A Generic Approach to Adaptively-Secure Broadcast Encryption in the Plain Model

Yao-Ching Hsieh[*] Brent Waters[†] David J. Wu[‡]

Abstract

Broadcast encryption allows a user to encrypt a message to N recipients with a ciphertext whose size scales sublinearly with N. The natural security notion for broadcast encryption is *adaptive* security which allows an adversary to choose the set of recipients *after* seeing the public parameters. Achieving adaptive security in broadcast encryption is challenging, and in the plain model, the primary technique is the celebrated dual-systems approach, which can be implemented over groups with bilinear maps. Unfortunately, it has been challenging to replicate the dual-systems approach in other settings (e.g., with lattices or witness encryption). Moreover, even if we focus on pairing-based constructions, the dual-systems framework critically relies on *decisional* (and source-group) assumptions. We do not have constructions of adaptively-secure broadcast encryption from *search* (or target-group) assumptions in the plain model.

Gentry and Waters (EUROCRYPT 2009) described a compiler that takes any *semi-statically-secure* broadcast encryption scheme and transforms it into an adaptively-secure scheme in the *random oracle* model. While semi-static security is easier to achieve and constructions are known from witness encryption as well as search (and target-group) assumptions on pairing groups, the transformed scheme relies on random oracles. In this work, we show that using *publicly-sampleable* projective PRGs, we can achieve adaptive security in the *plain model*. We then show how to build publicly-sampleable projective PRGs from many standard number-theoretic assumptions (e.g., CDH, LWE, RSA).

Our compiler yields the first adaptively-secure broadcast encryption scheme from *search* assumptions as well as the first such scheme from witness encryption in the plain model. We also obtain the first adaptively-secure pairing-based scheme in the plain model with $O_\lambda(N)$-size public keys and $O_\lambda(1)$-size ciphertexts (where $O_\lambda(\cdot)$ suppresses polynomial factors in the security parameter λ). Previous adaptively-secure pairing-based schemes in the plain model with $O_\lambda(1)$-size ciphertexts required $O_\lambda(N^2)$-size public keys.

1 Introduction

Broadcast encryption [FN93] allows a sender to encrypt a message to an arbitrary set of recipients $S \subseteq [N]$ with the property that any recipient $i \in S$ in the broadcast set can decrypt the encrypted message. On the other hand, even if all users outside S collude, they should not learn anything about the encrypted message. We say such schemes are fully collusion resistant. Finally, the size of the encrypted broadcast should be much smaller than the number of recipients $|S|$. A broadcast encryption scheme has optimal ciphertext size if the length of an encryption of a message μ is $|\mu| + \text{poly}(\lambda, \log |S|)$ bits, where λ is the security parameter.

In the three decades since the work of Fiat and Naor [FN93], there has been a long line of work studying the feasibility of broadcast encryption. Early works [NP00, NNL01, DF02, HS02, GST04] provide combinatoric constructions where the size of the ciphertext scales linearly with either the number of recipients or the complement of the set (i.e., the number of revoked users). The first fully collusion-resistant broadcast encryption scheme with short ciphertexts for all broadcast sets was the pairing-based scheme by Boneh, Gentry, and Waters [BGW05]. Subsequently, there have been many constructions from pairing-based assumptions [BW06, GW09, Wat09, GKSW10, CGW15, GKW18, Wee21, KMW23], new (non-standard) lattice assumptions [BV22, Wee22, Wee24, CW24, CHW25, Wee25], combinations of pairing-based and lattice-based assumptions [GQWW19, AY20, AWY20], multilinear maps [BS03, BW13, BWZ14], witness encryption [GVW19, FWW23], and indistinguishability obfuscation [BZ14].

[*]University of Washington, ychsieh@cs.washington.edu. Part of this work was done while visiting UT Austin.
[†]UT Austin and NTT Research, bwaters@cs.utexas.edu.
[‡]UT Austin, dwu4@cs.utexas.edu

Adaptive security. The natural security notion in broadcast encryption is *adaptive* security, which requires security to hold against adversaries that can choose the broadcast set (associated with the challenge ciphertext) *after* seeing the scheme parameters. Achieving adaptive security is challenging. The first construction of fully collusion-resistant broadcast encryption by Boneh, Gentry, and Waters [BGW05] considered a relaxed notion of *selective* security where the adversary is required to declare its challenge set $S \subseteq [N]$ at the beginning of the game (*before* seeing the scheme parameters). Many subsequent schemes constructing broadcast encryption from multilinear maps [BWZ14], indistinguishability obfuscation [BZ14], witness encryption [FWW23], or lattice-based assumptions [BV22, Wee22, Wee24, CW24, Wee25] only achieve selective security (or a slight strengthening called "semi-static security" [GW09] which we discuss below). Unlike identity-based encryption or attribute-based encryption, neither selective nor semi-static security imply adaptive security via complexity leveraging. This is because the standard approach of having the reduction algorithm guess the challenge set incurs a 2^N-loss in the reduction advantage. Correspondingly, this would inflate the security parameter by at least a factor of N, in which case, the size of the ciphertext becomes *linear* in the total number of users N.

The primary approach for constructing adaptively-secure broadcast encryption scheme is Waters' dual-system technique [Wat09], which provides a general template for realizing adaptive security for broadcast encryption (and other advanced encryption notions such as identity-based encryption and attribute-based encryption). In this setting, the security proof steps through a sequence of hybrid arguments where the challenge ciphertext and the secret keys for the broadcast encryption scheme are iteratively replaced by "semi-functional" variants. Once all of the secret keys and the challenge ciphertext are replaced by semi-functional versions, security holds unconditionally. This type of approach is taken in [Wat09, CGW15, Wee21, KMW23] to obtain adaptively-secure broadcast encryption from pairing-based assumptions. An earlier approach for constructing adaptively-secure broadcast encryption scheme by Boneh and Waters [BW06] showed how to use a traitor tracing scheme to implement a "dual-system-like" proof structure of switching out decryption keys one-by-one in the security game.

While the dual-system technique has been a successful paradigm for achieving adaptively-secure broadcast encryption, we have been unable to adapt these techniques to the lattice-based setting (or to constructions using witness encryption). Even if we consider pairing-based constructions, the schemes that rely on the dual-system methodology need to make *decisional* assumptions in the *source group*. This is needed to replace the ciphertext and secret keys with their (computationally indistinguishable) semi-functional analogs. In contrast, if we consider earlier selectively-secure pairing-based broadcast encryption schemes like [BGW05] or semi-statically-secure schemes such as [GW09], their security can be based on a *search* assumption[1] in the *target group*. Achieving adaptive security for broadcast encryption from search or target-group assumptions remains open.

The Genty-Waters approach. The work of Gentry and Waters [GW09] provides an alternative route to adaptive security. In their work, they first introduce the notion of semi-static security that is an intermediate notion between selective and adaptive security. In semi-static security, the adversary is required to commit to a superset $S^* \subseteq [N]$ at the beginning of the game and it is not allowed to request secret keys for any index $i \in S^*$. The adversary can choose its challenge set S to be any *subset* $S \subseteq S^*$ of the committed set. While semi-static security is stronger than selective security, it is still easier to achieve compared to adaptive security. The construction from [GW09] yields a semi-statically-secure broadcast encryption scheme from a computational target-group assumption. Similarly, constructions of broadcast encryption from indistinguishability obfuscation [BZ14] and witness encryption [FWW23] naturally satisfy semi-static security. The recent work of [CHW25] also shows how to construct a semi-statically-secure broadcast encryption scheme from succinct LWE in the random oracle model. The work of [GW09] then describes a generic compiler that takes any semi-statically secure broadcast encryption scheme and transforms it into an adaptively-secure scheme in the *random oracle model*. Thus, [GW09] provides a template for building adaptively-secure broadcast encryption from a broader range of cryptographic assumptions (by starting from a semi-statically-secure scheme), but has the drawback of needing to rely on random oracles.

