br, the string y, and the proof $\pi = (\mathbf{r}, h)$ to the adversary $\mathcal{A}$.
  4. Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.
- Hyb_2: Same as Hyb_1 except the challenger changes the distribution of $\mathbf{r}$ in Step 3.
  3. The challenger parses pp as $(\mathcal{G}, [\![\mathbf{M}]\!])$ in the manner described in Setup. Then it works as follows:
     - Sample $\mathsf{br} \xleftarrow{R} \mathfrak{B}$, $h \xleftarrow{R} \mathcal{H}_\lambda$, and $\mathbf{r}' \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathsf{out}}}$.

- Compute $y = h(\llbracket (\mathbf{M} + \mathrm{br} \cdot \mathbf{I}) \cdot \mathbf{r}' \rrbracket)$
- Sample $\mathbf{r} \xleftarrow{R} R_{y,h,\mathrm{br}}$ if $R_{y,h,\mathrm{br}} \neq \emptyset$ and set $\mathbf{r} = \mathbf{0}^{\ell_{\mathrm{out}}}$ otherwise.

The challenger sends the branch br, the string y, and the proof $\pi = (\mathbf{r}, h)$ to the adversary $\mathcal{A}$.

- Hyb_3: Same as Hyb_2 except the challenger aborts if $\mathrm{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$.

3. The challenger parses pp as $(\mathcal{G}, \llbracket \mathbf{M} \rrbracket)$ in the manner described in Setup. Then it works as follows:
   - Sample $\mathrm{br} \xleftarrow{R} \mathfrak{B}$.
   - If $\mathrm{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $y \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$, and $\pi = (\mathbf{0}^{\ell_{\mathrm{out}}}, \mathbf{0}^{\ell_s})$ to the adversary $\mathcal{A}$.
   - Sample $h \xleftarrow{R} \mathcal{H}_\lambda$ and $\mathbf{r}' \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathrm{out}}}$
   - Compute $y = h(\llbracket (\mathbf{M} + \mathrm{br} \cdot \mathbf{I}) \cdot \mathbf{r}' \rrbracket)$
   - Sample $\mathbf{r} \xleftarrow{R} R_{y,h,\mathrm{br}}$ if $R_{y,h,\mathrm{br}} \neq \emptyset$ and set $\mathbf{r} = \mathbf{0}^{\ell_{\mathrm{out}}}$ otherwise.

   The challenger sends the branch br, the string y, and the proof $\pi = (\mathbf{r}, h)$ to the adversary $\mathcal{A}$.

- Hyb_4: Same as Hyb_3 except the challenger changes the distribution of the input to the hash function h.

3. The challenger parses pp as $(\mathcal{G}, \llbracket \mathbf{M} \rrbracket)$ in the manner described in Setup. Then it works as follows:
   - Sample $\mathrm{br} \xleftarrow{R} \mathfrak{B}$.
   - If $\mathrm{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $y \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$, and $\pi = (\mathbf{0}^{\ell_{\mathrm{out}}}, \mathbf{0}^{\ell_s})$ to the adversary $\mathcal{A}$.
   - Sample $h \xleftarrow{R} \mathcal{H}_\lambda$
   - Sample $\mathbf{t} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\mathrm{out}}}$
   - Compute $y = h(\llbracket \mathbf{t} \rrbracket)$
   - Sample $\mathbf{r} \xleftarrow{R} R_{y,h,\mathrm{br}}$ if $R_{y,h,\mathrm{br}} \neq \emptyset$ and set $\mathbf{r} = \mathbf{0}^{\ell_{\mathrm{out}}}$ otherwise.

   The challenger sends the branch br, the string y, and the proof $\pi = (\mathbf{r}, h)$ to the adversary $\mathcal{A}$.

- Hyb_5: Same as Hyb_4 except the challenger changes the distribution of y in Step 3.

3. The challenger parses pp as $(\mathcal{G}, \llbracket \mathbf{M} \rrbracket)$ in the manner described in Setup. Then it works as follows:
   - Sample $y \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$
   - Sample $\mathrm{br} \xleftarrow{R} \mathfrak{B}$.
   - If $\mathrm{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $y \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$, and $\pi = (\mathbf{0}^{\ell_{\mathrm{out}}}, \mathbf{0}^{\ell_s})$ to the adversary $\mathcal{A}$.
   - Sample $h \xleftarrow{R} \mathcal{H}_\lambda$
   - Sample $\mathbf{r} \xleftarrow{R} R_{y,h,\mathrm{br}}$ if $R_{y,h,\mathrm{br}} \neq \emptyset$ and set $\mathbf{r} = \mathbf{0}^{\ell_{\mathrm{out}}}$ otherwise.

   The challenger sends the branch br, the string y, and the proof $\pi = (\mathbf{r}, h)$ to the adversary $\mathcal{A}$.

- Hyb_6: Same as Hyb_5 except the challenger changes the distribution of $\mathbf{r}$ in Step 3.

3. The challenger parses pp as $(\mathcal{G}, \llbracket \mathbf{M} \rrbracket)$ in the manner described in Setup. Then it works as follows:
   - Sample $y \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$
   - Sample $\mathrm{br} \xleftarrow{R} \mathfrak{B}$
   - If $\mathrm{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $y \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$, and $\pi = (\mathbf{0}^{\ell_{\mathrm{out}}}, \mathbf{0}^{\ell_s})$ to the adversary $\mathcal{A}$.
   - Recover $\mathbf{M}$ from $\llbracket \mathbf{M} \rrbracket$ by computing discrete logarithms.
   - Sample $h \xleftarrow{R} \mathcal{H}_\lambda$
   - Sample $\mathbf{t} \xleftarrow{R} T_{y,h}$ if $T_{y,h} \neq \emptyset$ and set $\mathbf{t} = \mathbf{0}^{\ell_{\mathrm{out}}}$ otherwise. Set $\mathbf{r} = (\mathbf{M} + \mathrm{br} \cdot \mathbf{I})^{-1}\mathbf{t}$.

   The challenger sends the branch br, the string y, and the proof $\pi = (\mathbf{r}, h)$ to the adversary $\mathcal{A}$.

- Hyb_7: Same as Hyb_6 except the challenger changes the distribution of $\mathbf{t}$ in Step 3.

3. The challenger parses pp as $(\mathcal{G}, \llbracket \mathbf{M} \rrbracket)$ in the manner described in Setup. Then it works as follows:

- Sample $y \xleftarrow{R} \{0,1\}^{\ell_{\text{out}}}$
- Sample $\text{br} \xleftarrow{R} \mathfrak{B}$
- If $\text{br} \in \mathfrak{B}_{\text{pp}}^{\text{BAD}}$ then abort and send br, $y \xleftarrow{R} \{0,1\}^{\ell_{\text{out}}}$, and $\pi = (\mathbf{0}^{\ell_{\text{out}}}, \mathbf{0}^{\ell_s})$ to the adversary $\mathcal{A}$.
- Recover $\mathbf{M}$ from $[\![\mathbf{M}]\!]$ by computing discrete logarithms.
- Sample $h \xleftarrow{R} \mathcal{H}_\lambda$
- For each $i \in [\ell_{\text{out}}]$, repeatedly sample $t_i \xleftarrow{R} \mathbb{Z}_p$ until finding one that satisfies $h([\![t_i]\!]) = y_i$. If for any index $i \in [\ell_{\text{out}}]$, no such t_i is found after λ iterations, then set $\mathbf{t} = \mathbf{0}^{\ell_{\text{out}}}$. Set $\mathbf{r} = (\mathbf{M} + \text{br} \cdot \mathbf{I})^{-1}\mathbf{t}$.

The challenger sends the branch br, the string y, and the proof $\pi = (\mathbf{r}, h)$ to the adversary $\mathcal{A}$.

This is the target randomness game for $b = 1$.

Let $\text{Hyb}_i(\mathcal{A})$ be the random variable denoting the output of an execution of experiment Hyb_i with adversary $\mathcal{A}$. We now analyze each adjacent pair of hybrid experiments.

Claim 5.13. *It holds that* $\Pr[\text{Hyb}_1(\mathcal{A}) = 1] = \Pr[\text{Hyb}_2(\mathcal{A}) = 1]$.

Proof. The only difference between these two distributions is how they sample $\mathbf{r} \in \mathbb{Z}_p^{\ell_{\text{out}}}$. Fix any br, h and let $\tilde{R}$ be the random variable denoting the value of $\mathbf{r}$ in the experiments. Consider the distribution of $\tilde{R}$ in the two experiments:

- In Hyb_1, the distribution of $\tilde{R}$ is uniform over $\mathbb{Z}_p^{\ell_{\text{out}}}$.
- In Hyb_2, take any $\mathbf{r} \in \mathbb{Z}_p^{\ell_{\text{out}}}$, let $y^* = h([\![(\mathbf{M} + \text{br} \cdot \mathbf{I})\mathbf{r}]\!])$. Consider now the distribution of $\tilde{R}$. By definition, the event $\tilde{R} = \mathbf{r}$ occurs when $\mathbf{r}' \in R_{y^*,h,\text{br}}$ and then $\mathbf{r}$ is sampled by drawing randomly from $R_{y^*,h,\text{br}}$. Since the challenger in Hyb_2 samples $\mathbf{r}' \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}$, this occurs with probability

$$\Pr[\tilde{R} = \mathbf{r}] = \Pr[\mathbf{r}' \in R_{y^*,h,\text{br}} : \mathbf{r}' \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}] \cdot \frac{1}{|R_{y^*,h,\text{br}}|} = \frac{|R_{y^*,h,\text{br}}|}{p^{\ell_{\text{out}}}} \cdot \frac{1}{|R_{y^*,h,\text{br}}|} = \frac{1}{p^{\ell_{\text{out}}}}$$

  Thus, the distribution of $\tilde{R}$ is also uniform over $\mathbb{Z}_p^{\ell_{\text{out}}}$.

We conclude that in both experiments, the distribution of $\mathbf{r}$ is uniform over $\mathbb{Z}_p^{\ell_{\text{out}}}$. Thus, these distributions are identically distributed and the claim follows. We note that the condition $R_{y,h,\text{br}} = \emptyset$ never triggers in Hyb_2 because $\mathbf{r}' \in R_{y,h,\text{br}}$ by definition of the hybrid. □

Claim 5.14. *Suppose* $1/N(\lambda) = \text{negl}(\lambda)$. *For all polynomials* $\ell_{out} = \ell_{out}(\lambda)$, *there exists a negligible function* $\text{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\text{Hyb}_2(\mathcal{A}) = 1] - \Pr[\text{Hyb}_3(\mathcal{A}) = 1]| = \text{negl}(\lambda)$.

Proof. The hybrids differ only if the sampled branch br lies in the set $\mathfrak{B}_{\text{pp}}^{\text{BAD}} \coloneqq \{\text{br} : \det(\mathbf{M} + \text{br} \cdot \mathbf{I}) = 0\}$. By Lemma 5.12, we have $|\mathfrak{B}_{\text{pp}}^{\text{BAD}}| \leq \ell_{\text{out}}$. Since br is sampled uniformly from a set of size $N(\lambda)$, the probability that $\text{br} \in \mathfrak{B}_{\text{pp}}^{\text{BAD}}$ is at most $\ell_{\text{out}}/N(\lambda)$. As ℓ_{out} is polynomial in λ and $1/N(\lambda) = \text{negl}(\lambda)$, this probability is negligible in λ. Therefore, the statistical distance between the two hybrids is negligible, and the claim follows. □

Claim 5.15. *It holds that* $\Pr[\text{Hyb}_3(\mathcal{A}) = 1] = \Pr[\text{Hyb}_4(\mathcal{A}) = 1]$.

Proof. The only difference between the hybrids lies in the generation of the input to the hash function h. In Hyb_3, we compute this input as $\mathbf{t} = (\mathbf{M} + \text{br} \cdot \mathbf{I}) \cdot \mathbf{r}'$ where $\mathbf{r}' \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}$, whereas in Hyb_4 we directly sample $\mathbf{t} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}$. By definition, for any $\text{br} \notin \mathfrak{B}_{\text{pp}}^{\text{BAD}}$ the matrix $\mathbf{M} + \text{br} \cdot \mathbf{I}$ is invertible over $\mathbb{Z}_p$. This implies that for $\mathbf{r}' \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}$, $(\mathbf{M} + \text{br} \cdot \mathbf{I}) \cdot \mathbf{r}'$ is distributed uniformly over $\mathbb{Z}_p^{\ell_{\text{out}}}$. Thus the distributions of $\mathbf{t} = (\mathbf{M} + \text{br} \cdot \mathbf{I}) \cdot \mathbf{r}'$ for $\mathbf{r}' \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}$ and $\mathbf{t} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}$ are identical and the claim follows. □

Claim 5.16. *Suppose* $\mathcal{H}_\lambda$ *is a universal hash function family. Then for all polynomials* $\ell_{out} = \ell_{out}(\lambda)$, *there exists a negligible function* $\text{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\text{Hyb}_4(\mathcal{A}) = 1] - \Pr[\text{Hyb}_5(\mathcal{A}) = 1]| = \text{negl}(\lambda)$.

Proof. The only difference between the hybrids lies in the way y is sampled. In Hyb_4 we sample $\mathbf{t} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}$ and set $y = h(\llbracket \mathbf{t} \rrbracket)$ while in Hyb_5 we sample $y \xleftarrow{R} \{0,1\}^{\ell_{\text{out}}}$. Since $\mathcal{H}_\lambda$ is a universal hash function and $\log p = \Theta(\lambda) \geq 1 + \omega(\log \lambda)$, it is statistically uniform (Corollary 2.6). Then the statistical distance between the joint distribution of $(h, h(\llbracket \mathbf{t} \rrbracket))$ and (h, y) when $h \xleftarrow{R} \mathcal{H}_\lambda, \mathbf{t} \xleftarrow{R} \mathbb{Z}_p^{\ell_{\text{out}}}$, and $y \xleftarrow{R} \{0,1\}^{\ell_{\text{out}}}$ is at most $\ell_{\text{out}} \cdot \mathsf{negl}(\lambda)$ by a standard hybrid argument. Since ℓ_{out} is polynomial in λ, the claim follows. □

Claim 5.17. *It holds that* $\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]$.

Proof. The proof is similar to that of Claim 5.15. It suffices to show that the distribution of $\mathbf{r}$ is identical in both hybrids. Fix $y \in \{0,1\}^{\ell_{\text{out}}}$, $h \in \mathcal{H}_\lambda$, and $\mathsf{br} \in \mathfrak{B} \setminus \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$. Since $\mathbf{M} + \mathsf{br} \cdot \mathbf{I}$ is invertible, the map $\mathbf{r} \mapsto (\mathbf{M} + \mathsf{br} \cdot \mathbf{I})\mathbf{r}$ induces a bijection between $R_{y,h,\mathsf{br}}$ and $T_{y,h}$. In particular, $T_{y,h} = \emptyset$ if and only if $R_{y,h,\mathsf{br}} = \emptyset$, and in this case both hybrids set $\mathbf{r} = \mathbf{0}^{\ell_{\text{out}}}$. Otherwise, when both sets are non-empty, sampling $\mathbf{t} \xleftarrow{R} T_{y,h}$ uniformly and setting $\mathbf{r} = (\mathbf{M} + \mathsf{br} \cdot \mathbf{I})^{-1}\mathbf{t}$ yields the uniform distribution over $R_{y,h,\mathsf{br}}$. This is exactly the distribution obtained by sampling $\mathbf{r} \xleftarrow{R} R_{y,h,\mathsf{br}}$ directly, and the claim follows. □

Claim 5.18. *Suppose $\mathcal{H}_\lambda$ is a universal hash function family. Then for all polynomials $\ell_{out} = \ell_{out}(\lambda)$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. The only difference between the hybrids lies in the distribution of $\mathbf{t}$. Fix a target $y \in \{0,1\}^{\ell_{\text{out}}}$ and a hash function h. We consider two cases:

- Suppose $T_{y,h} = \emptyset$. This means that there must exist at least one index i such that for all $\mathbf{t} \in \mathbb{Z}_p^{\ell_{\text{out}}}$, we have $h(\llbracket t_i \rrbracket) \neq y_i$. In this case, both hybrids set $\mathbf{t} = \mathbf{0}^{\ell_{\text{out}}}$. The challenger in both experiments then sets $\mathbf{r} = (\mathbf{M} + \mathsf{br} \cdot \mathbf{I})^{-1}\mathbf{t}$ and outputs $\pi = (\mathbf{r}, h)$. Thus, in this case, the challenger's behavior in the two experiments is identical.
- Suppose $T_{y,h} \neq \emptyset$. Observe that since h is applied component-wise, the condition $h(\llbracket \mathbf{t} \rrbracket) = y$ decouples into independent constraints for each component. Specifically, define the component preimage sets $T_{y_i,h} \coloneqq \{t_i \in \mathbb{Z}_p : h(\llbracket t_i \rrbracket) = y_i\}$. Then, we can write $T_{y,h} = T_{y_1,h} \times \cdots \times T_{y_{\ell_{\text{out}}},h}$. Then, sampling $\mathbf{t} \xleftarrow{R} T_{y,h}$ is equivalent to independently sampling each component $t_i \xleftarrow{R} T_{y_i,h}$.

  Now, we show that the sampling procedure for $\mathbf{t}$ in Hyb_7 yields a distribution that is statistically close to the distribution of $\mathbf{t}$ in Hyb_6. Since h is applied component-wise, we analyze the failure probability for a single component $i \in [\ell_{\text{out}}]$. In Hyb_7, the algorithm fails only if it cannot find a preimage t_i such that $h(\llbracket t_i \rrbracket) = y_i$ after λ iterations.

  To bound this failure probability, we rely on the statistical uniformity of $\mathcal{H}_\lambda$. Specifically, since $\mathcal{H}_\lambda$ is a universal hash function and $\log p = \Theta(\lambda) \geq 1 + \omega(\log \lambda)$, it is statistically uniform (Corollary 2.6). Specifically, let Bad be the event that the sampled h is sufficiently unbalanced such that $\Pr[h(\llbracket t_i \rrbracket) = 0 : t_i \xleftarrow{R} \mathbb{Z}_p] < 1/3$ or $\Pr[h(\llbracket t_i \rrbracket) = 1 : t_i \xleftarrow{R} \mathbb{Z}_p] < 1/3$. Since $\mathcal{H}_\lambda$ is statistically uniform, $\Pr[\mathsf{Bad} : h \xleftarrow{R} \mathcal{H}_\lambda] = \mathsf{negl}(\lambda)$.

  Conditioned on $h \notin \mathsf{Bad}$, the probability of failing to find a suitable t_i after λ independent samples is at most $(2/3)^\lambda = \mathsf{negl}(\lambda)$. Finally, applying a union bound over all ℓ_{out} components (where $\ell_{\text{out}} = \mathsf{poly}(\lambda)$), the total failure probability remains negligible. Thus the claim follows. □

Proof of Theorem 5.11. Combining Claims 5.13 to 5.18, we conclude via a hybrid argument that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$$

and the claim follows. □

Theorem 5.19 (Sampling on Constricted Branch)**.** *For all polynomials $\ell_s(\lambda)$ such that $|\mathcal{H}_\lambda| \leq 2^{\ell_s(\lambda)}$, Construction 5.6 satisfies sampling on constricted branch.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, output length ℓ_{out}, and branch $\text{br}^* \in \mathfrak{B}$. Let (pp, aux) be in the support of $\text{Setup}(1^\lambda, 1^{\ell_{\text{out}}}, \text{br}^*)$. By construction and Definition 5.3, on parsing pp as $(\mathcal{G}, [\![\mathbf{M}]\!])$ in the manner described in Setup we have $[\![\mathbf{M}]\!] = [\![\mathbf{u}\mathbf{v}^\intercal - \text{br}^* \cdot \mathbf{I}]\!]$, and $\text{aux} = (\mathcal{G}, [\![\mathbf{u}]\!])$. Define the set

$$\text{Range} \coloneqq \left\{ h(\alpha \cdot [\![\mathbf{u}]\!]) : h \in \mathcal{H}_\lambda,\ \alpha \in \mathbb{Z}_p \right\}.$$

Note that $|\text{Range}| \le p \cdot |\mathcal{H}_\lambda|$. Fix any $y^* \in \{0,1\}^{\ell_{\text{out}}}$ and proof π^* such that $\text{Verify}(\text{pp}, \text{br}^*, y^*, \pi^*) = 1$. By correctness of verification, $\pi^* = (\mathbf{r}^*, h^*)$ for some $\mathbf{r}^* \in \mathbb{Z}_p^{\ell_{\text{out}}}$ and $h^* \in \mathcal{H}_\lambda$, and moreover

$$y^* = h^*([\![(\mathbf{M} + \text{br}^* \cdot \mathbf{I}) \cdot \mathbf{r}^*]\!]) = h^*([\![\mathbf{u}\mathbf{v}^\intercal \mathbf{r}^*]\!]) = h^*(\mathbf{v}^\intercal \mathbf{r}^* \cdot [\![\mathbf{u}]\!]).$$

Let $\alpha^* \coloneqq \mathbf{v}^\intercal \mathbf{r}^* \in \mathbb{Z}_p$. Then $y^* = h^*(\alpha^* \cdot [\![\mathbf{u}]\!])$, and hence $y^* \in \text{Range}$. The SampleGuess algorithm samples $h \xleftarrow{\text{R}} \mathcal{H}_\lambda$ and $\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$ uniformly and independently, and outputs $y = h(\alpha \cdot [\![\mathbf{u}]\!])$. Therefore,

$$\Pr[y = y^* : y \leftarrow \text{SampleGuess}(\text{aux})] \ \ge\ \Pr[h = h^* \wedge \alpha = \alpha^*] \ =\ \frac{1}{|\mathcal{H}_\lambda| \cdot p}.$$

Since $|\mathcal{H}_\lambda| \le 2^{\ell_s(\lambda)}$ and $p = 2^{\Theta(\lambda)}$, letting $q(\lambda) = \ell_s(\lambda) + \Theta(\lambda)$ yields

$$\Pr[y = y^* : y \leftarrow \text{SampleGuess}(\text{aux})] \ge 2^{-q(\lambda)}.$$

Since $\ell_s(\lambda) = \text{poly}(\lambda)$, we conclude that $q(\lambda) = \text{poly}(\lambda)$. Thus the claim follows. □

Parameter instantiation. Let λ be the security parameter and let ℓ_{out} denote the output length. We now provide one possible instantiation of the parameters in Construction 5.6 to satisfy the requirements in Theorems 5.7, 5.8, 5.11 and 5.19.

- Let GroupGen be an obliviously-sampleable public-coin group generator where the ε-DDH assumption holds for some negligible function $\varepsilon(\lambda) = \text{negl}(\lambda)$.
- We set the branch set size N to be $N(\lambda) = 1/\varepsilon(\lambda)$.
- Let $\ell_{\mathbb{G}}(\lambda) = O(\lambda)$ be the bit-length of the representation of a group element. For each $\lambda \in \mathbb{N}$, let $\mathcal{H}_\lambda = \{h_s : s \in \{0,1\}^{\ell_{\mathbb{G}}(\lambda)}\}$ where $h_s(x) \coloneqq \langle s, x \rangle \bmod 2$. Let $\mathcal{H} = \{\mathcal{H}_\lambda\}_{\lambda \in \mathbb{N}}$.

We briefly verify that these parameters satisfy the necessary requirements:

- Since GroupGen is obliviously sampleable and the ε-MDDH assumption holds with respect to GroupGen, the conditions of Theorem 5.8 are satisfied. In particular, ε'-mode indistinguishability holds with

$$\varepsilon' = \max(2^{-\lambda}, \varepsilon(\lambda)) = \text{negl}(\lambda).$$

- For Theorem 5.11, observe that $\mathcal{H}$ is universal and that $1/N(\lambda) = \varepsilon(\lambda) = \text{negl}(\lambda)$.
- For Theorem 5.19, note that $\log p = O(\lambda)$ so $|\mathcal{H}_\lambda| = 2^{O(\lambda)}$, as required.

Corollary 5.20 (Branch-Constricting Generator from DDH)**.** *Let λ be a security parameter and take any negligible function $\varepsilon = \varepsilon(\lambda) = \text{negl}(\lambda)$. If the DDH assumption is ε-hard with respect to an obliviously-sampleable public-coin group generator* GroupGen*, then there exists a branch-constricting generator that satisfies ε-mode indistinguishability with a branch set of size $N(\lambda) = 1/\varepsilon(\lambda)$.*

In conjunction with Corollary 4.8, we now obtain a sometimes-constricting generator from the DDH assumption:

Corollary 5.21 (Sometimes-Constricting Generator from DDH)**.** *Let λ be a security parameter and take any negligible function $\varepsilon = \varepsilon(\lambda) = \text{negl}(\lambda)$. If the DDH assumption is ε-hard with respect to an obliviously-sampleable public-coin group generator* GroupGen*, then there exists a polynomial $p = p(\lambda)$ and a sometimes-constricting generator that satisfies completeness, mode indistinguishability, μ-guessing security, and target randomness where $\mu(\lambda) = 2^{-p(\lambda)} \cdot \varepsilon(\lambda)$.*

6 Branch-Constricting Generator from LWE

In this section, we show how to construct a branch-constricting generator from the learning with errors (LWE) assumption [Reg05]. We start with some preliminaries.

6.1 Lattice Preliminaries

For a vector $\mathbf{v} \in \mathbb{Z}^n$, we write $\|\mathbf{v}\|$ to denote the ℓ_∞-norm of $\mathbf{v}$. If $\mathbf{v} \in \mathbb{Z}_q^n$, we write $\|\mathbf{v}\|$ to denote the ℓ_∞-norm of the vector over $\mathbb{Z}^n$ obtained by first associating each component $v_i \in \mathbb{Z}_q$ with its unique representative in the set $(-q/2, q/2] \cap \mathbb{Z}$. For a matrix $\mathbf{A} \in \mathbb{Z}^{n\times m}$, we write $\|\mathbf{A}\| = \max_{i\in[m]} \|\mathbf{a}_i\|$ where $\mathbf{a}_i$ is the i^{th} column of $\mathbf{A}$. For two matrices $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{B} \in \mathbb{Z}_q^{k\times \ell}$ we write $\mathbf{A} \otimes \mathbf{B}$ to denote their tensor (or Kronecker) product.

Sampling random vectors. Next, we describe an algorithm for sampling random elements over $\mathbb{Z}_q$ from a uniform random bit-string. We will use these algorithms to construct our public-coin branch-constricting generator.

Definition 6.1 (Sampling Algorithms)**.** We define the algorithms $\mathsf{Samp}, \mathsf{Samp}^{-1}$ (for converting between a bit string and an integer) as follows:

- $\mathsf{Samp}(1^\lambda, q, t, r) \to s$: On input the security parameter 1^λ, a modulus q, a length parameter $t \geq 0$, and a bitstring $r \in \{0,1\}^{\lceil \log q\rceil+\lambda+t}$, the Samp algorithm interprets r as the binary representation of a $(\lceil \log q\rceil + \lambda + t)$-bit integer and outputs $s = r \bmod q$.
- $\mathsf{Samp}^{-1}(1^\lambda, q, t, s) \to r$: On input the security parameter 1^λ, modulus q, a length $t \geq 0$, and an integer $s \in \mathbb{Z}_q$, the Samp^{-1} algorithm samples $\gamma \xleftarrow{R} \{0, \ldots, \lfloor 2^{\lceil \log q\rceil+\lambda+t}/q\rfloor - 1\}$. It sets $r = q \cdot \gamma + s$ where r is represented as a binary string in $\{0,1\}^{\lceil \log q\rceil+\lambda+t}$.

By construction, algorithm Samp is deterministic and Samp^{-1} is randomized. In addition, they satisfy the following properties:

- **Perfect correctness**: For all $\lambda, q, t \in \mathbb{N}$ and all $s \in \mathbb{Z}_q$,
$$\Pr[\mathsf{Samp}(1^\lambda, q, t, r) = s : r \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, s)] = 1.$$
- **Statistical indistinguishability**: There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the statistical distance between the following distributions is at most $2^{-t} \cdot \mathsf{negl}(\lambda)$:
$$\left\{(r, \mathsf{Samp}(1^\lambda, q, t, r)) : r \xleftarrow{R} \{0,1\}^{\lceil \log q\rceil+\lambda+t}\right\} \quad \text{and} \quad \left\{(\mathsf{Samp}^{-1}(1^\lambda, q, t, s), s) : s \xleftarrow{R} \mathbb{Z}_q\right\}.$$
Note that this can be extended to polynomially many samples via a standard hybrid argument.

We extend these functions to matrices in a natural way by sampling component-wise.

Rounding. We associate $\mathbb{Z}_q$ with the interval $(-q/2, q/2] \cap \mathbb{Z}$. For ease of exposition, we define the following sets, each parameterized by a modulus q and a bound $B_R \in \mathbb{N}$:

$$\mathsf{Good}^-_{q,B_R} \coloneqq (-\lfloor q/2\rfloor + B_R, -B_R) \cap \mathbb{Z}_q \tag{6.1}$$
$$\mathsf{Good}^+_{q,B_R} \coloneqq (B_R, \lfloor q/2\rfloor - B_R) \cap \mathbb{Z}_q \tag{6.2}$$
$$\mathsf{Good}_{q,B_R} \coloneqq \mathsf{Good}^-_{q,B_R} \cup \mathsf{Good}^+_{q,B_R}. \tag{6.3}$$

Next, we define the following rounding functions $\mathsf{RoundS}\colon \mathbb{Z}_q \to \{0,1\}$ and $\mathsf{RoundB}_{q,B_R}\colon \mathbb{Z}_q \to \{0,1,\perp\}$ as follows:

$$\mathsf{RoundS}(w) = \begin{cases} 0 & \text{if } w \leq 0 \\ 1 & \text{otherwise} \end{cases} \qquad \text{and} \qquad \mathsf{RoundB}_{q,B_R}(w) = \begin{cases} 0 & \text{if } w \in \mathsf{Good}^-_{q,B_R} \\ 1 & \text{if } w \in \mathsf{Good}^+_{q,B_R} \\ \perp & \text{otherwise.} \end{cases} \tag{6.4}$$

We also define a reverse-sampling algorithm $\mathsf{RoundB}^{-1}_{q,B_R}$ as follows:

On input $y \in \{0,1\}$:

- If $y = 0$ output $w \xleftarrow{R} \mathrm{Good}^{-}_{q,B_R}$.
- If $y = 1$ output $w \xleftarrow{R} \mathrm{Good}^{+}_{q,B_R}$.

Figure 1: The reverse-sampling algorithm $\mathsf{RoundB}^{-1}_{q,B_R}$.

We extend the rounding functions to operate on matrices in a component-wise manner.

Discrete Gaussians over lattices. We use $D_{\mathbb{Z},\chi}$ to denote the discrete Gaussian distribution over $\mathbb{Z}$ with width parameter $\chi > 0$. Furthermore, let $\tilde{D}_{\mathbb{Z},\chi,B_G}$ denote the truncated discrete Gaussian defined by the following sampling procedure:

- Sample $x \leftarrow D_{\mathbb{Z},\chi}$.
- If $|x| < B_G$ output x, else output 0.

Lemma 6.2 (Gaussian Tail Bound [MP12, Lemma 2.6, adapted])**.** *For all $\chi > 0$ and $\lambda \in \mathbb{N}$*

$$\Pr[|x| > \sqrt{\lambda}\chi : x \leftarrow D_{\mathbb{Z},\chi}] < 2^{-\lambda}$$

By Lemma 6.2, the truncated discrete Gaussian distribution $\tilde{D}_{\mathbb{Z},\chi,B_G}$ with $B_G \geq \sqrt{\lambda}\chi$ is statistically close to the discrete Gaussian distribution $D_{\mathbb{Z},\chi}$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a vector $\mathbf{v} \in \mathbb{Z}_q^n$ in the column space of $\mathbf{A}$, we write $\mathbf{A}^{-1}_{\chi}(\mathbf{v})$ to denote a random variable $\mathbf{x} \leftarrow D^m_{\mathbb{Z},\chi}$ conditioned on $\mathbf{Ax} = \mathbf{v} \bmod q$.

The gadget matrix. We recall the definition of the gadget matrix [MP12]. For positive integers $n, q \in \mathbb{N}$, let $\mathbf{G}_n = \mathbf{I}_n \otimes \mathbf{g}^\intercal \in \mathbb{Z}_q^{n\times m'}$ be the gadget matrix where $\mathbf{g}^\intercal := [1, 2, \ldots, 2^{\lceil \log q \rceil - 1}]$ and $m' = n\lceil \log q \rceil$. The inverse function $\mathbf{G}_n^{-1} : \mathbb{Z}_q^{n\times t} \to \mathbb{Z}_q^{m'\times t}$ expands each entry $x \in \mathbb{Z}_q$ into a column of size $\lceil \log q \rceil$ consisting of the bits in the binary representation of x. By construction, for every matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times t}$, it follows that $\mathbf{G}_n \cdot \mathbf{G}_n^{-1}(\mathbf{A}) = \mathbf{A} \bmod q$. When the lattice dimension n is clear, we drop the subscript n.

Gadget trapdoors. Our constructions will use the gadget trapdoors from [MP12], which builds on a long sequence of works on constructing lattice trapdoors [Ajt96, GPV08, AP09, ABB10a, ABB10b, CHKP10].