Construction	$\lvert \mathsf{mpk} \rvert$	$\lvert \mathsf{sk} \rvert$	$\lvert \mathsf{ct} \rvert$	Assumption
[BW06]	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	subgroup decision
[Wat09]	N	N	1	2-Lin
[GKSW10]	$\sqrt{N}$	$\sqrt{N}$	N	2-Lin
[GKW18]	N^2	1	1	k-Lin ($k \geq 1$)
[Wee21]	$N^{1/3}$	$N^{1/3}$	$N^{1/3}$	bilateral k-Lin ($k \geq 2$)
This work + [GW09, §4]	N	1	1	decisional q-BDHE-sum
This work + [GW09, §3]*	N^2	1	1	search q-BDHE
This work + Construction 7.5	$N^{1+o(1)}$	1	1	search q-SC-BDHE
This work + [FWW23]	N	1	1	witness encryption + LWE

* The proof from [GW09] relies on the decisional q-BDHE assumption, but using hard-core predicates [GL89, HLR07], security can also be reduced to the search version of the same assumption.

Table 1: Comparison to *adaptively-secure* broadcast encryption schemes in the *plain* model. We measure the size of the master public key mpk, secret key sk, and ciphertext ct as a function of the number of users N. We suppress $\mathsf{poly}(\lambda, \log N)$ factors in all comparisons, where λ is a security parameter. For any constant $k \in \mathbb{N}$, the (bilateral) k-Lin assumption is a static decisional assumption over a prime-order pairing group. The parameter q indicates a q-type assumption where $q = \mathsf{poly}(\lambda, N)$. The q-bilinear Diffie-Hellman exponent (q-BDHE) assumption is a q-type assumption introduced in [BBG05]. The q-BDHE-sum and q-SC-BDHE refer to the sum variant (c.f., [GW09] and Assumption C.1) and the set-consistent variants of this assumption (c.f., [GLWW24] and Assumption 7.4), respectively. The constructions obtained in this work are through applying our compiler (Construction 5.1) to the listed semi-statically-secure broadcast encryption scheme.

1.1 Our Results

In this work, we revisit the Gentry-Waters compiler and show how to instantiate it in the *plain* model. Specifically, we show that we can replace the random oracle in the [GW09] construction with a (publicly-sampleable) projective pseudorandom generator (PRG) [ABI$^+$23] and prove adaptive security of the resulting construction in the plain model. While our construction relies on projective PRGs, our construction has a different syntax (i.e., requires a public-sampleability property) and relies on a different and incomparable set of security requirements than the application to computational secret sharing considered in [ABI$^+$23]; we refer to Section 2 for more details. We then show how to adapt the ideas underlying the projective PRG constructions from [ABI$^+$23] to obtain publicly-sampleable projective PRGs from a wide array of number-theoretic assumptions: the computational Diffie-Hellman (CDH) assumption (in pairing-free groups), the computational bilinear Diffie-Hellman (CBDH) assumption, the RSA assumption, as well as the learning with errors (LWE) assumption. Combining our new compiler with existing constructions of semi-statically-secure broadcast encryption, we immediately obtain several new adaptively-secure broadcast encryption schemes:

- **Security based on search assumptions.** Applying our compiler to the scheme from [GW09, §3], we obtain an adaptively-secure pairing-based broadcast encryption scheme from *search* assumptions in the plain model. All previous pairing-based approaches either needed decisional assumptions or random oracles.

- **A lattice-based instantiation via witness encryption.** Applying our compiler to the scheme from [FWW23, §4], we obtain an adaptively-secure broadcast encryption scheme from witness encryption together with the learning with errors (LWE) assumption. Previous schemes only achieved semi-static security. Combined with the witness encryption schemes from lattices [Tsa22, VWW22], we obtain the first adaptively-secure broadcast encryption from lattice assumptions in the plain model (specifically, the private-coin evasive LWE assumption [Wee22, Tsa22]).

[1]Technically, both [BGW05, GW09] prove security of their scheme using a decisional assumption in the target group, but we can replace these with the corresponding search assumption using hard-core predicates [GL89, HLR07].

	$\lvert\mathsf{crs}\rvert$	$\lvert\mathsf{pk}\rvert$	$\lvert\mathsf{sk}\rvert$	$\lvert\mathsf{ct}\rvert$	**Security**	**Assumption**
[WQZD10], [KMW23, §A]	N	N^2	1	1	selective	search q-BDHE*
[FWW23, §7] + [ZZGQ23]	N^2	N	1	1	selective	k-Lin ($k \geq 1$)†
[CW24]	N^2	N	1	1	selective	q-succinct LWE
[BZ14]	–	1	1	1	semi-static	$i\mathcal{O}$ + one-way functions
[FWW23, §4]	1	1	1	1	semi-static	witness encryption + LWE
[KMW23, §5]	N	N	1	1	semi-static	search q-BDHE*
[KMW23, §6]	N^2	N	1	1	adaptive	k-Lin ($k \geq 1$)
This work + [KMW23, §5]	N	N	1	1	adaptive	search q-BDHE
This work + [FWW23]	N	1	1	1	adaptive	witness encryption + LWE

* While the analysis in [KMW23, §5, §A] proves security from the decisional q-bilinear Diffie-Hellman exponent (BDHE) assumption, it is straightforward to modify the scheme to obtain security from the search version of the assumption (via hard-core predicates [GL89, HLR07]).

† The work of [FWW23] show a generic transformation from any registered attribute-based encryption (ABE) scheme [HLWW23] to a distributed broadcast encryption scheme. Here, we consider the instantiation using a pairing-based registered ABE scheme from the k-Lin assumption [ZZGQ23] (or alternatively, [AT24]). With q-type assumptions, we can reduce the public parameter size to $N^{1+o(1)}$ by applying the [FWW23] compiler to the registered ABE scheme from [GLWW24].

Table 2: Comparison to distributed broadcast encryption schemes in the *plain* model. We measure the size of the common reference string crs, the size of an individual user's public key pk, and the size of the secret key sk as a function of the number of users N. We suppress $\mathsf{poly}(\lambda, \log N)$ factors in all comparisons, where λ is a security parameter. The parameter q indicates a q-type assumption where $q = \mathsf{poly}(\lambda, N)$. The constructions obtained in this work are through applying our compiler (Construction A.2) to the listed semi-statically-secure broadcast encryption scheme.

- **Schemes with sub-quadratic public keys.** Applying our compiler to the scheme from [GW09, §4], we obtain a pairing-based broadcast encryption scheme in the plain model for N users where the public key contains $O(N)$ group elements, and the secret keys and ciphertext contain $O(1)$ group elements. Previous schemes with constant size ciphertexts [GKW18] had a quadratic-size public key (i.e., $\lvert\mathsf{mpk}\rvert = O(N^2)$). Our construction with linear-size public keys relies on a decisional assumption. In this work, we also give a new construction of a semi-statically-secure broadcast encryption scheme with a public key of nearly linear size (i.e., $N^{1+o(1)}$) and constant-size secret keys and ciphertexts where security is based on *search* assumptions. Coupled with our compiler, this yields an adaptively-secure broadcast encryption with the same efficiency (and security based on search assumptions).

We refer to Table 1 for a comparison of our new constructions with existing adaptively-secure broadcast encryption schemes in the plain model.

Application to distributed broadcast encryption. Distributed broadcast encryption [WQZD10, BZ14] is a trustless variant of broadcast encryption where instead of a central authority generating decryption keys for individual users, users instead sample their own public/secret key. In this model, there is a one-time setup procedure that outputs a common reference string (CRS) that users reference when generating their keys. Anyone can encrypt to an arbitrary set of public keys, and the resulting ciphertext must satisfy the usual succinctness requirement. The work of [KMW23] shows that the Gentry-Waters compiler can also be used to transform a semi-statically-secure distributed broadcast encryption scheme into an adaptively secure scheme in the random oracle model.

In this work, we show that our approach based on publicly-sampleable projective PRGs can also be used to upgrade any semi-statically-secure distributed broadcast encryption scheme into an adaptively-secure scheme. If we apply our construction to the recent pairing-based semi-statically-secure distributed broadcast encryption scheme from [KMW23], we obtain the first adaptively-secure construction with linear-size public parameters (and also the first scheme whose security only relies on search assumptions). Previous adaptively-secure schemes (in the plain model) required quadratic-size public parameters. Applying our compiler to [FWW23], we obtain the first adaptively-secure distributed broadcast encryption scheme from lattice assumptions.[2] We provide a comparison to previous constructions in Table 2.

2 Technical Overview

We first recall the syntax of a broadcast encryption scheme [FN93, BGW05]. Let N be the number of users in the system. Each user has a distinct index $i \in [N]$. Then, a broadcast encryption consists of the following algorithms:

- **Setup:** The setup algorithm generates the master public key mpk together with a master secret key msk (used to generate user secret keys).
- **Key generation:** The key-generation algorithm takes the master secret key msk and an index $i \in [N]$ and outputs the secret key sk_i for user i.
- **Encryption:** The encryption algorithm takes the master public key mpk, a set of users $S \subseteq [N]$, and a message μ, and outputs a ciphertext ct.
- **Decryption:** The decryption algorithm takes as input a ciphertext ct, the associated broadcast set $S \subseteq [N]$, and a secret key sk_i for some index $i \in S$, and outputs a message μ.

The correctness requirement says that if ct is an encryption of μ to a set S, then decrypting ct using any key sk_i for an index $i \in S$ should recover the message. The succinctness requirement says that the size of the ciphertext ct output by the encryption algorithm should be sublinear in the size of the broadcast set $|S|$.

Security for broadcast encryption. The starting point is the Gentry-Waters compiler [GW09] that generically transforms any *semi-statically-secure* broadcast encryption scheme into an adaptively-secure scheme in the random oracle model. We recall the definitions of adaptive security and semi-static security for broadcast encryption. We define two experiments, parameterized by a bit $b \in \{0, 1\}$.

- **Setup phase:** The challenger runs the setup algorithm to obtain the master public key mpk and the master secret key msk for the encryption scheme. The challenger gives mpk to the adversary.
- **Query phase:** The adversary can now (adaptively) request secret keys for users $i \in [N]$. On each query, the challenger responds with the secret key sk_i for user i.
- **Challenge phase:** Once the adversary is done making key-generation queries, it specifies a challenge set $S \subseteq [N]$ and a pair of challenge messages μ_0, μ_1. The requirement is that S does not contain any index $i \in [N]$ for which the adversary made a key-generation query. The adversary also specifies a pair of challenge messages μ_0, μ_1. The challenger responds with an encryption of μ_b (where $b \in \{0, 1\}$) to the set S.
- **Output phase:** The adversary then output a guess $b' \in \{0, 1\}$, which is the output of the experiment.

A broadcast encryption scheme is adaptively-secure if the output of the experiment when $b = 0$ is computationally indistinguishable from the output when $b = 1$. Next, semi-static security corresponds to the following relaxation on adaptive security:

- **Setup phase:** At the beginning of the setup phase (*before* seeing mpk), the adversary commits to a set $S^* \subseteq [N]$.
- **Query phase:** The adversary is no longer allowed to issue key-generation queries for any $i \in S^*$.
- **Challenge phase:** The challenge set S must be a subset of the committed set S^* (i.e., $S \subseteq S^*$).

[2] A concurrent work [CHW25] shows how to construct a semi-static distributed broadcast encryption scheme from the succinct LWE assumption in the random oracle model. They then apply the Gentry-Waters compiler [GW09] to obtain an adaptively-secure scheme in the random oracle model. This work gives a construction in the *plain* model by applying our compiler to the [FWW23] distributed broadcast encryption scheme based on witness encryption (and LWE).

In other words, in semi-static security, the adversary must commit to a *superset* of its eventual challenge set and it is not allowed to query for any keys in the committed set. The difference between semi-static security and selective security is the adversary does not have to commit to its *exact* challenge set S during the setup phase.

The Gentry-Waters compiler. We now recall the Gentry-Waters [GW09] compiler that transforms any semi-statically-secure broadcast encryption scheme into an adaptively-secure scheme in the random oracle model. We start by describing a simplified version of their approach that does *not* use random oracles, but has *long* ciphertexts:

- **Setup:** To construct an adaptively-secure secure scheme with N users, [GW09] instantiates the semi-statically-secure scheme with $2N$ users. We index the $2N$ users for the semi-statically-secure scheme by a pair (i, b) where $i \in [N]$ and $b \in \{0, 1\}$. The master public key is the master public key for the underlying semi-statically secure scheme. Let $\mathsf{sk}'_{i,b}$ denote the secret keys for user (i, b) for the underlying semi-statically-secure scheme.
- **Key-generation:** To generate a key for user i, the key-generation algorithm samples a bit $s_i \xleftarrow{\mathrm{R}} \{0, 1\}$. The key for user i is the pair $\mathsf{sk}_i = (s_i, \mathsf{sk}'_{i,s_i})$.
- **Encryption:** To encrypt a message μ with respect to a set $S \subseteq [N]$, the encryption oracle samples $t_i \xleftarrow{\mathrm{R}} \{0, 1\}$ for each $i \in S$. The encryption algorithm encrypts μ with respect to the set $S' = \{(i, t_i) : i \in S\}$ as well as $\bar{S}' = \{(i, 1 - t_i) : i \in S\}$ using the underlying semi-statically-secure scheme and obtains ciphertext $\mathsf{ct}'_0, \mathsf{ct}'_1$. The ciphertext is the triple $\mathsf{ct} = \big(\{(i, t_i)\}_{i \in S}, \mathsf{ct}'_0, \mathsf{ct}'_1\big)$.
- **Decryption:** If $i \in S$, then $(i, s_i) \in S$ or $(i, s_i) \in S'$, so user i can use $\mathsf{sk}_i = (s_i, \mathsf{sk}'_{i,s_i})$ to decrypt either ct'_0 or ct'_1 and correctness follows.