Theorem 6.3 (Gadget Trapdoors [MP12, adapted])**.** *Let n, m, q, χ be lattice parameters with $m \geq 3n\lceil \log q \rceil$. Then there exists an efficient algorithm* SamplePre *with the following syntax:*

- $\mathsf{SamplePre}(\mathbf{A}, \mathbf{R}_\mathbf{A}, \mathbf{y}, \chi) \to \mathbf{x}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, a trapdoor $\mathbf{R}_\mathbf{A}$, a target vector $\mathbf{y} \in \mathbb{Z}_q^n$, and a Gaussian width parameter χ, the preimage sampling algorithm outputs a vector $\mathbf{x} \in \mathbb{Z}_q^m$.*

For $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, suppose there exists $\mathbf{R}_\mathbf{A} \in \mathbb{Z}_q^{m\times m'}$ such that $\mathbf{AR}_\mathbf{A} = \mathbf{G}$. We call $\mathbf{R}_\mathbf{A}$ a "gadget trapdoor" for $\mathbf{A}$. Then, the SamplePre *algorithm satisfies the following properties:*

- ***Preimage sampling:*** *For all width parameters $\chi > 0$, if we sample $\mathbf{x} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{R}_\mathbf{A}, \mathbf{y}, \chi)$, then $\mathbf{Ax} = \mathbf{y}$.*
- ***Preimage distribution:*** *For all $\chi \geq m\,\|\mathbf{R}_\mathbf{A}\| \log n$, and all target vectors $\mathbf{y} \in \mathbb{Z}_q^n$, the statistical distance between the following distributions is at most 2^{-n}:*

$$\{\mathbf{x} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{R}_\mathbf{A}, \mathbf{y}, \chi)\} \quad \textit{and} \quad \left\{\mathbf{x} \leftarrow \mathbf{A}^{-1}_{\chi}(\mathbf{y})\right\}.$$

Gaussian samples. We also state the following lemma, which is a consequence of [GPV08, Lemma 5.2]. We prove it in Appendix B:

Lemma 6.4 (Gaussian Samples)**.** *Let n, m, q, χ be lattice parameters. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be a matrix with a gadget trapdoor $\mathbf{R}_\mathbf{A}$. If $\chi \geq 4m \|\mathbf{R}_\mathbf{A}\| \cdot \log m$ then there exists a negligible function $\mathsf{negl}(\cdot)$ such that the statistical distance between the following distributions is $\mathsf{negl}(m)$:*

$$\left\{(\mathbf{x}, \mathbf{A}\mathbf{x} \bmod q) : \mathbf{x} \leftarrow D_{\mathbb{Z},\chi}^m\right\} \quad \textit{and} \quad \left\{(\mathbf{x}, \mathbf{z}) : \mathbf{z} \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{x} \leftarrow \mathbf{A}_\chi^{-1}(\mathbf{z})\right\}.$$

Learning with errors. We now recall the learning with errors assumption [Reg05].

Assumption 6.5 (Learning With Errors [Reg05])**.** Let λ be a security parameter and $n = n(\lambda), m = m(\lambda), q = q(\lambda), \chi = \chi(\lambda)$ be lattice parameters. We say the ε-learning with errors assumption (ε-$\mathsf{LWE}_{n,m,q,\chi}$) with parameters (n, m, q, χ) holds if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathbf{A}, \mathbf{s}^\intercal\mathbf{A} + \mathbf{e}^\intercal) \to 1 : \begin{array}{c}\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m} \\ \mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{e} \leftarrow D_{\mathbb{Z},\chi}^m\end{array}\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathbf{A}, \mathbf{u}^\intercal) \to 1 : \begin{array}{c}\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m} \\ \mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m\end{array}\right]\right| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

Similar to the DDH case, we can extend the LWE assumption to polynomially-many samples of independent secrets and error vectors. The following result follows by a standard hybrid argument from Assumption 6.5:

Assumption 6.6 (Matrix Learning With Errors)**.** Let λ be a security parameter and $n = n(\lambda), m = m(\lambda), q = q(\lambda), \chi = \chi(\lambda)$ be lattice parameters. We say the ε-matrix learning with errors (ε-$\mathsf{MLWE}_{n,m,q,\chi}$) assumption with parameters (n, m, q, χ) holds if for all polynomials $\tau = \tau(\lambda)$ and all efficient (and possibly non-uniform) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathbf{A}, \mathbf{S}\mathbf{A} + \mathbf{E}) \to 1 : \begin{array}{c}\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m} \\ \mathbf{S} \xleftarrow{R} \mathbb{Z}_q^{\tau\times n}, \mathbf{E} \leftarrow D_{\mathbb{Z},\chi}^{\tau\times m}\end{array}\right] - \Pr\left[\mathcal{A}(1^\lambda, (\mathbf{A}, \mathbf{U})) \to 1 : \begin{array}{c}\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m} \\ \mathbf{U} \xleftarrow{R} \mathbb{Z}_q^{\tau\times m}\end{array}\right]\right| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$$

6.2 Construction of Branch-Constricting Generator from LWE

We now show how to construct a branch-constricting generator from LWE.

Construction 6.7 (Branch-Constricting Generator from LWE)**.** Let λ be a security parameter and ℓ_{out} be an output length. Define the following parameters:

- Let $N = 8\lambda$ and $k = k(\lambda, \ell_{\mathsf{out}})$ be a polynomial. We will use these parameters to define our branch set below, where N will denote the number of branch indices and k the chunk width.
- Let $n = n(\lambda, \ell_{\mathsf{out}})$ be the LWE dimension, $m = k \cdot N \cdot \ell_{\mathsf{out}}$ be the LWE matrix width, $q = q(\lambda, \ell_{\mathsf{out}})$ be the LWE modulus (we assume it to be an odd prime throughout the construction) and $\chi = \chi(\lambda, \ell_{\mathsf{out}})$ be the Gaussian width parameter.
- Let $B_M = B_M(\lambda, \ell_{\mathsf{out}})$ be a norm bound, $B_R = B_R(\lambda, \ell_{\mathsf{out}})$ be a rounding threshold, and $B_G = B_G(\lambda, \ell_{\mathsf{out}})$ be a truncation bound for the discrete Gaussian error distribution. We implicitly assume that $0 < B_M, B_R, B_G < q$.
- Let $\ell_{\mathsf{pp}} = \ell_{\mathsf{out}} \cdot m \cdot (\lceil \log q \rceil + \lambda + t)$ for $t = t(\lambda, \ell_{\mathsf{out}}) \geq 0$ be the length of the public coins.
- Define the branch set to be

$$\mathfrak{B} \coloneqq \left\{(i_1, \ldots, i_N, z_1, \ldots, z_N) : i_1, \ldots, i_N \in [k], z_1, \ldots, z_N \in \mathbb{Z}_q\right\}.$$

- Define $H \colon \mathfrak{B} \to \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$ for any $\mathsf{br} = (i_1, \ldots, i_N, z_1, \ldots, z_N) \in \mathfrak{B}$ as

$$H((i_1, \ldots, i_N, z_1, \ldots, z_N)) \coloneqq \mathbf{I}_{\ell_{\mathsf{out}}} \otimes \sum_{j \in [N]} z_j \boldsymbol{\eta}_{(j-1)\cdot k + i_j}^\intercal \tag{6.5}$$

  where $\boldsymbol{\eta}_i \in \mathbb{Z}_q^{k \cdot N}$ denotes the i^{th} standard basis vector. We will often write $\mathbf{H}_{\mathsf{br}}$ for $H(\mathsf{br})$.

We construct a branch-constricting generator $\Pi_{\mathsf{BCG}} = (\mathsf{Setup}, \mathsf{Sample}, \mathsf{Verify}, \mathsf{SampleGuess})$ with branch set $\mathfrak{B}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$: On input the security parameter 1^λ, the output length $1^{\ell_{\mathsf{out}}}$, and a branch $\mathsf{br}^* \in \mathfrak{B}$, sample $\mathbf{S} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{\ell_{\mathsf{out}} \times n}$, $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$, $\mathbf{E} \leftarrow \tilde{D}_{\mathbb{Z},\chi,B_G}^{\ell_{\mathsf{out}} \times m}$ and compute $\mathbf{C} = \mathbf{SA} - \mathbf{H}_{\mathsf{br}^*} + \mathbf{E}$. Recall that $\mathbf{H}_{\mathsf{br}^*} = H(\mathsf{br}^*)$ as defined in Eq. (6.5). Output $\mathsf{pp} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{C})$ and $\mathsf{aux} = \mathbf{S}$.
- $\mathsf{Sample}(\mathsf{pp}, \mathsf{br})$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ (which includes an implicit description of 1^λ and $1^{\ell_{\mathsf{out}}}$) and a branch $\mathsf{br} \in \mathfrak{B}$, let $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp}) \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. Then:
  - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$. Recall that $\mathbf{H}_{\mathsf{br}} = H(\mathsf{br})$ as defined in Eq. (6.5).
  - Initialize an ordered list $L = \emptyset$. For $i \in [\lambda]$:
    - Sample $\mathbf{r}_i \leftarrow D_{\mathbb{Z},\chi}^m$.
    - If $\|\mathbf{r}_i\| > B_M$ then set $\mathbf{y}_i = \perp^{\ell_{\mathsf{out}}}$ and $\mathbf{r}_i = \mathbf{0}^m$. Add $(\mathbf{y}_i, \mathbf{r}_i)$ to L and continue to the next iteration.
    - Compute $\mathbf{w}_i = \mathbf{C}_{\mathsf{br}} \mathbf{r}_i$.
    - Compute $\mathbf{y}_i = \mathsf{RoundB}_{q,B_R}(\mathbf{w}_i)$ and add $(\mathbf{y}_i, \mathbf{r}_i)$ to L. Recall that RoundB_{q,B_R} was defined in Eq. (6.4).
  - Let $(\mathbf{y}^*, \mathbf{r}^*)$ be the first element of L for which there does not exist any index $j \in [\ell_{\mathsf{out}}]$ such that $y_j^* = \perp$. If such a tuple does not exist then output $\mathbf{y} = \mathbf{0}^{\ell_{\mathsf{out}}}$, $\boldsymbol{\pi} = \mathbf{0}^m$. Otherwise output $\mathbf{y} = \mathbf{y}^*$ and $\boldsymbol{\pi} = \mathbf{r}^*$.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}, \mathbf{y}, \boldsymbol{\pi})$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ (which includes an implicit description of 1^λ and $1^{\ell_{\mathsf{out}}}$), a branch $\mathsf{br} \in \mathfrak{B}$, a string $\mathbf{y} \in \{0,1\}^{\ell_{\mathsf{out}}}$, and a proof $\boldsymbol{\pi} = \mathbf{r}$, compute $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$, and $\tilde{\mathbf{y}} = \mathsf{RoundB}_{q,B_R}((\mathbf{C} + \mathbf{H}_{\mathsf{br}}) \cdot \mathbf{r})$. Output 1 if one of the following holds:
  - There exists an index $j \in [\ell_{\mathsf{out}}]$ such that $\tilde{\mathbf{y}}_j = \perp$, $\mathbf{y} = \mathbf{0}^{\ell_{\mathsf{out}}}$, and $\boldsymbol{\pi} = \mathbf{0}^m$.
  - It holds that $\tilde{\mathbf{y}} = \mathbf{y}$ and $\|\mathbf{r}\| \le B_M$.
- $\mathsf{SampleGuess}(\mathsf{aux})$: On input the auxiliary information $\mathsf{aux} = \mathbf{S} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times n}$ sample $\boldsymbol{\alpha} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and output $\mathbf{t}^* = \mathsf{RoundS}(\mathbf{S}\boldsymbol{\alpha})$. Recall that RoundS was defined in Eq. (6.4).

Theorem 6.8 (Completeness)**.** *Construction 6.7 is complete.*

Proof. Take any $\lambda, \ell_{\mathsf{out}} \in \mathbb{N}$ and any branches $\mathsf{br} \in \mathfrak{B}$. Take $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$, and $(\mathbf{y}, \boldsymbol{\pi})$ in the support of $\mathsf{Sample}(\mathsf{pp}, \mathsf{br})$. $\mathsf{Samp}(1^\lambda, q, t, \mathsf{pp}) = \mathbf{C}$ for $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. By construction of Sample, we have two cases:

- **Case 1:** $\boldsymbol{\pi} = \mathbf{0}^m$ and $\mathbf{y} = \mathbf{0}^{\ell_{\mathsf{out}}}$. In this case the verification algorithm computes $\tilde{\mathbf{y}} = \mathsf{RoundB}_{q,B_R}(\mathbf{0}^{\ell_{\mathsf{out}}}) = \perp^{\ell_{\mathsf{out}}}$. This will be accepted by the verification algorithm.
- **Case 2:** $\boldsymbol{\pi} = \mathbf{r} \in \mathbb{Z}_q^m$ such that $\|\mathbf{r}\| \le B_M$ and $\mathbf{y} = \mathsf{RoundB}_{q,B_R}((\mathbf{C} + \mathbf{H}_{\mathsf{br}}) \cdot \mathbf{r})$ such that for all $j \in [\ell_{\mathsf{out}}]$, $y_j \neq \perp$. Since $\tilde{\mathbf{y}}$ is also computed in the same way as $\mathbf{y}$, we have $\tilde{\mathbf{y}} = \mathbf{y}$. Further $\|\mathbf{r}\| \le B_M$ so this too will be accepted by the verification algorithm.

Since the verification algorithm accepts in both cases, completeness holds. □

Theorem 6.9 (Mode Indistinguishability)**.** *Suppose $B_G \ge \sqrt{\lambda + t}\chi$ and the ε-$\mathsf{MLWE}_{n,m,q,\chi}$ assumption holds for some $\varepsilon(\lambda)$. Then Construction 6.7 satisfies ε'-mode indistinguishability where $\varepsilon'(\lambda) = \max(\varepsilon(\lambda), 2^{-t})$.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the mode indistinguishability game. We define the following sequence of hybrid experiments:

- Hyb_1: This is the mode indistinguishability game with bit $b = 0$. Specifically, the game proceeds as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the length $1^{\ell_{\mathsf{out}}}$ and a branch $\mathsf{br}^* \in \mathfrak{B}$.
  2. The challenger samples $\mathsf{pp} \xleftarrow{\mathrm{R}} \{0,1\}^{\ell_{\mathsf{pp}}}$ and gives pp to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

- Hyb_2: Same as Hyb_1, except the challenger changes how it samples the public parameters pp:
  2. The challenger samples $\mathbf{C} \xleftarrow{R} \mathbb{Z}_q^{\ell_{\text{out}} \times m}$ and sets $\mathsf{pp} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{C})$ and gives pp to $\mathcal{A}$.
- Hyb_3: Same as Hyb_2, except the challenger changes how it samples the public parameters pp:
  2. The challenger samples $\mathbf{S} \xleftarrow{R} \mathbb{Z}_q^{\ell_{\text{out}} \times n}$, $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{E} \leftarrow D_{\mathbb{Z},\chi}^{\ell_{\text{out}} \times m}$ and computes $\mathbf{C} = \mathbf{SA} - \mathbf{H}_{\mathsf{br}^*} + \mathbf{E}$. It sets $\mathsf{pp} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{C})$ and gives pp to $\mathcal{A}$.
- Hyb_4: Same as Hyb_3, except the challenger changes how it samples the error $\mathbf{E}$:
  2. The challenger samples $\mathbf{S} \xleftarrow{R} \mathbb{Z}_q^{\ell_{\text{out}} \times n}$, $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{E} \leftarrow \tilde{D}_{\mathbb{Z},\chi,B_G}^{\ell_{\text{out}} \times m}$ and computes $\mathbf{C} = \mathbf{SA} - \mathbf{H}_{\mathsf{br}^*} + \mathbf{E}$. It sets $\mathsf{pp} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{C})$ and gives pp to $\mathcal{A}$.

  This is the mode indistinguishability game with bit $b = 1$.

Let $\mathsf{Hyb}_i(\mathcal{A})$ be the random variable denoting the output of an execution of experiment Hyb_i with adversary $\mathcal{A}$. We now analyze each adjacent pair of hybrid experiments.

Claim 6.10. *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = 2^{-t} \cdot \mathsf{negl}(\lambda)$.

Proof. This proof is similar to that of Claim 5.9. The only difference between the hybrids is in the way the public parameter pp is sampled. In Hyb_1, the public parameters pp are sampled uniformly at random from $\{0,1\}^{\ell_{\mathsf{pp}}}$, while in Hyb_2, they are sampled as $\mathsf{pp} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{C})$ for uniformly random $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\text{out}} \times m}$. Since $\mathcal{A}$ is efficient, the matrix dimensions ℓ_{out} and $m = k \cdot N \cdot \ell_{\text{out}}$ are polynomials in λ. Then by the statistical indistinguishability property of Definition 6.1, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the statistical distance between the following distributions is at most $2^{-t} \cdot \mathsf{negl}(\lambda)$

$$\left\{\mathsf{pp} : \mathsf{pp} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{pp}}}\right\} \quad \text{and} \quad \left\{\mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{C}) : \mathbf{C} \xleftarrow{R} \mathbb{Z}_q^{\ell_{\text{out}} \times m}\right\}.$$

These distributions exactly correspond to the distribution of pp in Hyb_1 and Hyb_2 respectively, so the claim follows. □

Claim 6.11. *Suppose the* ε-$\mathsf{MLWE}_{n,m,q,\chi}$ *assumption holds. Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \varepsilon(\lambda) \cdot \mathsf{negl}(\lambda)$.

Proof. The proof is similar to that of Claim 5.10. Suppose $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| > \varepsilon(\lambda) \cdot \delta(\lambda)$ for some non-negligible δ. Without loss of generality, suppose that for each security parameter λ, algorithm $\mathcal{A}$ chooses a fixed value of $\ell_{\text{out}} = \ell_{\text{out}}(\lambda)$. We now use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ that breaks the ε-$\mathsf{MLWE}_{n,m,q,\chi}$ assumption with dimension ℓ_{out} with the same advantage. Algorithm $\mathcal{B}$ works as follows:

1. On input the security parameter 1^λ and the challenge $(\mathbf{A}, \mathbf{C})$, algorithm $\mathcal{B}$ invokes algorithm $\mathcal{A}$ on the same security parameter.
2. Algorithm $\mathcal{A}$ outputs the output length $1^{\ell_{\text{out}}}$ and a branch $\mathsf{br}^* \in \mathfrak{B}$.
3. Algorithm $\mathcal{B}$ sends $\mathsf{pp} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{C} - \mathbf{H}_{\mathsf{br}^*})$ to $\mathcal{A}$.
4. At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit b' which $\mathcal{B}$ also outputs.

We argue that depending on the distribution of the challenge, algorithm $\mathcal{B}$ either perfectly simulates an execution of Hyb_2 or of Hyb_3 for $\mathcal{A}$:

- Suppose the challenger sends $(\mathbf{A}, \mathbf{C})$ for uniformly random $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\text{out}} \times m}$. Algorithm $\mathcal{B}$ computes $\mathbf{C} - \mathbf{H}_{\mathsf{br}^*}$ and sends $\mathsf{pp} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{C} - \mathbf{H}_{\mathsf{br}^*})$ to $\mathcal{A}$. Since $\mathbf{C}$ is uniformly random, the distribution of $\mathbf{C} - \mathbf{H}_{\mathsf{br}^*}$ is uniform over $\mathbb{Z}_q^{\ell_{\text{out}} \times m}$ and independent of br^*. This coincides with the distribution in Hyb_2, so $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

- Suppose the challenger sends $(\mathbf{A}, \mathbf{C})$ to $\mathcal{B}$ where $\mathbf{C} = \mathbf{SA} + \mathbf{E}$. Algorithm $\mathcal{B}$ computes $\mathbf{C} - \mathbf{H}_{\mathsf{br}^*} = \mathbf{SA} + \mathbf{E} - \mathbf{H}_{\mathsf{br}^*}$ and sends $\mathsf{pp} \leftarrow \mathsf{Samp}^{-1}(1^\lambda, q, t, \mathbf{SA} - \mathbf{H}_{\mathsf{br}^*} + \mathbf{E})$ to $\mathcal{A}$. This coincides with the distribution in Hyb_3, so $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ wins with the same advantage $\varepsilon(\lambda) \cdot \delta(\lambda)$, breaking the ε-$\mathsf{MLWE}_{n,m,q,\chi}$ assumption. □

Claim 6.12. *Suppose $B_G \geq \sqrt{\lambda + t}\,\chi$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| = 2^{-t} \cdot \mathsf{negl}(\lambda)$.*

Proof. The only difference between the hybrids is in the distribution from which $\mathbf{E}$ is sampled. Since $\mathcal{A}$ is efficient, the matrix dimensions ℓ_{out} and $m = k \cdot N \cdot \ell_{\text{out}}$ are polynomials in λ. By a Gaussian tail bound (Lemma 6.2) and a hybrid argument (over the components of $\mathbf{E}$), the statistical distance between the distributions of $\mathbf{E}$ in the two hybrids is at most $\ell_{\text{out}} \cdot m \cdot 2^{-(t+\lambda)}$, which is $2^{-t} \cdot \mathsf{negl}(\lambda)$. Thus the claim follows. □

Proof of Theorem 6.9. By Claims 6.10 to 6.12, the difference between each pair of adjacent hybrids can be bounded by $\max(2^{-t}, \varepsilon(\lambda)) \cdot \mathsf{negl}(\lambda)$. The claim now follows by a standard hybrid argument. □

Theorem 6.13 (Most-Branches Target Randomness)**.** *Suppose $B_M \geq \sqrt{\lambda}\chi$, $\chi \geq \max(4m \log m, m \log \ell_{out})$, $m \geq 3\ell_{out} \lceil \log q \rceil$, $k \geq 4\ell_{out} \lceil \log q \rceil$, and $q > 2\ell_{out}(4B_R + 3)$. Then for all polynomials $\ell_{out} = \ell_{out}(\lambda)$, Construction 6.7 satisfies most-branches target randomness.*

Proof. We first prove a few helper lemmas that show that, with high probability over the choice of branches $\mathsf{br} \in \mathfrak{B}$, the matrix $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$ has a gadget trapdoor for *any* $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\text{out}} \times m}$.

Lemma 6.14. *Take any matrix $\mathbf{C} \in \mathbb{Z}_q^{\ell_{out} \times k \cdot N}$ where $k \geq 4\ell_{out} \lceil \log q \rceil$. Then there exists a set of vectors $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \{-1, 0, 1\}^k$ such that for all $i \in [N]$:*

- *It holds that $\mathbf{C}(\mathbf{h}_i \otimes \tilde{\boldsymbol{\eta}}_i) = \mathbf{0}^{\ell_{out}}$, where $\tilde{\boldsymbol{\eta}}_i \in \mathbb{Z}_q^N$ is the i^{th} standard basis vector.*
- *The fraction of indices where $\mathbf{h}_i$ is non-zero is at least $1/4$.*

Proof. Suppose $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\text{out}} \times k \cdot N}$ where $k \geq 4\ell_{\text{out}} \log q$. We split $\mathbf{C}$ into N "chunks" of k columns each: $\mathbf{C} = [\mathbf{C}_1 \mid \cdots \mid \mathbf{C}_N]$ where for all $i \in [N]$, $\mathbf{C}_i \in \mathbb{Z}_q^{\ell_{\text{out}} \times k}$. Then we need to show that for all $i \in [N]$, there exists a vector $\mathbf{h}_i$ such that

- $\mathbf{C}(\mathbf{h}_i \otimes \tilde{\boldsymbol{\eta}}_i) = \mathbf{C}_i \mathbf{h}_i = \mathbf{0}^{\ell_{\text{out}}}$; and
- the fraction of indices where $\mathbf{h}_i$ is non-zero is at least $1/4$.

Consider the following (inefficient) algorithm $\mathsf{FindVector}(\mathbf{C}_i)$ for finding such a vector:

On input $\mathbf{C}_i \in \mathbb{Z}_q^{\ell_{\text{out}} \times k}$:

1. Initialize $S = \varnothing$ and $\mathbf{h}_i = \mathbf{0}^k$.
2. While $|S| < k/4$:
   (a) Let $\mathbf{h}'$ be the lexicographically first vector in $\{-1, 0, 1\}^k$ such that
      - It holds that $\mathbf{h}'$ is not identically $\mathbf{0}^k$.
      - For all indices $z \in S$, $h'_z = 0$.
      - It holds that $\mathbf{C}_i \mathbf{h}' = \mathbf{0}^{\ell_{\text{out}}}$.

   (b) Update $\mathbf{h}_i$ to $\mathbf{h}_i + \mathbf{h}'$
   (c) Update $S = \{z \in [k] : h_{i,z} \neq 0\}$.
3. Output $\mathbf{h}_i$.

We will now show that for all $\mathbf{C}_i \in \mathbb{Z}_q^{\ell_{\text{out}} \times k}$, algorithm $\mathsf{FindVector}(\mathbf{C}_i)$ finds a vector $\mathbf{h}_i$ satisfying the required conditions.

- Consider any iteration of the algorithm. By the initialization step in Step 1 and the update rule in Step 2c, the set S always contains exactly the indices $z \in [k]$ where $h_{i,z} \neq 0$. Furthermore, by the continuation condition of the while loop, we are guaranteed that $|S| < k/4$ on every iteration.
- We first show that on every iteration, there exists $\mathbf{y}_0 \neq \mathbf{y}_1 \in \{0,1\}^k$ such that
  - It holds that $\mathbf{C}_i\mathbf{y}_0 = \mathbf{C}_i\mathbf{y}_1$.
  - For all $z \in S$, we have that $y_{0,z} = y_{1,z} = 0$.

  To see this, we first count the number of possible outputs formed by multiplying $\mathbf{C}_i$ by a vector $\mathbf{y}$. Consider the mapping $\mathbf{y} \mapsto \mathbf{C}_i\mathbf{y}$. Since $\mathbf{C}_i \in \mathbb{Z}_q^{\ell_{\text{out}} \times k}$, the image of this mapping has size at most $q^{\ell_{\text{out}}}$. Next we count the number of possible bit vectors that satisfy our constraints. There are 2^k possible bit vectors of length k. If we restrict ourselves to bit vectors $\mathbf{y}$ such that $y_z = 0$ for all $z \in S$, then there are $2^{k-|S|}$ such vectors. Since $|S| < k/4$, the number of inputs that satisfy our conditions is $2^{k-|S|} > 2^{3k/4}$. Since $k \geq 4\ell_{\text{out}}\lceil \log q \rceil$, we have $2^{3k/4} \geq q^{3\ell_{\text{out}}} > q^{\ell_{\text{out}}}$. On the other hand, the number of possible outputs is at most $q^{\ell_{\text{out}}}$. Hence $2^{k-|S|} > q^{\ell_{\text{out}}}$. By the pigeonhole principle, there must exist two distinct vectors $\mathbf{y}_0, \mathbf{y}_1$ satisfying the above conditions that produce the same output under the mapping.
- Consider the vector $\mathbf{y}_1 - \mathbf{y}_0 \in \{-1, 0, 1\}^k$. Since $\mathbf{y}_0 \neq \mathbf{y}_1$, the vector $\mathbf{y}_1 - \mathbf{y}_0$ has at least one nonzero entry in $\{-1, 1\}$. Moreover, $y_{1,z} - y_{0,z} = 0$ for all $z \in S$ by construction. Finally, since $\mathbf{C}_i\mathbf{y}_0 = \mathbf{C}_i\mathbf{y}_1$, we have $\mathbf{C}_i(\mathbf{y}_1 - \mathbf{y}_0) = 0$. Hence, the vector $\mathbf{y}_1 - \mathbf{y}_0$ satisfies all of the conditions in Step 2a of the FindVector algorithm. Since at least one vector exists, Step 2a of the FindVector algorithm will always find one (say, by exhaustive search).
- Finally, we show that the loop must terminate. Recall that the vector $\mathbf{h}'$ chosen in Step 2a satisfies the following properties:
  - It has at least one index where it is non-zero.
  - It is zero at every index where $\mathbf{h}_i$ is non-zero.

  Thus, when $\mathbf{h}'$ is added to $\mathbf{h}_i$ in Step 2b, it introduces at least one new non-zero coordinate without affecting the coordinates that are already non-zero. Then the set S gains at least one new index in each iteration, so $|S|$ strictly increases in every iteration. Therefore the loop must terminate, and upon termination we have $|S| \geq k/4$. Thus, $\mathbf{h}_i$ has at least $k/4$ non-zero coordinates.

Thus, we run the algorithm for each $i \in [N]$ to find the vectors $\mathbf{h}_1, \ldots, \mathbf{h}_N$ satisfying the conditions required by Lemma 6.14. □

Lemma 6.15. *Let $\lambda \in \mathbb{N}$ be a security parameter, and let $\ell_{out} = \ell_{out}(\lambda)$ and $q = q(\lambda, \ell_{out})$ be polynomials. For a matrix $\mathbf{C} \in \mathbb{Z}_q^{\ell_{out} \times k \cdot N}$ where $k \geq 4\ell_{out}\lceil \log q \rceil$ and $N = 8\lambda$, let $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \{-1, 0, 1\}^k$ be the vectors satisfying the conditions of Lemma 6.14. Define*

$$S \coloneqq \left\{ \sum_{i \in [N]} b_i(\mathbf{h}_i \otimes \tilde{\boldsymbol{\eta}}_i) : b_1, \ldots, b_N \in \{0,1\} \right\} \subseteq \{-1, 0, 1\}^{k \cdot N} \tag{6.6}$$

where $\tilde{\boldsymbol{\eta}}_i \in \mathbb{Z}_q^N$ is the i^{th} standard basis vector. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, for all $u \in \mathbb{Z}_q$, and for all but a $\mathsf{negl}(\lambda)$ fraction of branches $\mathsf{br} = (i_1, \ldots, i_N, z_1, \ldots, z_N) \in \mathfrak{B}$, there exists a vector $\mathbf{d} \in S$ such that

$$\sum_{j \in [N]} z_j d_{(j-1)k + i_j} = u.$$

Proof. Fix a security parameter λ, and let $\ell_{\text{out}} = \ell_{\text{out}}(\lambda)$ and $q = q(\lambda, \ell_{\text{out}})$ be polynomials. Take any matrix $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\text{out}} \times k \cdot N}$ where $k \geq 4\ell_{\text{out}} \log q$ and $N = 8\lambda$, and let $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \{-1, 0, 1\}^k$ be the corresponding vectors satisfying the conditions of Lemma 6.14. Let S be as defined in Eq. (6.6). Then, we have the following:

- By the definition of S, any vector $\mathbf{d} = [d_1, \ldots, d_{k \cdot N}]^\intercal \in S$ will have entries in $\{-1, 0, 1\}$ and have the form $\mathbf{d} = \left[b_1 \mathbf{h}_1^\intercal \mid \cdots \mid b_N \mathbf{h}_N^\intercal\right]^\intercal$ where for $i \in [N]$, $\mathbf{h}_i \in \{-1, 0, 1\}^k$. For notational convenience, define $a_j \coloneqq h_{j,i_j}$.
- For $\mathsf{br} = (i_1, \ldots, i_N, z_1, \ldots, z_N) \in \mathfrak{B}$, define $F_{\mathsf{br}} \colon \{0,1\}^N \to \mathbb{Z}_q$ as

$$F_{\mathsf{br}}(b_1, \ldots, b_N) \coloneqq \sum_{j \in [N]} b_j z_j a_j$$

  By definition

$$\sum_{j \in [N]} z_j d_{(j-1)k + i_j} = \sum_{j \in [N]} b_j z_j a_j = F_{\mathsf{br}}(b_1, \ldots, b_N).$$

  The claim now reduces to showing that for all but a negligible fraction of branches $\mathsf{br} \in \mathfrak{B}$, the function F_{br} is surjective.
- We start by showing that for all $j \in [N]$, with $1 - \mathsf{negl}(\lambda)$ probability over the choice of $\mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}$, at least an $(N/8)$-fraction of the a_j's are non-zero. Let X_j be the indicator random variable for the event $a_j \neq 0$. Since at least $1/4$ fraction of indices in $\mathbf{h}_j$ are non-zero and each index $i_j \xleftarrow{\mathrm{R}} [k]$ is sampled uniformly at random, this means $\Pr[X_j = 1] \geq 1/4$. Then by a Chernoff bound (Lemma 2.1), we obtain:

$$\Pr\left[\sum_{j \in [N]} X_j < N/8 : \mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}\right] = e^{-\Omega(N)}.$$

  Since $N = 8\lambda$, with probability $1 - e^{-\Omega(\lambda)} = 1 - \mathsf{negl}(\lambda)$ over the choice of br, at least λ of the a_j's are non-zero. Let $\mathfrak{B}' \subseteq \mathfrak{B}$ be the set of branches where this condition holds (i.e., where at least λ of the a_j's are non-zero).
- For $\mathsf{br} \in \mathfrak{B}'$, let $I_{\mathsf{br}} = \{j_1, \ldots, j_\lambda\}$ denote the first λ indices where $a_j \neq 0$. Define the function $F'_{\mathsf{br}} \colon \{0,1\}^\lambda \to \mathbb{Z}_q$ as

$$F'_{\mathsf{br}}(b'_1, \ldots, b'_\lambda) \coloneqq \sum_{k \in [\lambda]} b'_k z_{j_k} a_{j_k}.$$

  We note that F'_{br} is a restricted version of the function F_{br} and its image is a subset of the image of F_{br}. Specifically, given any vector $(b'_1, \ldots, b'_\lambda) \in \{0,1\}^\lambda$ as input for F'_{br}, we can translate it to an input $(b_1, \ldots, b_N) \in \{0,1\}^N$ for F_{br} as follows:
  - For each $k \in [\lambda]$, set $b_{j_k} = b'_k$.
  - For all other indices $j \notin I_{\mathsf{br}}$, set $b_j = 0$.