To argue security, the work of [GW09] first leverages semi-static security to switch ct'_0 from an encryption of μ_0 to an encryption of μ_1. Then, by an analogous argument, they switch ct'_1 from an encryption of μ_0 to an encryption of μ_1. This suffices to argue that an encryption of μ_0 is computationally indistinguishable from an encryption of μ_1. Here, we sketch the argument for switching ct'_0. Let $\mathcal{A}$ be an adversary for the adaptive security game.

- **Setup phase:** At the start of the game, the reduction algorithm (playing the semi-static security game) samples $s_i \xleftarrow{\mathrm{R}} \{0, 1\}$ for all $i \in [N]$ and commits to the set $S^* = \{(i, 1 - s_i) : i \in [N]\}$. The reduction algorithm forwards the master public key for the semi-statically-secure scheme to $\mathcal{A}$.
- **Key-generation phase:** Whenever algorithm $\mathcal{A}$ makes a key-generation query for an index $i \in [N]$, the reduction algorithm makes a key-generation query (i, s_i) to the semi-static challenger and receives a key sk_{i,s_i}. By construction $(i, s_i) \notin S^*$, so this is allowed. The reduction algorithm replies to $\mathcal{A}$ with $\mathsf{sk}_i = (s_i, \mathsf{sk}_{i,s_i})$.
- **Challenge phase:** When algorithm $\mathcal{A}$ makes a challenge query for a set $S \subseteq [N]$ and messages μ_0, μ_1, the reduction algorithm makes a challenge query on the set $S = \{(i, 1 - s_i) : i \in S\} \subseteq S^*$ and messages μ_0, μ_1 to its challenger and receives a ciphertext ct'_0. Finally, the reduction defines $t_i \coloneqq 1 - s_i$ for all $i \in S$ and constructs ct'_1 as in the real scheme (as an encryption of μ_0 for set $\{(i, 1 - t_i) : i \in S\}$). The reduction algorithm replies with the ciphertext $\mathsf{ct} = \big(\{(i, t_i)\}_{i \in S}, \mathsf{ct}'_0, \mathsf{ct}'_1\big)$.

By construction, the reduction algorithm is a valid semi-static adversary. Moreover, since it samples $s_i \xleftarrow{\mathrm{R}} \{0, 1\}$, the secret keys are also perfectly distributed. It suffices to consider the distribution of the random string t. Recall that in the adaptive security game, the adversary cannot ask for the key for any index $i \in S$ appearing in the challenge set. This means the value of $t_i = 1 - s_i$ is perfectly hidden from the view of the adversary, and so the distribution of t_i is independent and uniform, as required. We now highlight two important properties of this reduction:

- **Two-key approach.** Every user has two possible secret keys, but the key-generation algorithm only gives out the secret key for one of them (chosen at random). In the security analysis, the reduction algorithm knows one of the two keys for *each* user, which allows it to answer key-generation queries for every user. At the same time, the challenge ciphertext is encrypted to the set of keys the reduction algorithm does *not* know, which is essential for being able to invoke semi-static security.

- **Challenge ciphertext hides unused bits.** In the real scheme, the bit $s_i \in \{0,1\}$ associated with each secret key sk_i is uniform and independent. The same is true for the bits $t_i \in \{0,1\}$ in the challenge ciphertext. In the security analysis, the reduction algorithm samples a *single* string $s \xleftarrow{\mathrm{R}} \{0,1\}$ and uses s_i for the key-generation queries and sets $t_i = 1 - s_i$ in the challenge ciphertext. At first glance, this may appear to violate the independence of s_i and t_i. The key is that the reduction only gives out s_i for $i \notin S$ and $t_i = 1 - s_i$ for $i \in S$ (because the adversary cannot request keys for users in the challenge set). Since the adversary sees at most one of s_i or $t_i = 1 - s_i$ for each $i \in [N]$, the view of the adversary is correctly simulated. Thus, the reduction critically relies on the fact that the simulated challenge ciphertext *hides* s_i for all $i \notin S$. This was trivial to argue for the above construction with long ciphertexts, but will be important in our construction.

As described, the transformation yields a scheme with long ciphertexts (linear in the size of the broadcast set) because the string t in the ciphertext is $|S|$ bits long. The work of [GW09] leverages the random oracle to *compress* t. Namely, the ciphertext contains a (short) seed $\sigma \xleftarrow{\mathrm{R}} \{0,1\}^\rho$, and each bit t_i is obtained by computing $t_i \leftarrow \mathcal{H}(\sigma, i)$ where $\mathcal{H}$ is modeled as a random oracle (i.e., $\mathsf{ct} = (\sigma, \mathsf{ct}'_0, \mathsf{ct}'_1)$). In the security analysis, the reduction algorithm "programs" the random oracle to output $t_i = \mathcal{H}(\sigma, i) \coloneqq 1 - s_i$. Importantly, the values of $\mathcal{H}(\sigma, i)$ for $i \notin S$ are *not* programmed (they are uniform and independent of s_i). This ensures that the challenge ciphertext hides the value of s_i for $i \notin S$. This yields a construction with short ciphertexts.

Replacing the random oracle with a PRG. The Gentry-Waters compiler relies on the random oracle to *compress* an $|S|$-bit random string into a λ-bit string. If we want to avoid the random oracle, a natural approach is to replace it with a pseudorandom generator (PRG). For example, instead of computing $t_i \leftarrow \mathcal{H}(r, i)$, the encryption algorithm could instead sample a seed $\sigma \in \{0,1\}^\lambda$ for a PRG, compute $t = \mathsf{PRG}(\sigma) \in \{0,1\}^n$, and then encrypt according to the bits of t (corresponding to indices $i \in S$). This approach preserves correctness, so the question is security.

Consider the adaptation of the previous Gentry-Waters argument where the reduction algorithm samples $\sigma \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ in the setup phase and then sets $s = \mathsf{PRG}(\sigma)$. In the challenge phase, the reduction algorithm sets σ to be the random seed in the challenge ciphertext (i.e., $\mathsf{ct} = (\sigma, \mathsf{ct}'_0, \mathsf{ct}'_1)$). Since σ is a random seed, the marginal distribution of the challenge ciphertext is correctly simulated. However, this approach does not satisfy the second requirement described above; the challenge ciphertext no longer hides the value of s_i for $i \notin S$. Indeed, the string σ in the challenge ciphertext completely reveals s_i, and as a result, there is a clear correlation between the key-generation queries and the challenge ciphertext (that would not exist in the real scheme). Thus, the naïve reduction strategy is not sufficient.

Projective PRGs. To implement the Gentry-Waters proof strategy, we need a way to take the PRG seed σ (that determines the string $s = \mathsf{PRG}(\sigma)$) and constrain it to a new seed σ_S such that $\mathsf{PRG}(\sigma_S)$ agrees with S on all indices $i \in S$, and moreover, the bits s_i for $i \notin S$ look random even given σ_S. This is precisely the property satisfied by a projective PRG [ABI+23], a notion recently introduced in the study of succinct computational secret sharing. Importantly for our application to broadcast encryption, the length of the constrained seed $\hat{\sigma}$ must be sublinear in the size of S.[3] Formally, a projective PRG with output length N consists of three algorithms:

- $\mathsf{Setup}(1^\lambda, 1^N) \to (\mathsf{pp}, \sigma)$: The setup algorithm takes as input the security parameter λ and the PRG output length N and outputs the set of public parameters pp along with a PRG seed σ.
- $\mathsf{Project}(\mathsf{pp}, \sigma, S) \to \hat{\sigma}_S$: The project algorithm takes the public parameters pp, the seed σ and a set $S \subseteq [N]$ and outputs a projected seed $\hat{\sigma}_S$. We require that the size of the projected seed $\hat{\sigma}_S$ to be sublinear in $|S|$.
- $\mathsf{Eval}(\mathsf{pp}, \hat{\sigma}_S, S, i) \to t_i$: The evaluation algorithm takes the public parameters pp, a projected seed $\hat{\sigma}_S$, the associated set $S \subseteq [N]$, and an index $i \in S$, and outputs the bit $t_i \in \{0,1\}$ at index i.

For a seed σ, we define the PRG output to be the string $s = \mathsf{PRG}(\sigma) \in \{0,1\}^N$ where $s_i = \mathsf{Eval}(\mathsf{pp}, \hat{\sigma}_{[N]}, [N], i)$ and $\hat{\sigma}_{[N]} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [N])$ is the seed projected onto the full output space $[N]$. Then, we require the projective PRG to satisfy the following properties:

[3]Constrained pseudorandom functions (PRFs) [BW13] are an analog of projective PRGs for the setting of PRFs. Unlike projective PRGs, these do not have a *succinctness* requirement.

- **Correctness:** If we project a seed σ onto a set S, the projected seed should evaluate to the same value as $\mathsf{PRG}(\sigma)$ on all inputs $i \in S$. Namely, if $\hat{\sigma}_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, S)$, then $\mathsf{Eval}(\hat{\sigma}_S, S, i) = \mathsf{Eval}(\hat{\sigma}_{[N]}, [N], i)$ for all $i \in S$.
- **Adaptive pseudorandomness:** Given a projected seed $\hat{\sigma}_S$ for some set S, the bits s_i for $i \notin S$ remain pseudorandom where $s = \mathsf{PRG}(\sigma)$. For our application to adaptively-secure broadcast encryption, we require pseudorandomness to hold against an *adaptive* adversary that can choose the set S after seeing arbitrary bits of s (so long as the challenge set S excludes such bits).

Suppose we now substitute a projective PRG for the random oracle in the Gentry-Waters construction. In the proof, the reduction algorithm would sample a PRG seed $\sigma \xleftarrow{\text{R}} \{0,1\}^\lambda$ and derive the string $s \leftarrow \mathsf{PRG}(\sigma)$. The challenge ciphertext would in turn contain a seed $\hat{\sigma}_S$ projected to the set S. This ensures that $\mathsf{Eval}(\mathsf{pp}, \hat{\sigma}_S, S, i)$ agrees with s_i on $i \in S$ and that s_i is pseudorandom even given $\hat{\sigma}_S$. This satisfies the two key requirements needed to carry out the Gentry-Waters proof strategy *without* random oracles.

Publicly-sampleable projective PRGs. Substituting a projective PRG introduces a new wrinkle into our construction. Namely, each ciphertext in the (transformed) broadcast encryption scheme contains a seed for a projective PRG (constrained to S). In our current abstraction, projective PRGs are defined with respect to a set of (long) public parameters pp, and the seed σ is tied to the choice of public parameters (this is necessary to support projection). In these constructions, we cannot resample a seed independently of the public parameters, and since the public parameters are long, we also cannot include a fresh set of public parameters as part of the ciphertext. Of course, we also cannot give out the seed for the projective PRG as part of the public parameters either. Thus, it is unclear how to support *public* encryption (which needs the ability to sample a constrained seed for the projective PRG). To resolve this problem, we augment the projective PRG with an additional public sampling algorithm:

- $\mathsf{Samp}(\mathsf{pp}, S) \to \hat{\sigma}_S$: The sampling algorithm takes as input the public parameters pp and a set S and outputs a "simulated" seed $\tilde{\sigma}_S$.

Next, we require the projective PRG to satisfy a sampling indistinguishability property:

- **Sampling indistinguishability:** For all sets $S \subseteq [N]$, the simulated seed output by the public sampling algorithm $\mathsf{Samp}(\mathsf{pp}, S)$ be computationally indistinguishable from the projected seed output by $\mathsf{Project}(\mathsf{pp}, \sigma, S)$. In the formal distinguishing experiment, the adversary sees the public parameters pp and either a simulated seed or a projected seed. Notably, it does *not* observe any output bits of the PRG.

In the construction, the (honest) encryption algorithm samples a seed using $\mathsf{Samp}(\mathsf{pp}, S)$ while the reduction constructs it using the projection algorithm. This allows us to implement the Gentry-Waters proof strategy without random oracles. We provide the formal definition of publicly-sampleable projective PRGs in Section 4 and our adaptation of the Gentry-Waters compiler in Section 5. An analogous compiler can also be used to upgrade any semi-statically-secure distributed broadcast encryption scheme [WQZD10, BZ14] into an adaptively secure distributed broadcast encryption scheme. We describe this compiler in Appendix A.

2.1 Constructing Publicly-Sampleable Projective PRGs.

The work of [ABI+23] shows how to construct projective PRGs with different efficiency properties from the decisional Diffie-Hellman (DDH), decisional bilinear Diffie-Hellman (DBDH), RSA, and the learning with errors (LWE) assumptions. The existing constructions do not provide an explicit public sampling algorithm nor do they satisfy the new security notions we consider in this work (adaptive pseudorandomness and sampling indistinguishability). However, it is possible to extend the existing constructions to satisfy our functionality and security requirements. We start with a quick summary of our main constructions:

- **Computational Diffie-Hellman:** We start by showing how to construct a publicly-sampleable projective PRG from the computational Diffie-Hellman assumption in a pairing-free group. The scheme has quadratic-size public parameters (measured as a function of the output length). Our construction is a direct adaptation of the

DDH-based construction from [ABI+23, §3.4.1], except we introduce a secret shift to support public sampling and also show that using hard-core predicates, we can base security on a *search* assumption rather than a decisional assumption. The latter distinction is important for achieving the first adaptively-secure broadcast encryption schemes from *search* assumptions.

- **Computational bilinear Diffie-Hellman:** Next, we show how a technique by Boyen and Waters [BW10] can be used to reduce the public parameter size of the CDH construction from quadratic to linear when working over a pairing group. This yields the first projective PRG from bilinear maps with linear-size public parameters; the pairing-based construction from [ABI+23, §3.4.2] still requires *quadratic-size* public parameters.

- **Learning with errors:** We show how to adapt the LWE-based projective PRG from [ABI+23, Appendix A] to obtain a publicly-sampleable projective PRG with linear-size public parameters. The main difference is again introducing a secret shift to achieve the public sampling property. This scheme has linear-size public parameters.

- **RSA:** Finally, we show that the RSA-based projective PRG from [ABI+23, §3.2] is also publicly-sampleable and yields a publicly-sampleable projective PRG with linear-size public parameters (and can plausibly be made constant-size; see Remark B.9).

To summarize, publicly-sampleable projective PRGs can be realized from most standard number-theoretic assumptions. Combined with our new compiler, this means building a semi-statically-secure broadcast encryption from any of these assumptions immediately implies an adaptively-secure broadcast encryption scheme. We now provide a more detailed overview of our constructions.