  Evaluating F_{br} on the transformed input yields:

$$F_{\mathsf{br}}(b_1, \ldots, b_N) = \sum_{j \in [N]} b_j z_j a_j = \sum_{k \in [\lambda]} b_{j_k} z_{j_k} a_{j_k} + \sum_{j \notin I_{\mathsf{br}}} b_j z_j a_j = F'_{\mathsf{br}}(b'_1, \ldots, b'_\lambda).$$

  Thus, to show that F_{br} is surjective, it suffices to show that F'_{br} is surjective. We will now proceed to show this.
- Next, we show that the family $\left\{F'_{\mathsf{br}}\right\}_{\mathsf{br} \in \mathfrak{B}'}$ forms a universal hash function family. Take any two distinct inputs $\mathbf{x}, \mathbf{y} \in \{0,1\}^\lambda$. Since $\mathbf{x} \neq \mathbf{y}$, there exists some index $\mathsf{ind} \in [\lambda]$ such that $x_{\mathsf{ind}} \neq y_{\mathsf{ind}}$. We evaluate the collision probability:

$$\Pr[F'_{\mathsf{br}}(\mathbf{x}) = F'_{\mathsf{br}}(\mathbf{y}) : \mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}'] = \Pr\left[\sum_{k \in [\lambda]} (x_k - y_k) z_{j_k} a_{j_k} = 0 : \mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}'\right].$$

  Isolating the term for ind, this is equivalent to:

$$\Pr\left[z_{j_{\mathsf{ind}}} (x_{\mathsf{ind}} - y_{\mathsf{ind}}) a_{j_{\mathsf{ind}}} = -\sum_{k \neq \mathsf{ind}} (x_k - y_k) z_{j_k} a_{j_k} : \mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}'\right].$$

Since $\mathbf{x}, \mathbf{y} \in \{0,1\}^\lambda$ and $x_{\mathsf{ind}} \neq y_{\mathsf{ind}}$, we have $(x_{\mathsf{ind}} - y_{\mathsf{ind}}) \in \{-1, 1\}$. Furthermore, by the definition of I_{br}, $a_{j_{\mathsf{ind}}} \neq 0$, meaning $a_{j_{\mathsf{ind}}} \in \{-1, 1\}$. Thus, the product $(x_{\mathsf{ind}} - y_{\mathsf{ind}})a_{j_{\mathsf{ind}}} \in \{-1, 1\}$, which is invertible in $\mathbb{Z}_q$. For any fixed choice of z_{j_k} for $k \neq \mathsf{ind}$, there is exactly one value of $z_{j_{\mathsf{ind}}} \in \mathbb{Z}_q$ that satisfies the equation. Because $z_{j_{\mathsf{ind}}}$ is drawn uniformly from $\mathbb{Z}_q$ and is independent of the other variables, the equation holds with probability exactly $1/q$ when q is prime (i.e., $\mathbb{Z}_q$ is a field).

- By the leftover hash lemma (Corollary 2.6), we conclude that there exists a negligible function $\mathsf{negl}'(\cdot)$ such that the statistical distance between the following distributions is $\mathsf{negl}'(\lambda)$:

$$\left\{(\mathsf{br}, F'_{\mathsf{br}}(b'_1, \ldots b'_\lambda)) : \mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}', b'_1, \ldots b'_\lambda \xleftarrow{\mathrm{R}} \{0,1\}\right\} \quad \text{and} \quad \left\{(\mathsf{br}, u) : \mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}', u \xleftarrow{\mathrm{R}} \mathbb{Z}_q\right\} \tag{6.7}$$

- We now argue that with all but negligible probability over the choice of branches $\mathsf{br} \in \mathfrak{B}'$, for every $u \in \mathbb{Z}_q$ there exist $b'_1, \ldots, b'_\lambda \in \{0,1\}$ such that $F'_{\mathsf{br}}(b'_1, \ldots, b'_\lambda) = u$. Suppose towards contradiction that with non-negligible probability ε, the sampled branch $\mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}$ produces a function F'_{br} that is not surjective. Let $\mathfrak{B}'' \subseteq \mathfrak{B}'$ be the set of such "deficient" branches, meaning for every $\mathsf{br} \in \mathfrak{B}''$, there exists a missing target $u_{\mathsf{br}} \in \mathbb{Z}_q$ such that $F'_{\mathsf{br}}(b'_1, \ldots, b'_\lambda) \neq u_{\mathsf{br}}$ for all $b'_1, \ldots, b'_\lambda \in \{0,1\}$.

  Consider the event where a branch $\mathsf{br} \in \mathfrak{B}''$ is sampled, followed by sampling $u = u_{\mathsf{br}}$:

  - In the first distribution of Eq. (6.7) (where u is generated by evaluating F'_{br} on uniform bits), the probability of producing $(\mathsf{br}, u_{\mathsf{br}})$ is exactly 0.
  - In the second distribution of Eq. (6.7) (where u is drawn uniformly from $\mathbb{Z}_q$), the probability of producing $(\mathsf{br}, u_{\mathsf{br}})$ is ε/q.

  Since q is polynomial in λ and ε is non-negligible, the quantity ε/q is non-negligible. This implies the statistical distance between the two distributions is non-negligible, contradicting the statistical uniformity established in Eq. (6.7). Thus, F'_{br} is surjective for all but a negligible fraction of branches, and the claim follows. □

Corollary 6.16. *Let $\lambda \in \mathbb{N}$ be a security parameter, and let $\ell_{out} = \ell_{out}(\lambda)$ and $q = q(\lambda, \ell_{out})$ be polynomials. Let $N = 8\lambda$, $k \geq 4\ell_{out} \log q$, and $m = k \cdot N \cdot \ell_{out}$. For any matrix $\mathbf{C} \in \mathbb{Z}_q^{\ell_{out} \times m}$, for all but a $\mathsf{negl}(\lambda)$ fraction of branches $\mathsf{br} \in \mathfrak{B}$, there exists a matrix $\mathbf{R}_{\mathbf{C},\mathsf{br}} \in \{-1,0,1\}^{m \times m'}$ such that*

$$\mathbf{C}_{\mathsf{br}} \mathbf{R}_{\mathbf{C},\mathsf{br}} = \mathbf{G}$$

where $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$ and $\mathbf{H}_{\mathsf{br}}$ was defined in Eq. (6.5), $\mathbf{G} \in \mathbb{Z}_q^{\ell_{out} \times m'}$ is the gadget matrix, and $m' = \ell_{out} \lceil \log q \rceil$.

Proof. Fix a security parameter λ, and let $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$ and $q = q(\lambda, \ell_{\mathsf{out}})$ be polynomials. Take any $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$ where $k \geq 4\ell_{\mathsf{out}} \lceil \log q \rceil$.

- Decompose $\mathbf{C}$ into ℓ_{out} "chunks" of $k \cdot N$ columns each, that is, $\mathbf{C} = [\mathbf{C}_1 \mid \cdots \mid \mathbf{C}_{\ell_{\mathsf{out}}}]$ where for all $i \in [\ell_{\mathsf{out}}]$, $\mathbf{C}_i \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times k \cdot N}$. For each $\mathbf{C}_i$, let $\mathbf{h}_{i,1}, \ldots, \mathbf{h}_{i,N} \in \{-1,0,1\}^k$ be the vectors satisfying the conditions of Lemma 6.14.
- For $i \in [\ell_{\mathsf{out}}]$, define the set

$$S_i \coloneqq \left\{ \sum_{j \in [N]} b_j (\mathbf{h}_{i,j} \otimes \tilde{\boldsymbol{\eta}}_j) : b_1, \ldots, b_N \in \{0,1\} \right\}$$

  where $\tilde{\boldsymbol{\eta}}_i \in \mathbb{Z}_q^N$ is the i^{th} standard basis vector. By definition of $\mathbf{h}_{i,1}, \ldots, \mathbf{h}_{i,N}$ (see Lemma 6.14), for every $\mathbf{d}_i \in S_i$, we have $\mathbf{C}_i \mathbf{d}_i = \mathbf{0}^{\ell_{\mathsf{out}}}$, since for all $j \in [N]$, $\mathbf{C}_i(\mathbf{h}_{i,j} \otimes \tilde{\boldsymbol{\eta}}_j) = \mathbf{0}^{\ell_{\mathsf{out}}}$.
- Consider the matrix $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$. We will now show that for all but a negligible fraction of branches $\mathsf{br} \in \mathfrak{B}$ and for every $\mathbf{u} \in \mathbb{Z}_q^{\ell_{\mathsf{out}}}$, there exists a vector $\mathbf{d} \in \{-1,0,1\}^m$ such that $\mathbf{C}_{\mathsf{br}} \mathbf{d} = \mathbf{u}$. Let $\mathbf{v}_{\mathsf{br}}^\intercal \coloneqq \sum_{j \in [N]} z_j \boldsymbol{\eta}_{(j-1)k+i_j}^\intercal \in \mathbb{Z}_q^{k \cdot N}$. We can express $\mathbf{H}_{\mathsf{br}}$ as $\mathbf{I}_{\ell_{\mathsf{out}}} \otimes \mathbf{v}_{\mathsf{br}}^\intercal$. Let $\mathbf{d} = [\mathbf{d}_1^\intercal \mid \cdots \mid \mathbf{d}_{\ell_{\mathsf{out}}}^\intercal]^\intercal$ where for $i \in [\ell_{\mathsf{out}}]$, $\mathbf{d}_i \in \{-1,0,1\}^{k \cdot N}$. As $\mathbf{C}\mathbf{d} = \sum_{i \in [\ell_{\mathsf{out}}]} \mathbf{C}_i \mathbf{d}_i = \mathbf{0}^{\ell_{\mathsf{out}}}$, note that:

$$\mathbf{C}_{\mathsf{br}} \mathbf{d} = (\mathbf{C} + \mathbf{H}_{\mathsf{br}})\mathbf{d} = \mathbf{H}_{\mathsf{br}} \mathbf{d}.$$

- Since $\mathbf{H}_{\mathsf{br}} = \mathbf{I}_{\ell_{\mathsf{out}}} \otimes \mathbf{v}_{\mathsf{br}}^{\intercal}$, the i^{th} coordinate of the output $\mathbf{H}_{\mathsf{br}}\mathbf{d} = (\mathbf{I}_{\ell_{\mathsf{out}}} \otimes \mathbf{v}_{\mathsf{br}}^{\intercal})\mathbf{d}$ is precisely

$$\mathbf{v}_{\mathsf{br}}^{\intercal}\mathbf{d}_i = \sum_{j\in[N]} z_j d_{i,(j-1)k+i_j}.$$

- By Lemma 6.15, for all but a negligible fraction of branches $\mathsf{br} \in \mathfrak{B}$, it holds that for all $u_i \in \mathbb{Z}_q$, there exists a $\mathbf{d}_i \in \{-1,0,1\}^{k\cdot N}$ such that $\mathbf{v}_{\mathsf{br}}^{\intercal}\mathbf{d}_i = u_i$. That is, for any target vector $\mathbf{u} = [u_1,\ldots,u_{\ell_{\mathsf{out}}}]^{\intercal} \in \mathbb{Z}_q^{\ell_{\mathsf{out}}}$, there exist vectors $\mathbf{d}_1 \in S_1,\ldots,\mathbf{d}_{\ell_{\mathsf{out}}} \in S_{\ell_{\mathsf{out}}}$ such that

$$\forall i \in [\ell_{\mathsf{out}}] : \quad \sum_{j\in[N]} z_j d_{i,(j-1)k+i_j} = u_i. \tag{6.8}$$

  This demonstrates that for any vector $\mathbf{u} \in \mathbb{Z}_q^{\ell_{\mathsf{out}}}$, there exists a vector $\mathbf{d} \in \{-1,0,1\}^m$ such that $\mathbf{H}_{\mathsf{br}}\mathbf{d} = \mathbf{u}$, and correspondingly, that $\mathbf{C}_{\mathsf{br}}\mathbf{d} = \mathbf{u}$.

- By taking the target vectors $\mathbf{u}$ to be the columns of the gadget matrix, we conclude that for all but a negligible fraction of branches $\mathsf{br} \in \mathfrak{B}$, there exists a matrix $\mathbf{R}_{\mathsf{C},\mathsf{br}} \in \{-1,0,1\}^{m\times m'}$ such that $\mathbf{C}_{\mathsf{br}}\mathbf{R}_{\mathsf{C},\mathsf{br}} = \mathbf{G}$. □

Now returning to the proof of Theorem 6.13, we define the Preprocess and Target algorithms as follows:

- $\mathsf{Preprocess}(\mathsf{pp},\mathsf{br})$: On input the public parameters $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ and branch $\mathsf{br} \in \mathfrak{B}$:
  - Parse $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ for $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}}\times m}$.
  - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}} \in \mathbb{Z}_q^{\ell_{\mathsf{out}}\times m}$.
  - Compute a gadget trapdoor $\mathbf{R}_{\mathsf{C},\mathsf{br}} \in \{-1,0,1\}^{m\times m'}$ for $\mathbf{C}_{\mathsf{br}}$ so that $\mathbf{C}_{\mathsf{br}}\mathbf{R}_{\mathsf{C},\mathsf{br}} = \mathbf{G}$. If such a trapdoor does not exist then set $\mathbf{R}_{\mathsf{C},\mathsf{br}} = \perp$.

  Output $\mathsf{td} = (\mathbf{C}_{\mathsf{br}}, \mathbf{R}_{\mathsf{C},\mathsf{br}}, \mathsf{br})$.

- $\mathsf{Target}(\mathsf{td},\mathbf{y})$: Given a trapdoor $\mathsf{td} = (\mathbf{C}_{\mathsf{br}}, \mathbf{R}_{\mathsf{C},\mathsf{br}}, \mathsf{br})$ and a target string $\mathbf{y} \in \{0,1\}^{\ell_{\mathsf{out}}}$:
  - If $\mathbf{R}_{\mathsf{C},\mathsf{br}} = \perp$ then abort and output $\boldsymbol{\pi} = \mathbf{0}^m$.
  - Compute $\mathbf{w} \leftarrow \mathsf{RoundB}^{-1}_{q,B_R}(\mathbf{y})$, where $\mathsf{RoundB}^{-1}_{q,B_R}$ is the reverse-sampling algorithm defined in Fig. 1.
  - Sample $\mathbf{r} \leftarrow \mathsf{SamplePre}(\mathbf{C}_{\mathsf{br}}, \mathbf{R}_{\mathsf{C},\mathsf{br}}, \mathbf{w}, \chi)$, where SamplePre is the preimage-sampling algorithm from Theorem 6.3.

  Output $\boldsymbol{\pi} = \mathbf{r}$.

We now show that (Preprocess, Target) satisfy the required properties.

Efficiency. The size of the trapdoor $\mathsf{td} = (\mathbf{C}_{\mathsf{br}}, \mathbf{R}_{\mathsf{C},\mathsf{br}}, \mathsf{br})$ is $O(\ell_{\mathsf{out}}\cdot m + m\cdot m' + N\log(qk)) = O(8\lambda\ell_{\mathsf{out}}^2 k + \lambda\ell_{\mathsf{out}}^2 k(\lceil\log q\rceil) + 8\lambda\log(qk))$ which is $\mathsf{poly}(\lambda,\ell_{\mathsf{out}})$ for all polynomials $k = k(\lambda,\ell_{\mathsf{out}})$ and $\log q = O(\lambda)$. Moreover, all steps of the Target algorithm run in $\mathsf{poly}(\lambda,\ell_{\mathsf{out}})$ time, so it is efficient.

Most-branches target randomness. Before proving this property, we set up some useful notation. Let $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$ be a set of public parameters and let $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp}) \in \mathbb{Z}_q^{\ell_{\mathsf{out}}\times m}$. Define

$$\mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}} \coloneqq \left\{\mathsf{br} : \nexists\, \mathbf{R}_{\mathsf{C},\mathsf{br}} \in \{-1,0,1\}^{m\times m'} \text{ such that } \mathbf{C}_{\mathsf{br}}\mathbf{R}_{\mathsf{C},\mathsf{br}} = \mathbf{G} \text{ where } \mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}\right\}. \tag{6.9}$$

Suppose there exists a polynomial $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$ and a possibly unbounded adversary $\mathcal{A}$ that can break the target randomness of Construction 6.7. We define a sequence of hybrids as follows:

- Hyb_1: This is the target randomness game where $b = 0$.

1. On input the security parameter 1^λ and the output length $1^{\ell_{\mathsf{out}}}$, the adversary $\mathcal{A}$ sends pp to the challenger.
2. The challenger computes $\ell_{\mathsf{pp}} = \ell_{\mathsf{pp}}(\lambda, \ell_{\mathsf{out}})$ and checks that $\mathsf{pp} \in \{0,1\}^{\ell_{\mathsf{pp}}}$. Otherwise, it halts with output 0.
3. The challenger parses $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ where $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. Then it works as follows:
   - Sample $\mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}$
   - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$.
   - Initialize an ordered list $L = \emptyset$. For $i \in [\lambda]$:
     * Sample $\mathbf{r}_i \leftarrow D_{\mathbb{Z},\chi}^m$.
     * If $\|\mathbf{r}_i\| > B_M$ then set $\mathbf{y}_i = \perp^{\ell_{\mathsf{out}}}$ and $\mathbf{r}_i = \mathbf{0}^m$. Add $(\mathbf{y}_i, \mathbf{r}_i)$ to L and continue to the next iteration.
     * Compute $\mathbf{w}_i = \mathbf{C}_{\mathsf{br}} \mathbf{r}_i$.
     * Compute $\mathbf{y}_i = \mathsf{RoundB}_{q,B_R}(\mathbf{w}_i)$ and add $(\mathbf{y}_i, \mathbf{r}_i)$ to L.
   - Let $(\mathbf{y}^*, \mathbf{r}^*)$ be the first element of L for which there does not exist any index $j \in [\ell_{\mathsf{out}}]$ such that $y_j^* = \perp$. If such a tuple does not exist then set $\mathbf{y} = \mathbf{0}^{\ell_{\mathsf{out}}}$, $\boldsymbol{\pi} = \mathbf{0}^m$. Otherwise set $\mathbf{y} = \mathbf{y}^*$ and $\boldsymbol{\pi} = \mathbf{r}^*$.

   Finally send $\mathsf{br}, \mathbf{y}$ and $\boldsymbol{\pi}$ to the adversary $\mathcal{A}$.
4. Finally, the adversary outputs a guess $b' \in \{0,1\}$ which is the output of the experiment.

- Hyb_2: Same as Hyb_1 except we change the distribution of $\mathbf{r}_i$ in Step 3. Specifically, the challenger no longer checks the norm constraint on $\mathbf{r}_i$.

3. The challenger parses $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ where $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. Then it works as follows:
   - Sample $\mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}$
   - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$.
   - Initialize an ordered list $L = \emptyset$. For $i \in [\lambda]$:
     * Sample $\mathbf{r}_i \leftarrow D_{\mathbb{Z},\chi}^m$.
     * ~~If $\|\mathbf{r}_i\| > B_M$ then set $\mathbf{y}_i = \perp^{\ell_{\mathsf{out}}}$ and $\mathbf{r}_i = \mathbf{0}^m$. Add $(\mathbf{y}_i, \mathbf{r}_i)$ to L and continue to the next iteration.~~
     * Compute $\mathbf{w}_i = \mathbf{C}_{\mathsf{br}} \mathbf{r}_i$.
     * Compute $\mathbf{y}_i = \mathsf{RoundB}_{q,B_R}(\mathbf{w}_i)$ and add $(\mathbf{y}_i, \mathbf{r}_i)$ to L.
   - Let $(\mathbf{y}^*, \mathbf{r}^*)$ be the first element of L for which there does not exist any index $j \in [\ell_{\mathsf{out}}]$ such that $y_j^* = \perp$. If such a tuple does not exist then set $\mathbf{y} = \mathbf{0}^{\ell_{\mathsf{out}}}$, $\boldsymbol{\pi} = \mathbf{0}^m$. Otherwise set $\mathbf{y} = \mathbf{y}^*$ and $\boldsymbol{\pi} = \mathbf{r}^*$.

   Finally send $\mathsf{br}, \mathbf{y}$ and $\boldsymbol{\pi}$ to the adversary $\mathcal{A}$.

- Hyb_3: Same as Hyb_2 except the challenger aborts if $\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$, where $\mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ was defined in Eq. (6.9).

3. The challenger parses $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ where $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. Then it works as follows:
   - Sample $\mathsf{br} \xleftarrow{\mathrm{R}} \mathfrak{B}$
   - If $\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $\mathbf{y} \xleftarrow{\mathrm{R}} \{0,1\}^{\ell_{\mathsf{out}}}$, and $\boldsymbol{\pi} = \mathbf{0}^m$ to the adversary $\mathcal{A}$.
   - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$.
   - Initialize an ordered list $L = \emptyset$. For $i \in [\lambda]$:
     * Sample $\mathbf{r}_i \leftarrow D_{\mathbb{Z},\chi}^m$.
     * Compute $\mathbf{w}_i = \mathbf{C}_{\mathsf{br}} \mathbf{r}_i$.
     * Compute $\mathbf{y}_i = \mathsf{RoundB}_{q,B_R}(\mathbf{w}_i)$ and add $(\mathbf{y}_i, \mathbf{r}_i)$ to L.
   - Let $(\mathbf{y}^*, \mathbf{r}^*)$ be the first element of L for which there does not exist any index $j \in [\ell_{\mathsf{out}}]$ such that $y_j^* = \perp$. If such a tuple does not exist then set $\mathbf{y} = \mathbf{0}^{\ell_{\mathsf{out}}}$, $\boldsymbol{\pi} = \mathbf{0}^m$. Otherwise set $\mathbf{y} = \mathbf{y}^*$ and $\boldsymbol{\pi} = \mathbf{r}^*$.

   Finally send $\mathsf{br}, \mathbf{y}$ and $\boldsymbol{\pi}$ to the adversary $\mathcal{A}$.

- Hyb_4: Same as Hyb_3, except the challenger changes how it samples $(\mathbf{w}_i, \mathbf{r}_i)$ in each iteration:

3. The challenger parses $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ where $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. Then it works as follows:

- Sample $\mathsf{br} \xleftarrow{R} \mathfrak{B}$
- If $\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $\mathbf{y} \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$, and $\boldsymbol{\pi} = \mathbf{0}^m$ to the adversary $\mathcal{A}$.
- Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$.
- Initialize an ordered list $L = \emptyset$. For $i \in [\lambda]$:
  - Sample $\mathbf{w}_i \xleftarrow{R} \mathbb{Z}_q^{\ell_{\mathrm{out}}}$.
  - Sample $\mathbf{r}_i \leftarrow \mathbf{C}_{\mathsf{br},\chi}^{-1}(\mathbf{w}_i)$.
  - Compute $\mathbf{y}_i = \mathsf{RoundB}_{q,B_R}(\mathbf{w}_i)$ and add $(\mathbf{y}_i, \mathbf{r}_i)$ to L.
- Let $(\mathbf{y}^*, \mathbf{r}^*)$ be the first element of L for which there does not exist any index $j \in [\ell_{\mathrm{out}}]$ such that $y_j^* = \perp$. If such a tuple does not exist then set $\mathbf{y} = \mathbf{0}^{\ell_{\mathrm{out}}}$, $\boldsymbol{\pi} = \mathbf{0}^m$. Otherwise set $\mathbf{y} = \mathbf{y}^*$ and $\boldsymbol{\pi} = \mathbf{r}^*$.

Finally send $\mathsf{br}, \mathbf{y}$ and $\boldsymbol{\pi}$ to the adversary $\mathcal{A}$.

- Hyb_5: Same as Hyb_4, except the challenger replaces the iterative sampling with sampling a single vector $\mathbf{w} \in \mathbb{Z}_q^{\ell_{\mathrm{out}}}$.

3. The challenger parses $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ where $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathrm{out}} \times m}$. Then it works as follows:
   - Sample $\mathsf{br} \xleftarrow{R} \mathfrak{B}$
   - If $\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $\mathbf{y} \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$, and $\boldsymbol{\pi} = \mathbf{0}^m$ to the adversary $\mathcal{A}$.
   - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$.
   - Sample $\mathbf{w} \xleftarrow{R} \mathsf{Good}_{q,B_R}^{\ell_{\mathrm{out}}} = (\mathsf{Good}_{q,B_R}^{-} \cup \mathsf{Good}_{q,B_R}^{+})^{\ell_{\mathrm{out}}}$. We refer to Eqs. (6.1) to (6.3) for the definitions of these sets.
   - Sample $\mathbf{r} \leftarrow \mathbf{C}_{\mathsf{br},\chi}^{-1}(\mathbf{w})$. Set $\boldsymbol{\pi} = \mathbf{r}$.
   - Compute $\mathbf{y} = \mathsf{RoundB}_{q,B_R}(\mathbf{w})$.

   Finally send $\mathsf{br}, \mathbf{y}$ and $\boldsymbol{\pi}$ to the adversary $\mathcal{A}$.

Note that iterating over $i \in [\lambda]$ is no longer necessary, since by definition RoundB_{q,B_R} never outputs $\perp$ when the components of $\mathbf{w}$ lie in Good_{q,B_R}.

- Hyb_6: Same as Hyb_5 except the challenger changes how it samples $(\mathbf{w}, \mathbf{y})$.

3. The challenger parses $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ where $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathrm{out}} \times m}$. Then it works as follows:
   - Sample $\mathsf{br} \xleftarrow{R} \mathfrak{B}$
   - If $\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $\mathbf{y} \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$, and $\boldsymbol{\pi} = \mathbf{0}^m$ to the adversary $\mathcal{A}$.
   - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$.
   - Sample $\mathbf{y} \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$.
   - Sample $\mathbf{w} \leftarrow \mathsf{RoundB}_{q,B_R}^{-1}(\mathbf{y})$.
   - Sample $\mathbf{r} \leftarrow \mathbf{C}_{\mathsf{br},\chi}^{-1}(\mathbf{w})$. Set $\boldsymbol{\pi} = \mathbf{r}$.

   Finally send $\mathsf{br}, \mathbf{y}$ and $\boldsymbol{\pi}$ to the adversary $\mathcal{A}$.

- Hyb_7: Same as Hyb_6 except the challenger changes how it samples $\mathbf{r}$.

3. The challenger parses $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ where $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathrm{out}} \times m}$. Then it works as follows:
   - Sample $\mathsf{br} \xleftarrow{R} \mathfrak{B}$
   - If $\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $\mathbf{y} \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$, and $\boldsymbol{\pi} = \mathbf{0}^m$ to the adversary $\mathcal{A}$.
   - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$.
   - Let $\mathbf{R}_{\mathbf{C},\mathsf{br}}$ be a gadget trapdoor for $\mathbf{C}_{\mathsf{br}}$, such that $\left\|\mathbf{R}_{\mathbf{C},\mathsf{br}}\right\| = 1$ and $\mathbf{C}_{\mathsf{br}}\mathbf{R}_{\mathbf{C},\mathsf{br}} = \mathbf{G}$.
   - Sample $\mathbf{y} \xleftarrow{R} \{0,1\}^{\ell_{\mathrm{out}}}$.
   - Sample $\mathbf{w} \leftarrow \mathsf{RoundB}_{q,B_R}^{-1}(\mathbf{y})$.
   - Sample $\mathbf{r} \leftarrow \mathsf{SamplePre}(\mathbf{C}_{\mathsf{br}}, \mathbf{R}_{\mathbf{C},\mathsf{br}}, \mathbf{w}, \chi)$. Set $\boldsymbol{\pi} = \mathbf{r}$.

Finally send br, $\mathbf{y}$ and $\boldsymbol{\pi}$ to the adversary $\mathcal{A}$.

This is the target randomness game for $b = 1$.

Let $\mathsf{Hyb}_i(\mathcal{A})$ be the random variable denoting the output of an execution of experiment Hyb_i with adversary $\mathcal{A}$. We now analyze each adjacent pair of hybrid experiments.

Claim 6.17. *Suppose $B_M \geq \sqrt{\lambda}\chi$. Then for all polynomials $\ell_{out} = \ell_{out}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. The only difference between the hybrids is that in Hyb_2, the challenger no longer checks if $\|\mathbf{r}_i\| > B_M$. It suffices to argue that the probability that there exists an $i \in [\lambda]$ and $j \in [m]$ such that $|r_{i,j}| > B_M$ is negligible. Since $B_M \geq \sqrt{\lambda}\chi$, the Gaussian tail bound (Lemma 6.2) implies that $\Pr[|r_{i,j}| > B_M] \leq 2^{-\lambda}$. Since m is polynomial in λ, a union bound implies that the probability that such an $r_{i,j}$ exists is at most $m \cdot \lambda \cdot 2^{-\lambda}$, which is negligible in λ. Thus the claim follows. □

Claim 6.18. *Suppose $k \geq 4\ell_{out} \lceil \log q \rceil$. Then for all polynomials $\ell_{out} = \ell_{out}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. The only difference between the hybrids occurs when we sample a br such that $\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$. Recall that $\mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ is the set of branches where $\mathbf{C}_{\mathsf{br}}$ does not have a gadget trapdoor. Since $k \geq 4\ell_{\mathsf{out}} \lceil \log q \rceil$, Corollary 6.16 implies that there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}|/|\mathfrak{B}| = \mathsf{negl}(\lambda)$. Since br is chosen uniformly at random from $\mathfrak{B}$, $\Pr[\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}] = \mathsf{negl}(\lambda)$ and the claim follows. □

Claim 6.19. *Suppose $\chi \geq 4m \log m$. Then for all polynomials $\ell_{out} = \ell_{out}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. We define a series of $\lambda + 1$ hybrids corresponding to the iterations of the for loop. We only focus on Step 3 of the hybrids. The remaining steps are the same as in Hyb_3 and Hyb_4. For $i^* \in [\lambda + 1]$, define Hyb_{3,i^*} to be the same as Hyb_3 and Hyb_4 except for the following:

3. The challenger parses $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp})$ where $\mathbf{C} \in \mathbb{Z}_q^{\ell_{\mathsf{out}} \times m}$. Then it works as follows:
   - Sample $\mathsf{br} \xleftarrow{R} \mathfrak{B}$
   - If $\mathsf{br} \in \mathfrak{B}_{\mathsf{pp}}^{\mathsf{BAD}}$ then abort and send br, $\mathbf{y} \xleftarrow{R} \{0,1\}^{\ell_{\mathsf{out}}}$, and $\boldsymbol{\pi} = \mathbf{0}^m$ to the adversary $\mathcal{A}$.
   - Compute $\mathbf{C}_{\mathsf{br}} = \mathbf{C} + \mathbf{H}_{\mathsf{br}}$.
   - Initialize an ordered list $L = \emptyset$. For $i \in [\lambda]$:
     - If $i < i^*$
       - Sample $\mathbf{w}_i \xleftarrow{R} \mathbb{Z}_q^{\ell_{\mathsf{out}}}$.
       - Compute $\mathbf{r}_i \leftarrow \mathbf{C}_{\mathsf{br},\chi}^{-1}(\mathbf{w}_i)$.
     - Else
       - Sample $\mathbf{r}_i \leftarrow D_{\mathbb{Z},\chi}^m$.
       - Compute $\mathbf{w}_i = \mathbf{C}_{\mathsf{br}} \mathbf{r}_i$.
     - Compute $\mathbf{y}_i = \mathsf{RoundB}_{q,B_R}(\mathbf{w}_i)$ and add $(\mathbf{y}_i, \mathbf{r}_i)$ to L.
   - Let $(\mathbf{y}^*, \mathbf{r}^*)$ be the first element of L for which there does not exist any index $j \in [\ell_{\mathsf{out}}]$ such that $y_j^* = \perp$. If such a tuple does not exist then set $\mathbf{y} = \mathbf{0}^{\ell_{\mathsf{out}}}$, $\boldsymbol{\pi} = \mathbf{0}^m$. Otherwise set $\mathbf{y} = \mathbf{y}^*$ and $\boldsymbol{\pi} = \mathbf{r}^*$.

   Finally send br, $\mathbf{y}$ and $\boldsymbol{\pi}$ to the adversary $\mathcal{A}$.

Note that $\mathsf{Hyb}_{3,1}$ corresponds to Hyb_3, whereas $\mathsf{Hyb}_{3,\lambda+1}$ corresponds to Hyb_4. We proceed to show that each adjacent pair of hybrids is statistically indistinguishable.