A construction from CDH. Our first construction is from the *computational* Diffie-Hellman problem where the public parameters are *quadratic* in the output length of the PRG. Our construction is an adaptation of the projective PRG scheme based on DDH from [ABI+23, §3.4.1]. One of the objectives in this work is to realize adaptive broadcast encryption from *search* assumptions, which is the reason we focus on CDH rather than DDH. Note that the [ABI+23] construction from DDH can easily be adapted to a construction from CDH (but the scheme does not support public sampling). We work over a group $\mathbb{G}$ of prime order p. Throughout, we use implicit notation to represent group elements [EHK+13]: namely, for $x \in \mathbb{Z}_p$, we write $[x]_{\mathbb{G}} := g^x$.

- **Setup:** The setup algorithm first samples random exponents $a_i, s_i \xleftarrow{\text{R}} \mathbb{Z}_p$ for all $i \in [N]$ along with a blinding factor $\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$. The public parameters consist of the group elements $[a_i]_{\mathbb{G}}, [s_i]_{\mathbb{G}}$ for all $i \in [N]$ along with the cross-terms $[a_i s_j]_{\mathbb{G}}$ for all $j \neq i$. The seed consists of the exponents $\sigma = (\alpha, s_1, \ldots, s_N)$.

- **Evaluation:** For each $i \in [N]$, we define the i^{th} bit t_i of the PRG output to be a hard-core predicate hc applied to a shifted diagonal term $[a_i(\alpha + s_i)]_{\mathbb{G}}$: namely, $t_i = \text{hc}([a_i(\alpha + s_i)]_{\mathbb{G}})$.

- **Projection:** The projected seed $\hat{\sigma}_S$ for a set S is $\hat{\sigma}_S := \alpha + \sum_{i \in S} s_i \in \mathbb{Z}_p$. In combination with the terms $[a_i]_{\mathbb{G}}$ and $[a_i s_j]_{\mathbb{G}}$ for $j \neq i$ in the public parameters, this suffices to compute the output bit $t_i = \text{hc}([a_i(\alpha + s_i)]_{\mathbb{G}})$:

$$\hat{\sigma}_S \cdot [a_i]_{\mathbb{G}} - \sum_{j \in S \setminus \{i\}} [a_i s_j]_{\mathbb{G}} = [\alpha a_i]_{\mathbb{G}} + [s_i a_i]_{\mathbb{G}} + \sum_{j \in S \setminus \{i\}} [a_i s_j]_{\mathbb{G}} - \sum_{j \in S \setminus \{i\}} [a_i s_j]_{\mathbb{G}} = [a_i(\alpha + s_i)]_{\mathbb{G}}.$$

- **Public sampling:** The simulated key for any set S is a uniform random field element $\tilde{\sigma}_S \xleftarrow{\text{R}} \mathbb{Z}_p$.

We now check that this satisfies our security requirements:

- **Adaptive pseudorandomness:** Under CDH, it is easy to show that $[a_i s_i]_{\mathbb{G}}$ is computationally unpredictable given the public parameters $[a_i]_{\mathbb{G}}$, $[s_i]_{\mathbb{G}}$, and $[a_i s_j]_{\mathbb{G}}$ for $i \neq j$ and the projected seed $\hat{\sigma}_i = \alpha + \sum_{j \in S} s_j$. In particular, the reduction algorithm gets $[a_i]_{\mathbb{G}}$ and $[s_i]_{\mathbb{G}}$ from the CDH challenge and picks all other exponents itself. Importantly, we only require unpredictability when $i \notin S$, which allows the reduction to simulate $\hat{\sigma}$. Since $[a_i s_i]_{\mathbb{G}}$ is computationally unpredictable, the same holds for $[a_i(\alpha + s_i)]_{\mathbb{G}}$. Finally, pseudorandomness follows from the security of the hard-core predicate [GL89, HLR07] (see Theorem 3.2).

- **Sampling indistinguishability:** Observe that the public parameters pp are *independent* of α. Thus, over the randomness of $\alpha \xleftarrow{R} \mathbb{Z}_p$, for any set S, the distribution of a projected seed $\hat{\sigma}_S$ is a uniform random field element. This is the same distribution as the $\tilde{\sigma}_S$ output by the public sampling procedure.

The difference between our construction from the DDH-based construction from [ABI+23, §3.4.1] is the extra offset α we introduce to support public sampling (and sampling indistinguishability) as well as the use of the hard-core predicate to base security on CDH rather than DDH. Without the blinding factor α, one can use the public parameters to test whether a projected seed is consistent with the public parameters or not. Namely, to check if a projected seed σ_S for a set S is a valid projected seed, one simply takes an index $j \notin S$ and checks whether $[a_j]_{\mathbb{G}} \cdot \sigma_S = \sum_{i \in S} [a_j s_i]_{\mathbb{G}}$, which holds whenever $\sigma_S = \sum_{i \in S} s_i$ is a valid projected seed. We provide the full details in Section 6.1.

Reducing the public parameter size using pairings. A disadvantage of the above CDH construction is the size of the public parameters is quadratic in the output length. In the context of broadcast encryption, the PRG output length corresponds to the number of users in the system, so using our CDH-based projective PRG to boost a semi-statically-secure scheme to an adaptively-secure scheme would lead to a scheme with quadratic-size public parameters. A natural question then is whether we can reduce this overhead. Here, we show that the approach of Boyen and Waters [BW10] can be directly applied to obtain a publicly-sampleable projective PRG with a *linear-size* public parameters. This gives the first projective PRG with linear-size public parameters from standard pairing assumptions. Previously, the work of [ABI+23] showed how to use pairings to obtain a "reusable" projective PRG, but still with quadratic-size public parameters. Our application does not rely on reusability, and thus, we are able to achieve significant compression.

Let $e\colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ be an efficiently-computable, non-degenerate (symmetric) bilinear map, where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order p. We represent elements of $\mathbb{G}$ and $\mathbb{G}_T$ implicitly as $[x]_{\mathbb{G}}$ and $[x]_{\mathbb{G}_T}$, respectively. Boyen and Waters [BW10] show how to use a bilinear map to compress $[a_i s_j]_{\mathbb{G}_T}$ for all $i \neq j$ with $O(N)$ elements in $\mathbb{G}$, while simultaneously ensuring that the non-cross-terms $[a_i s_i]_{\mathbb{G}_T}$ are hidden.[4] Very briefly, the Boyen-Waters approach is as follows:

- The public parameters contain encodings $[a_i]_{\mathbb{G}}, [s_i]_{\mathbb{G}}, [a_i(\gamma i + \delta)]_{\mathbb{G}}, [s_i(\gamma i + \delta)]_{\mathbb{G}}$ for all $i \in [N]$, where $\gamma, \delta \xleftarrow{R} \mathbb{Z}_p$ are random blinding factors.