Claim 6.20. *Suppose $\chi \geq 4m \log m$. Then for all polynomials $\ell_{out} = \ell_{out}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_{3,i^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{3,i^*+1}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. The only difference between the hybrids occurs for iteration $i = i^*$ of the for loop. In Hyb_{3,i^*}, the challenger samples $\mathbf{r}_{i^*} \leftarrow D^m_{\mathbb{Z},\chi}$, $\mathbf{w}_{i^*} = \mathbf{C}_{\mathsf{br}}\mathbf{r}_{i^*}$, while in Hyb_{3,i^*+1}, the challenger samples $\mathbf{w}_{i^*} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{\ell_{\mathsf{out}}}$, $\mathbf{r}_{i^*} \leftarrow \mathbf{C}^{-1}_{\mathsf{br},\chi}(\mathbf{w}_{i^*})$. Suppose the challenger does not abort. Then $\mathsf{br} \notin \mathfrak{B}^{\mathsf{BAD}}_{\mathsf{pp}}$ and the matrix $\mathbf{C}_{\mathsf{br}}$ has a gadget trapdoor $\mathbf{R}_{\mathsf{C},\mathsf{br}} \in \{-1,0,1\}^{m\times m'}$ with $\left\|\mathbf{R}_{\mathsf{C},\mathsf{br}}\right\| = 1$. Since $\chi \geq 4m\log m$, the conditions of Lemma 6.4 are satisfied, implying that there exists a negligible function $\mathsf{negl}(\cdot)$ such that the statistical distance between the joint distribution of $(\mathbf{r}_{i^*}, \mathbf{w}_{i^*})$ in Hyb_{3,i^*} and Hyb_{3,i^*+1} is $\mathsf{negl}(m)$. Furthermore, since k, N, and ℓ_{out} are polynomials in λ and $m = k \cdot N \cdot \ell_{\mathsf{out}}$, the statistical distance between the distributions is also negligible in λ, and the claim follows. □

Claim 6.19 now follows from Claim 6.20 by a standard hybrid argument. □

Claim 6.21. *Suppose $q > 2\ell_{out}(4B_R + 3)$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Observe that for any $i \in [\lambda], j \in [\ell_{\mathsf{out}}]$, $\mathsf{RoundB}_{q,B_R}(w_{i,j}) = \perp$ if and only if $w_{i,j} \notin \mathsf{Good}_{q,B_R}$. Then the only difference between the hybrids occurs when, for all λ iterations of Hyb_4, the challenger samples a $\mathbf{w}_i$ with an index j for which $w_{i,j} \notin \mathsf{Good}_{q,B_R}$. Denote this event for iteration i by E_i. Then by a union bound over the ℓ_{out} components, $\Pr[E_i] \leq \ell_{\mathsf{out}}(4B_R + 3)/q$. Since the challenger samples $\mathbf{w}_i$ independently over the λ iterations, $\Pr[\forall i \in [\lambda] : E_i] \leq (\ell_{\mathsf{out}}(4B_R + 3)/q)^\lambda < 2^{-\lambda}$ which is negligible in λ. Thus the claim follows. □

Claim 6.22. *It holds that $\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]$.*

Proof. It suffices to show that the joint distribution of $(\mathbf{w}, \mathbf{y})$ in both hybrids is the same. Note that by definition (Eqs. (6.1) to (6.3)), $|\mathsf{Good}^+_{q,B_R}| = |\mathsf{Good}^-_{q,B_R}| = |\mathsf{Good}_{q,B_R}|/2$. Let W, Y be the random variables denoting the value of $\mathbf{w}, \mathbf{y}$ in the experiments. Consider the distribution of (W, Y). For a pair $\mathbf{w}, \mathbf{y}$ such that $\mathbf{y} = \mathsf{RoundB}_{q,B_R}(\mathbf{w})$:

- In Hyb_5, $\Pr[W = \mathbf{w}, Y = \mathbf{y}] = |\mathsf{Good}_{q,B_R}|^{-\ell_{\mathsf{out}}}$.
- In Hyb_6, $\Pr[W = \mathbf{w}, Y = \mathbf{y}] = \Pr[W = \mathbf{w} : Y = \mathbf{y}] \cdot 2^{-\ell_{\mathsf{out}}} = (|\mathsf{Good}_{q,B_R}|/2)^{-\ell_{\mathsf{out}}} \cdot 2^{-\ell_{\mathsf{out}}} = |\mathsf{Good}_{q,B_R}|^{-\ell_{\mathsf{out}}}$.

As the distributions are identical, the claim follows. □

Claim 6.23. *Suppose $\chi \geq m \cdot \log \ell_{out}$ and $k \geq 4\ell_{out}\lceil \log q \rceil$. Then for all polynomials $\ell_{out} = \ell_{out}(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. The only difference between the hybrids is in the distribution of $\mathbf{r}$. In Hyb_6, $\mathbf{r} \leftarrow \mathbf{C}^{-1}_{\mathsf{br},\chi}(\mathbf{w})$ while in Hyb_7, $\mathbf{r} \leftarrow \mathsf{SamplePre}(\mathbf{C}_{\mathsf{br}}, \mathbf{R}_{\mathsf{C},\mathsf{br}}, \mathbf{w}, \chi)$. Since $\left\|\mathbf{R}_{\mathsf{C},\mathsf{br}}\right\| = 1$, $\chi \geq m\cdot\log \ell_{\mathsf{out}}$, $m = k\cdot N\cdot \ell_{\mathsf{out}} \geq 3\ell_{\mathsf{out}}\lceil \log q \rceil$ and ℓ_{out} is polynomial in λ, we appeal to Theorem 6.3 to conclude that there exists a negligible function $\mathsf{negl}(\cdot)$ such that the statistical distance between these distributions of $\mathbf{r}$ is $\mathsf{negl}(\lambda)$. Thus the claim follows. □

Proof of Theorem 6.13. Combining Claims 6.17 to 6.19 and 6.21 to 6.23, we conclude via a hybrid argument that there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$$

and the claim follows. □

Theorem 6.24 (Sampling on Constricted Branch)**.** *Suppose $B_R = B_M \cdot B_G \cdot m$. For all polynomials $n = n(\lambda)$ and $q = q(\lambda, \ell_{out})$ such that $\log q = O(\lambda)$, Construction 6.7 satisfies sampling on constricted branch.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, output length $\ell_{\mathsf{out}} \in \mathbb{N}$, and branch $\mathsf{br}^* \in \mathfrak{B}$. Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{out}}}, \mathsf{br}^*)$. By construction and Definition 6.1, $\mathbf{C} = \mathsf{Samp}(1^\lambda, q, t, \mathsf{pp}) = \mathbf{SA} - \mathbf{H}_{\mathsf{br}^*} + \mathbf{E}$ and $\mathsf{aux} = \mathbf{S}$. Define the set

$$\mathsf{Range} \coloneqq \left\{\mathsf{RoundS}(\mathbf{S}\boldsymbol{\alpha}) : \boldsymbol{\alpha} \in \mathbb{Z}_q^n\right\}$$

Note that by definition, $|\mathsf{Range}| \leq q^n$. Fix any $\mathbf{y}^* \in \{0,1\}^{\ell_{\mathsf{out}}}$ and proof $\boldsymbol{\pi}^*$ such that $\mathsf{Verify}(\mathsf{pp}, \mathsf{br}^*, \mathbf{y}^*, \boldsymbol{\pi}^*) = 1$. We show $\mathbf{y}^* \in \mathsf{Range}$. We consider two cases based on the structure of the verification algorithm:

- **Case 1:** $\boldsymbol{\pi}^* = \mathbf{0}^m$ and $\mathbf{y}^* = \mathbf{0}^{\ell_{\text{out}}}$. Then $\mathbf{y}^* = \mathsf{RoundS}(\mathbf{S} \cdot \mathbf{0}^n)$, hence $\mathbf{y}^* \in \mathsf{Range}$.
- **Case 2:** $\boldsymbol{\pi}^* = \mathbf{r}^* \in \mathbb{Z}_q^m$ such that $\|\mathbf{r}^*\| \leq B_M$ and $\mathbf{y}^* = \mathsf{RoundB}_{q,B_R}((\mathbf{C} + \mathbf{H}_{\text{br}^*}) \cdot \mathbf{r}^*)$ such that for all $j \in [\ell_{\text{out}}]$, $y_j^* \neq \perp$. Since for all $j \in [\ell_{\text{out}}]$, $y_j^* \neq \perp$, it means that all the components of $\mathbf{w}^* = (\mathbf{C} + \mathbf{H}_{\text{br}^*}) \cdot \mathbf{r}^*$ are in the set Good_{q,B_R} (see Eq. (6.3)) and $\|\mathbf{r}^*\| \leq B_M$. Then we have $\|\mathbf{E}\mathbf{r}^*\| \leq \|\mathbf{E}\| \cdot \|\mathbf{r}^*\| \cdot m \leq B_M \cdot B_G \cdot m = B_R$. We argue that

$$\mathsf{RoundS}(\mathbf{S}\mathbf{A}\mathbf{r}^*) = \mathsf{RoundB}_{q,B_R}((\mathbf{S}\mathbf{A} + \mathbf{E}) \cdot \mathbf{r}^*).$$

  Let $\mathbf{u} = \mathbf{S}\mathbf{A}\mathbf{r}^*$ and consider any index $i \in [\ell_{\text{out}}]$ such that $\mathsf{RoundB}_{q,B_R}(w_i^*) = 1$.

  - By definition of RoundB_{q,B_R} (Eq. (6.4)) this implies $w_i^* \in (B_R, \lfloor q/2 \rfloor - B_R)$.
  - Since $\|\mathbf{E}\mathbf{r}^*\| \leq B_R$, $u_i \in (0, \lfloor q/2 \rfloor)$.
  - Then by definition of RoundS (Eq. (6.4)), it holds that $\mathsf{RoundS}(u_i) = 1$.

  An analogous argument holds when $\mathsf{RoundB}_{q,B_R}(w_i^*) = 0$ as well. This implies $\mathbf{y}^* = \mathsf{RoundS}(\mathbf{u})$ and $\mathbf{y}^* \in \mathsf{Range}$.

Thus for any $\mathbf{y}^*$ that verifies, $\mathbf{y}^* \in \mathsf{Range}$, implying that there exists an $\boldsymbol{\alpha}^* \in \mathbb{Z}_q^n$ such that $\mathbf{y}^* = \mathsf{RoundS}(\mathbf{S}\boldsymbol{\alpha}^*)$. Recall that the SampleGuess algorithm takes as input $\mathsf{aux} = \mathbf{S}$, samples $\boldsymbol{\alpha} \xleftarrow{\text{R}} \mathbb{Z}_q^n$ and outputs $\mathbf{y} = \mathsf{RoundS}(\mathbf{S}\boldsymbol{\alpha})$. Therefore,

$$\Pr[\mathbf{y} = \mathbf{y}^* : \mathbf{y} \leftarrow \mathsf{SampleGuess}(\mathsf{aux})] \;\geq\; \Pr[\boldsymbol{\alpha} = \boldsymbol{\alpha}^*] \;=\; q^{-n} = 2^{-n \log q}.$$

Since n is polynomial in λ and q satisfies $\log q = O(\lambda)$, there exists a polynomial p such that $2^{-n \log q} \geq 2^{-p(\lambda)}$. Thus the claim follows. □

Parameter instantiation. Let λ be a security parameter and ℓ_{out} be an output length. We now provide one possible instantiation of the parameters in Construction 6.7 to satisfy Theorems 6.8, 6.9, 6.13 and 6.24. In the following, we assume that $\ell_{\text{out}} < 2^\lambda$.

- When setting parameters, we work under the assumption that $q \leq 2^{O(\lambda)}$. Our final parameter instantiations will satisfy this property. In this case, $\log q = O(\lambda)$.
- We set $k = 4\ell_{\text{out}} \lceil \log q \rceil = O(\ell_{\text{out}}\lambda)$. This gives $m = k \cdot N \cdot \ell_{\text{out}} = O(\lambda^2 \ell_{\text{out}}^2)$.
- We set $\chi = 4m^2 = O(\lambda^4 \ell_{\text{out}}^4)$.
- We set $t = \lambda^3$.
- We set the bounds to be $B_M = \sqrt{\lambda}\chi$, $B_G = \sqrt{\lambda + \lambda^3}\chi$ and $B_R = B_M \cdot B_G \cdot m = O(\lambda^{12}\ell_{\text{out}}^{10})$.
- We choose a prime modulus $q = \mathsf{poly}(\lambda, \ell_{\text{out}})$ such that $q \geq 10\ell_{\text{out}} B_R = O(\lambda^{12}\ell_{\text{out}}^{11})$. In particular, since $q = \mathsf{poly}(\lambda, \ell_{\text{out}})$ and $\ell_{\text{out}} < 2^\lambda$, this means $q \leq 2^{O(\lambda)}$ which satisfies our initial assumption.
- Finally, we choose $n = \lambda^{3/\delta}$ for a constant $\delta \in (0, 1)$, such that the 2^{-n^δ}-$\mathsf{LWE}_{n,m,q,\chi}$ assumption holds.
- Since $N = 8\lambda$, the above setting of parameters yields a branch set with size $|\mathfrak{B}| = (qk)^N \leq 2^{O(\lambda^2)}$.

We briefly verify that these parameters satisfy the necessary requirements:

- Since $B_G = \sqrt{\lambda + \lambda^3}\chi$, and the $2^{-n^\delta} = 2^{-\lambda^3}$-$\mathsf{LWE}_{n,m,q,\chi}$ assumption holds, the conditions of Theorem 6.9 are satisfied. In particular, mode indistinguishability holds with $\varepsilon' = 2^{-\lambda^3}$.
- For Theorem 6.13, we have $B_M = \sqrt{\lambda}\chi$, $\chi = 4m^2 \geq \max(4 \cdot m \cdot \log m,\ m \cdot \log \ell_{\text{out}})$, $m \geq 3\ell_{\text{out}} \lceil \log q \rceil$, $k = 4\ell_{\text{out}} \lceil \log q \rceil$ and $q \geq 10\ell_{\text{out}} B_R > 2\ell_{\text{out}}(4B_R + 3)$. Thus, the conditions of the theorem are satisfied.
- For Theorem 6.24, note that $B_R = B_M \cdot B_G \cdot m$, $n(\lambda)$ is polynomial in λ, and $\log q = O(\lambda)$, hence the requirements are met.

Corollary 6.25 (Branch-Constricting Generator from LWE)**.** *Let λ be a security parameter. Assuming sub-exponential security of LWE with a polynomial modulus-to-noise ratio, there exists a branch-constricting generator that satisfies $2^{-\lambda^3}$-mode indistinguishability with a branch size of $2^{O(\lambda^2)}$.*

In conjunction with Corollary 4.8, we now obtain a sometimes-constricting generator from the LWE assumption with a polynomial modulus-to-noise ratio.

Corollary 6.26 (Sometimes-Constricting Generator from LWE)**.** *Let λ be a security parameter. Assuming sub-exponential security of LWE with a polynomial modulus-to-noise ratio, there exists a polynomial $p = p(\lambda)$ and a sometimes-constricting generator that satisfies completeness, mode indistinguishability, μ-guessing security, and target randomness where $\mu = 2^{-p(\lambda)}$.*

Acknowledgments

Brent Waters is supported by NSF CNS-2318701 and a Simons Investigator Award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Sloan Fellowship, a Microsoft Research Faculty Fellowship, a Google Research Scholar Award, an Amazon Research Award, and a gift from the Stellar Development Foundation.

References

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *CRYPTO*, 2010.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, 1996.

[AP09] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, 2009.

[BCD+25] Pedro Branco, Arka Rai Choudhuri, Nico Döttling, Abhishek Jain, Giulio Malavolta, and Akshayaram Srinivasan. Black-box non-interactive zero knowledge from vector trapdoor hash. In *EUROCRYPT*, 2025.

[BFJ+20] Saikrishna Badrinarayanan, Rex Fernando, Aayush Jain, Dakshita Khurana, and Amit Sahai. Statistical ZAP arguments. In *EUROCRYPT*, 2020.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, 1988.

[BKM20] Zvika Brakerski, Venkata Koppula, and Tamer Mour. NIZK from LPN and trapdoor hash via correlation intractability for approximable relations. In *CRYPTO*, 2020.

[BLN+25] Eli Bradley, George Lu, Shafik Nassar, Brent Waters, and David J. Wu. A hidden-bits approach to statistical ZAPs from LWE. In *TCC*, 2025.

[BLV03] Boaz Barak, Yehuda Lindell, and Salil P. Vadhan. Lower bounds for non-black-box zero knowledge. In *FOCS*, 2003.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat-Shamir: from practice to theory. In *STOC*, 2019.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, 2003.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, 2010.

[CJJQ23] Geoffroy Couteau, Abhishek Jain, Zhengzhong Jin, and Willy Quach. A note on non-interactive zero-knowledge from CDH. In *CRYPTO*, 2023.