- Observe now that the public parameters allow one to compute $[\gamma a_i s_j]_{\mathbb{G}_T}$ for all $i \neq j$, but *not* terms of the form $[\gamma a_i s_i]_{\mathbb{G}_T}$. To see this, first observe

$$[a_i]_{\mathbb{G}} \cdot [s_j(\gamma j + \delta)]_{\mathbb{G}} - [s_j]_{\mathbb{G}} \cdot [a_i(\gamma i + \delta)]_{\mathbb{G}} = [\gamma a_i s_j (j - i)]_{\mathbb{G}_T}.$$

  When $i \neq j$, the user can recover $[\gamma a_i s_j]_{\mathbb{G}_T} = [\gamma a_i s_j (j-i)]_{\mathbb{G}_T} \cdot (j-i)^{-1}$. However, when $i = j$, then the above expression yields the identity element, and the user is unable to compute $[\gamma a_i s_i]_{\mathbb{G}_T}$.

We obtain a publicly-sampleable projective PRG with linear-size public parameters by using the above technique to compress the public parameters from our basic CDH construction described above (where the evaluations now happen in the *target* group $\mathbb{G}_T$). Hardness in turn relies on the computational bilinear Diffie-Hellman assumption (i.e., given random elements $[u]_{\mathbb{G}}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}}$, it is hard to compute $[uvw]_{\mathbb{G}_T}$). We provide the full details in Section 6.2. As noted above, our construction gives a pairing-based projective PRG with linear-size public parameters, which improves upon the construction from [ABI+23] which needed quadratic-size public parameters. Note though that the construction from [ABI+23] satisfies an additional reusability property that is important for their applications, which our construction does not satisfy; our application to broadcast encryption does not need reusability.

A construction from LWE. We can also build a publicly-sampleable projective PRG from the learning with errors (LWE) assumption [Reg05]. This construction can be viewed as a direct translation of the construction from CDH, and follows the analogous lattice-based instantiation proposed in [ABI+23]. Essentially, we replace the exponents $[a_i]_{\mathbb{G}}$ and $[s_i]_{\mathbb{G}}$ in the public parameters with random vector $\mathbf{a}_i, \mathbf{s}_i \xleftarrow{R} \mathbb{Z}_q^n$. The cross terms $[a_i s_j]_{\mathbb{G}}$ then consist of LWE samples $\mathbf{s}_j^{\mathsf{T}} \mathbf{a}_i + e_{i,j}$, where $e_{i,j}$ is a small noise term. Under the plain LWE assumption, given $\mathbf{a}_1, \ldots, \mathbf{a}_N$ and $\mathbf{s}_j \mathbf{a}_i^{\mathsf{T}} + e_{i,j}$, the value of the (noise-free) non-cross-terms $\mathbf{s}_i \mathbf{a}_i$ are hidden. To support public sampling and sampling

[4]If we did not require the non-cross-terms to be hidden, then we could simply publish encodings of $[a_i]_{\mathbb{G}}$ and $[s_i]_{\mathbb{G}}$ for all $i \in [N]$, and use the pairing to compute $[a_i s_j]_{\mathbb{G}_T} := [a_i]_{\mathbb{G}} \cdot [s_j]_{\mathbb{G}}$. The challenge is to reveal the cross-terms while hiding the non-cross-terms.

indistinguishability, we again introduce a random shift to the projected key: $\hat{\sigma}_S := \mathbf{r} + \sum_{i \in S} \mathbf{s}_i$, and define the evaluation at i to be $\lfloor (\mathbf{r} + \mathbf{s}_i)^{\intercal} \mathbf{a}_i \rceil$, where $\lfloor \cdot \rceil$ denotes the rounding operation (i.e., $\lfloor x \rceil$ outputs 0 if $|x| < q/4$ and 1 otherwise). This yields a construction with quadratic-size public parameters.

To obtain a scheme with linear-size public parameters, we can rely on the same technique from [ABI+23] based on key-homomorphic puncturable PRFs. Namely, instead of giving our $\mathbf{s}_j^{\intercal} \mathbf{a}_i + e_{i,j}$ for all $j \neq i$, the [ABI+23] shows that we can instead give out a "punctured key" $\tilde{\mathbf{s}}_j$ which can be used to compute $\mathbf{s}_j^{\intercal} \mathbf{a}_i + e_{i,j}$ on all $i \neq j$. This approach can be concretely instantiated using the Brakerski-Vaikuntanathan [BV15] key-homomorphic constrained PRF. We provide the formal details in Section 6.3.

A construction from RSA. Finally, in Appendix B, we also show that the projective PRG scheme based on RSA from [ABI+23] satisfies our public-sampleability properties essentially with only a few syntactic modifications. However, because the security properties we need in our work do not follow as a black-box from existing definitions, we include a formal proof of our security requirements in Appendix B.

2.2 Semi-Statically-Secure Broadcast Encryption with Short Public Parameters

As a final contribution of this work, we also show how to construct a semi-statically-secure broadcast encryption with nearly-linear-size public parameters (i.e., public parameters of size $N^{1+o(1)}$) and constant-size secret keys and ciphertext from search assumptions. Previous pairing-based constructions of semi-statically-secure broadcast encryption with constant-size secret keys either needed quadratic-size public parameters [GW09, §3] or relied on decisional assumptions [GW09, §4].[5] The starting point of our construction is the construction from [GW09, Section 3], which has an $O(N^2)$-sized public key. Our construction proceeds as follows:

- **Security from search assumption:** The original construction [GW09, §3] relies on the decisional q-bilinear Diffie-Hellman exponent (q-BDHE) assumption. This is a target group assumption that essentially asserts the pseudorandomness of an element $Z \in \mathbb{G}_T$ in the target group. In the construction, Z is used to blind the message. Instead of relying on the pseudorandomness of Z, we could alternatively rely on the *unpredictability* of Z and blind the message using a hard-core predicate on Z (e.g., apply the Goldreich-Levin extractor to Z to derive a pseudorandom pad [GL89, HLR07]). This in turn allows us to base security on a search assumption rather than a decisional assumption. Note that a similar approach of replacing a decisional assumption with a search assumption does *not* seem applicable to adaptively-secure constructions based on the dual-systems methodology [Wat09, CGW15, Wee21, KMW23]. A dual-systems proof operates by changing the distribution of the challenge ciphertext and the secret keys in the security proof, and these changes rely on decisional assumptions in a more fundamental manner.

- **Reducing the CRS size:** The quadratic-size public key in [GW09, §3] is due to the fact that the master public key contains cross terms $h_i^{r_j}$ for all $i \neq j \in [N]$. These cross terms are needed to ensure decryption. In this setting, we are not able to rely on the earlier cross-term compression approach of [BW10], because correctness requires that the cross terms be given out in the *base* group rather than the target group. However, we are able to apply the combinatoric approaches based on progression-free sets [ET36] from [Lip12, GLWW24]. This allows us to reduce the CRS size from quadratic to nearly linear ($N^{1+o(1)}$). Specifically, the key insight in the [GLWW24] approach (in the context of reducing the CRS size in registered ABE schemes) is to choose the values h_i and r_j in a *correlated* manner such there are many pairs of indices $(i, j) \neq (k, \ell)$ that share a common cross-term (i.e., $h_i^{r_j} = h_k^{r_\ell}$). We show in Section 7 that a similar technique can be applied to the Gentry-Waters broadcast encryption scheme to obtain a scheme with a nearly-linear-size public key.