[CKSU21] Geoffroy Couteau, Shuichi Katsumata, Elahe Sadeghi, and Bogdan Ursu. Statistical ZAPs from group-based assumptions. In *TCC*, 2021.

[CKU20] Geoffroy Couteau, Shuichi Katsumata, and Bogdan Ursu. Non-interactive zero-knowledge in pairing-free groups from weaker assumptions. In *EUROCRYPT*, 2020.

[DJJ24] Quang Dao, Aayush Jain, and Zhengzhong Jin. Non-interactive zero-knowledge from LPN and MQ. In *CRYPTO*, 2024.

[DN00] Cynthia Dwork and Moni Naor. Zaps and their applications. In *FOCS*, 2000.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge L. Villar. An algebraic framework for Diffie-Hellman assumptions. In *CRYPTO*, 2013.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *FOCS*, 1990.

[GJJM20] Vipul Goyal, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Statistical Zaps and new oblivious transfer protocols. In *EUROCRYPT*, 2020.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptol.*, 7(1), 1994.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4), 1999.

[JJ21] Abhishek Jain and Zhengzhong Jin. Non-interactive zero knowledge from sub-exponential DDH. In *EUROCRYPT*, 2021.

[LPWW20] Benoît Libert, Alain Passelègue, Hoeteck Wee, and David J. Wu. New constructions of statistical NIZKs: Dual-mode DV-NIZKs and more. In *EUROCRYPT*, 2020.

[LVW19] Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. 2-message publicly verifiable WI from (subexponential) LWE. *IACR Cryptol. ePrint Arch.*, 2019.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. In *FOCS*, 2004.

[Pei08] Chris Peikert. Limits on the hardness of lattice problems in ℓ_p norms. *Comput. Complex.*, 17(2), 2008.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In *CRYPTO*, 2019.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC*, 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

[Wat24] Brent Waters. A new approach for non-interactive zero-knowledge from learning with errors. In *STOC*, 2024.

[WWW25] Brent Waters, Hoeteck Wee, and David J. Wu. New techniques for preimage sampling: Improved NIZKs and more from LWE. In *EUROCRYPT*, 2025.

A Proof of Lemma 2.9 (Complexity Leveraging)

Suppose $\Pi_{\mathsf{NIZK1}} = (\mathsf{Setup}_1, \mathsf{Prove}_1, \mathsf{Verify}_1)$ is a NIZK satisfying completeness, witness-indistinguishability, and $(1, 2^{-\lambda^{\varepsilon_s}})$-non-adaptive soundness for constant $\varepsilon_s \in (0,1)$ in the common random string model. Let $p(\lambda)$ be any polynomial. Then for some constant $c \in \mathbb{N}$, there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$, $p(\lambda) \leq \lambda^c$. Define the rescaled security parameter $\tilde{\lambda} \coloneqq \lambda^{c/\varepsilon_s}$. We construct a NIZK $\Pi_{\mathsf{NIZK2}} = (\mathsf{Setup}_2, \mathsf{Prove}_2, \mathsf{Verify}_2)$ satisfying completeness, witness-indistinguishability, and $(1, 2^{-p(\lambda)})$-non-adaptive soundness in the common random string model as follows:

- $\mathsf{Setup}_2(1^\lambda, 1^n)$ outputs $\mathsf{Setup}_1(1^{\tilde{\lambda}}, 1^n)$ where $\tilde{\lambda} = \lambda^{c/\varepsilon_s}$.
- $\mathsf{Prove}_2(\mathsf{crs}, x, w)$ outputs $\mathsf{Prove}_1(\mathsf{crs}, x, w)$.
- $\mathsf{Verify}_2(\mathsf{crs}, x, \pi)$ outputs $\mathsf{Verify}_1(\mathsf{crs}, x, \pi)$.

Completeness. Immediate from the completeness of Π_{NIZK1}.

Soundness. Let $\mathcal{A}$ be any efficient adversary against the soundness of Π_{NIZK2}. By construction, $\mathcal{A}$ induces an adversary against Π_{NIZK1} with security parameter $\tilde{\lambda}$. By the soundness of Π_{NIZK1}, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all λ,

$$\Pr[\mathcal{A} \text{ wins the soundness game for } \Pi_{\mathsf{NIZK2}}] \leq 2^{-\tilde{\lambda}^{\varepsilon_s}} \cdot \mathsf{negl}(\tilde{\lambda}) = 2^{-\lambda^c} \cdot \mathsf{negl}(\tilde{\lambda}).$$

For all $\lambda > \lambda_0$, we have $\lambda^c \geq p(\lambda)$, and thus

$$\Pr[\mathcal{A} \text{ wins the soundness game for } \Pi_{\mathsf{NIZK2}}] \leq 2^{-p(\lambda)} \cdot \mathsf{negl}(\tilde{\lambda}).$$

Since $\tilde{\lambda} = \mathsf{poly}(\lambda)$, $\mathsf{negl}(\tilde{\lambda})$ is negligible in λ, completing the soundness proof.

Witness indistinguishability. Witness indistinguishability also follows directly from that of Π_{NIZK1}. Indeed, for any efficient adversary $\mathcal{A}$, the witness-indistinguishability advantage of $\mathcal{A}$ against Π_{NIZK1}, $\mathsf{WIAdv}_{\mathsf{NIZK}}(\mathcal{A}) \leq \mathsf{negl}(\tilde{\lambda})$, which is negligible in λ. □

B Proof of Lemma 6.4 (Gaussian Samples)

In this section, we give a self-contained proof of Lemma 6.4. To argue this, we show that whenever a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ has a "good" gadget trapdoor, the SIS-lattice associated with $\mathbf{A}$ (i.e., the lattice $\Lambda^\perp(\mathbf{A}) \coloneqq \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A}\mathbf{x} = \mathbf{0}^n \bmod q\}$) has a small smoothing parameter. Lemma 6.4 then follows immediately by [GPV08]. We start by stating a few preliminaries and helper lemmas from prior works:

Lattices. Let $\mathbf{B} \in \mathbb{R}^{n \times n}$ be a full-rank matrix (over $\mathbb{R}$). Then the n-dimensional lattice $\Lambda = \mathcal{L}(\mathbf{B})$ generated by $\mathbf{B}$ is defined to be $\Lambda \coloneqq \mathbf{B} \cdot \mathbb{Z}^n = \{\mathbf{B}\mathbf{z} : \mathbf{z} \in \mathbb{Z}^n\}$.

Dual lattices. For a lattice Λ, we define the *dual lattice* $\Lambda^* = \{\mathbf{w} \in \mathbb{R}^n : \forall \mathbf{x} \in \Lambda, \mathbf{w}^\intercal \mathbf{x} \in \mathbb{Z}\}$. If $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ for integers n, m, q, we define the q-ary lattices

$$\Lambda^\perp(\mathbf{A}) \coloneqq \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{A}\mathbf{x} = \mathbf{0}^n \bmod q\}$$
$$\Lambda(\mathbf{A}) \coloneqq \left\{\mathbf{y} \in \mathbb{Z}^m : \mathbf{y} = \mathbf{A}^\intercal \mathbf{x} \bmod q \text{ for some } \mathbf{x} \in \mathbb{Z}^n\right\}.$$

It can be seen from their definitions that $\Lambda^\perp(\mathbf{A})$ and $\Lambda(\mathbf{A})$ are scaled duals:

$$\Lambda^\perp(\mathbf{A}) = q \cdot \Lambda(\mathbf{A})^* \quad \text{and} \quad \Lambda(\mathbf{A}) = q \cdot (\Lambda^\perp(\mathbf{A}))^*$$

We write $\lambda_1^\infty(\Lambda) \coloneqq \min_{\mathbf{v} \in \Lambda \setminus \{\mathbf{0}\}} \|\mathbf{v}\|$ to denote the ℓ_∞-norm of the shortest non-zero vector in Λ.

Smoothing parameter. We recall the notion of the smoothing parameter introduced in [MR04]. For an n-dimensional lattice Λ and a positive real number $\varepsilon > 0$, the smoothing parameter $\eta_\varepsilon(\Lambda)$ is the smallest real value $\chi > 0$ such that $\rho_{1/\chi}(\Lambda^*) \leq 1 + \varepsilon$.

Lemma B.1 (Smoothing Parameter Bound [Pei08, Lemma 3.5])**.** *Let Λ be an n-dimensional lattice. Then there exists a negligible function $\varepsilon = \varepsilon(n)$ such that $\eta_\varepsilon(\Lambda) \leq \log n / \lambda_1^\infty(\Lambda^*)$.*

Corollary B.2 (Smoothing Parameter of $\Lambda^\perp(\mathbf{A})$ [GPV08, Lemma 5.3])**.** *Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and suppose $\lambda_1^\infty(\Lambda(\mathbf{A})) \geq t$. Then there exists a negligible function $\varepsilon = \varepsilon(m)$ such that $\eta_\varepsilon(\Lambda^\perp(\mathbf{A})) \leq q/t \cdot \log m$.*

Proof. Since $\Lambda(\mathbf{A})$ and $\Lambda^\perp(\mathbf{A})$ are scaled duals, we have $\Lambda(\mathbf{A}) = q \cdot (\Lambda^\perp(\mathbf{A}))^*$ and $\lambda_1^\infty(\Lambda(\mathbf{A})) = q \cdot \lambda_1^\infty((\Lambda^\perp(\mathbf{A}))^*)$. Now, using Lemma B.1, we obtain for the lattice $\Lambda^\perp(\mathbf{A})$

$$\eta_\varepsilon(\Lambda^\perp(\mathbf{A})) \leq \frac{\log m}{\lambda_1^\infty((\Lambda^\perp(\mathbf{A}))^*)} = \frac{q \cdot \log m}{\lambda_1^\infty(\Lambda(\mathbf{A}))} \leq q/t \cdot \log m.$$

The claim follows. □

Lemma B.3 (Gaussian Samples [GPV08, Lemma 5.2, adapted])**.** *Assume the columns of $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ generate $\mathbb{Z}_q^n$, and let $\varepsilon = \varepsilon(m)$ be a negligible function. If $\chi \geq \eta_\varepsilon(\Lambda^\perp(\mathbf{A}))$, then the statistical distance between the following distributions is at most 2ε:*

$$\left\{(\mathbf{x}, \mathbf{A}\mathbf{x} \bmod q) : \mathbf{x} \leftarrow D_{\mathbb{Z},\chi}^m\right\} \quad \text{and} \quad \left\{(\mathbf{x}, \mathbf{z}) : \mathbf{z} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n, \mathbf{x} \leftarrow \mathbf{A}_\chi^{-1}(\mathbf{z})\right\}.$$

Lemma B.4 (Norm of $\mathbf{x}^\intercal\mathbf{G}$)**.** *Let $\mathbf{G} \in \mathbb{Z}_q^{n \times t}$ be the gadget matrix, where $t = n\lceil \log q \rceil$. Then for all $\mathbf{x} \neq \mathbf{0}^n \bmod q$, $\|\mathbf{x}^\intercal\mathbf{G} \bmod q\| \geq q/4$.*

Proof. Take any $\mathbf{x} \neq \mathbf{0}^n \bmod q$. Observe that the entries of $\mathbf{G}$ are the vectors $2^i \boldsymbol{\eta}_j \in \mathbb{Z}_q^n$ where $i \in \{0, \ldots, \lceil \log q \rceil - 1\}$, $j \in [n]$, and $\boldsymbol{\eta}_j$ is the j^{th} standard basis vector. Since $\mathbf{x} \neq \mathbf{0}^n \bmod q$, there exist indices i, j such that $2^i \mathbf{x}^\intercal \boldsymbol{\eta}_j \bmod q \geq q/4$. The claim follows. □

Lemma B.5 (Gadget Trapdoor $\implies$ Large $\lambda_1^\infty(\Lambda(\mathbf{A}))$)**.** *Let n, m, q be lattice parameters and*

$$\mathbf{A}\mathbf{R}_\mathbf{A} = \mathbf{G} \bmod q$$

where $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{R}_\mathbf{A} \in \mathbb{Z}_q^{m \times t}$. Then $\lambda_1^\infty(\Lambda(\mathbf{A})) \geq q/(4m\,\|\mathbf{R}_\mathbf{A}\|)$.

Proof. Take a non-zero vector $\mathbf{y} \in \Lambda(\mathbf{A})$. By definition, $\mathbf{y}^\intercal = \mathbf{x}^\intercal\mathbf{A} \bmod q$. Since we associate $\mathbb{Z}_q$ with $(-q/2, q/2]$, we have $\|\mathbf{x}^\intercal\mathbf{A} \bmod q\| \leq q/2$. This implies that $\|\mathbf{y}\| \geq \|\mathbf{x}^\intercal\mathbf{A} \bmod q\|$. We now consider two cases:

- Suppose $\mathbf{y} = \mathbf{0}^m \bmod q$. Since $\mathbf{y} \neq \mathbf{0}^m$, this means $\|\mathbf{y}\| \geq q/2$.

- Suppose $\mathbf{y} \neq \mathbf{0}^m \bmod q$. This in turn implies that $\mathbf{x} \neq \mathbf{0}^n \bmod q$. We will show that $\|\mathbf{x}^\intercal \mathbf{A} \bmod q\| \geq q/(4m \|\mathbf{R}_\mathbf{A}\|)$. Suppose otherwise (i.e., that $\|\mathbf{x}^\intercal \mathbf{A} \bmod q\| < q/(4m \|\mathbf{R}_\mathbf{A}\|)$). Then the following relation holds over the integers:

$$(\mathbf{x}^\intercal \mathbf{A} \bmod q) \cdot \mathbf{R}_\mathbf{A} = (\mathbf{x}^\intercal \mathbf{A} \mathbf{R}_\mathbf{A} \bmod q) \tag{B.1}$$

  This means $\|(\mathbf{x}^\intercal \mathbf{A} \bmod q) \cdot \mathbf{R}_\mathbf{A}\| = \|\mathbf{x}^\intercal \mathbf{A} \mathbf{R}_\mathbf{A} \bmod q\| = \|\mathbf{x}^\intercal \mathbf{G} \bmod q\| \geq q/4$ by Lemma B.4. Since Eq. (B.1) holds over the integers, this means $\|(\mathbf{x}^\intercal \mathbf{A} \bmod q) \cdot \mathbf{R}_\mathbf{A}\| \leq \|(\mathbf{x}^\intercal \mathbf{A} \bmod q)\| \cdot \|\mathbf{R}_\mathbf{A}\| \cdot m < q/4$. This leads to a contradiction, and we are done.

Thus, we conclude that $\|\mathbf{y}\| \geq \|\mathbf{x}^\intercal \mathbf{A} \bmod q\| \geq q/(4m \|\mathbf{R}_\mathbf{A}\|)$. □

Proof of Lemma 6.4. Suppose $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is a matrix with gadget trapdoor $\mathbf{R}_\mathbf{A}$. Then, by definition, we have $\mathbf{A}\mathbf{R}_\mathbf{A} = \mathbf{G}$. We first show that the columns of $\mathbf{A}$ generate $\mathbb{Z}_q^n$. For any $\mathbf{u} \in \mathbb{Z}_q^n$, observe that

$$\mathbf{A}\mathbf{R}_\mathbf{A}\mathbf{G}^{-1}(\mathbf{u}) = \mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{u}) \bmod q = \mathbf{u} \bmod q.$$

Hence, for every $\mathbf{u} \in \mathbb{Z}_q^n$, there exists a vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{x} = \mathbf{u} \bmod q$, and therefore $\mathbf{A}$ has rank n modulo q. Now, by Lemma B.5, $\lambda_1^\infty(\Lambda(\mathbf{A})) \geq q/(4m \|\mathbf{R}_\mathbf{A}\|)$. Then, by using Corollary B.2, we obtain that there exists a negligible function $\varepsilon(m) = \mathrm{negl}(m)$ such that $\eta_\varepsilon(\Lambda^\perp(\mathbf{A})) \leq 4m \|\mathbf{R}_\mathbf{A}\| \cdot \log m$. Since $\chi \geq 4m \|\mathbf{R}_\mathbf{A}\| \cdot \log m \geq \eta_\varepsilon(\Lambda^\perp(\mathbf{A}))$ and the columns of $\mathbf{A}$ span $\mathbb{Z}_q^n$, the conditions of Lemma B.3 are satisfied, implying that the statistical distance between

$$\left\{(\mathbf{x}, \mathbf{A}\mathbf{x} \bmod q) : \mathbf{x} \leftarrow D_{\mathbb{Z},\chi}^m\right\} \quad \text{and} \quad \left\{(\mathbf{x}, \mathbf{z}) : \mathbf{z} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n, \mathbf{x} \leftarrow \mathbf{A}_\chi^{-1}(\mathbf{z})\right\}$$

is at most 2ε. Since $\varepsilon(m)$ is negligible in m, the claim follows. □