Combining this semi-statically-secure broadcast encryption scheme with our publicly-sampleable projective PRG from the computational bilinear Diffie-Hellman assumption, we obtain the first adaptively-secure broadcast encryption

[5]Technically, [GW09, §4] constructs an adaptively-secure identity-based broadcast encryption scheme in the *random oracle model*, but they note that a variant of their construction gives a semi-statically-secure broadcast encryption scheme with linear-size public keys in the plain model. For completeness, we provide an explicit description of this scheme in Appendix C. If we apply our compiler to this construction, we obtain an adaptively-secure broadcast encryption with linear-size public keys in the *plain model*; see Table 1.

with a nearly-linear-size public key and constant-size secret keys and ciphertexts from a search q-type assumption on bilinear groups.

3 Preliminaries

Throughout this work, we write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n] := \{1, \ldots, n\}$. For positive integers $a, b \in \mathbb{N}$ we write $[a, b] := \{a, a+1, \ldots, b\}$. For a positive integer $p \in \mathbb{N}$, we write $\mathbb{Z}_p$ to denote the ring of integers modulo p. We write $\mathsf{poly}(\lambda)$ to denote a fixed polynomial in λ. We write $\mathsf{negl}(\lambda)$ to denote a function that is negligible in λ (i.e., a function that is $o(\lambda^{-c})$ for all $c \in \mathbb{N}$). We say an event occurs with overwhelming probability if the probability of its complement occurring is negligible. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. For two ensembles of distributions $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_2 = \{\mathcal{D}_{2,\lambda}\}_{\lambda \in \mathbb{N}}$ indexed by a security parameter, we say they are computationally indistinguishable if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr[\mathcal{A}(1^\lambda, x) = 1 : x \leftarrow \mathcal{D}_{1,\lambda}] - \Pr[\mathcal{A}(1^\lambda, x) = 1 : x \leftarrow \mathcal{D}_{2,\lambda}]\right| = \mathsf{negl}(\lambda).$$

We say they are statistically indistinguishable if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, the statistical distance between them is $\mathsf{negl}(\lambda)$.

Goldreich-Levin hardcore bit. In this work, we rely on the classic Goldreich-Levin hardcore bit [GL89] to achieve security under search assumptions. Here, we state a formulation from [HLR07] that applies to any computationally unpredictable random variable.

Definition 3.1 (Computational Unpredictablility)**.** Let $(X, Y) = \{(X_\lambda, Y_\lambda)\}_{\lambda \in \mathbb{N}}$ be an ensemble of joint distributions over pairs of values. We say that X is computationally unpredictable given Y if for all efficient (and possibly non-uniform) adversaries $\mathcal{A}$, there exist a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[\mathcal{A}(1^\lambda, y) = x : (x, y) \leftarrow (X_\lambda, Y_\lambda)] = \mathsf{negl}(\lambda).$$

Theorem 3.2 (Goldreich-Levin [GL89, HLR07], adapted)**.** *Let $(X, Y) = \{(X_\lambda, Y_\lambda)\}_{\lambda \in \mathbb{N}}$ be an ensemble of joint distributions over pairs of values, where the support of X_λ is $\{0,1\}^{\rho(\lambda)}$. Let $\mathsf{hc}(x, r) := \langle x, r \rangle$ be the Goldreich-Levin extractor. Suppose that X is computationally unpredictable given Y. Then, for all efficient (and possibly non-uniform) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr\left[\mathcal{A}(1^\lambda, r, y, \mathsf{hc}(x, r)) = 1 : \begin{array}{c} (x, y) \leftarrow (X_\lambda, Y_\lambda) \\ r \xleftarrow{\mathrm{R}} \{0,1\}^{\rho(\lambda)} \end{array}\right] - \Pr\left[\mathcal{A}(1^\lambda, r, y, b) = 1 : \begin{array}{c} (x, y) \leftarrow (X_\lambda, Y_\lambda) \\ r \xleftarrow{\mathrm{R}} \{0,1\}^{\rho(\lambda)}, b \xleftarrow{\mathrm{R}} \{0,1\} \end{array}\right]\right| = \mathsf{negl}(\lambda).$$

3.1 Broadcast Encryption

We now recall the formal definition of a broadcast encryption scheme [FN93].

Definition 3.3 (Broadcast Encryption)**.** A broadcast encryption scheme is a tuple of efficient algorithms $\Pi_{\mathsf{BE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^N) \to (\mathsf{mpk}, \mathsf{msk})$: On input the security parameter λ and the number of users N, the setup algorithm outputs a master public key mpk and a master secret key msk.
- $\mathsf{KeyGen}(\mathsf{msk}, i) \to \mathsf{sk}_i$: On input the master secret key msk and an index $i \in [N]$, the key-generation algorithm outputs a secret key sk_i.
- $\mathsf{Enc}(\mathsf{mpk}, S, \mu) \to \mathsf{ct}$: On input the master public key mpk, a set $S \subseteq [N]$, and a message $\mu \in \{0,1\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{mpk}, S, \mathsf{sk}_i, \mathsf{ct}) \to \mu$: On input the master public key mpk, a set of recipients $S \subseteq [N]$, a secret key sk_i, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \{0,1\}$.

We require that Π_{BE} satisfy the following properties:

- **Correctness:** For all $\lambda, N \in \mathbb{N}$, all sets $S \subseteq [N]$, all indices $i \in S$, all messages $\mu \in \{0,1\}$, all $(\mathsf{mpk}, \mathsf{msk})$ in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, and all secret keys sk_i in the support of $\mathsf{KeyGen}(\mathsf{msk}, i)$, we have
$$\Pr[\mathsf{Dec}(\mathsf{mpk}, S, \mathsf{sk}_i, \mathsf{Enc}(\mathsf{mpk}, S, \mu)) = \mu] = 1.$$
- **Adaptive security:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $\beta \in \{0,1\}$, we define the adaptive-security experiment $\mathsf{EXP}_{\mathsf{BE}}^{(\beta)}(1^\lambda, \mathcal{A})$:
  - **Setup:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the number of users 1^N. The challenger computes $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and gives mpk to $\mathcal{A}$.
  - **Key-generation queries:** Algorithm $\mathcal{A}$ can now make (adaptive) key-generation queries. On each query, algorithm $\mathcal{A}$ specifies an index $i \in [N]$ and the challenger responds with $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$.
  - **Challenge query:** After $\mathcal{A}$ finishes making evaluation queries, it outputs a set $S \subseteq [N] \setminus I$, where $I \subseteq [N]$ is the set of indices on which algorithm $\mathcal{A}$ made a key-generation query. The challenger computes $\mathsf{ct}_\beta \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, \beta)$ and replies to algorithm $\mathcal{A}$ with ct_β.
  - **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.

  We say that Π_{BE} satisfies adaptive security if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\left|\Pr[\mathsf{EXP}_{\mathsf{BE}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{BE}}^{(1)}(1^\lambda, \mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$
- **Succinctness:** For all $\lambda, N \in \mathbb{N}$, all key pairs $(\mathsf{mpk}, \mathsf{msk})$ in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all sets $S \subseteq [N]$, all bits $\mu \in \{0,1\}$, and all ciphertexts ct in the support of $\mathsf{Enc}(\mathsf{pp}, \mathsf{ct}, \mu)$, it holds that $|\mathsf{ct}| \le o(|S|) \cdot \mathsf{poly}(\lambda, \log N)$.

Semi-static security. Next, we recall the notion of semi-static security for broadcast encryption introduced by Gentry and Waters [GW09]. At a high-level, in the semi-static security game, the adversary has to pre-commit to a set $S^* \subseteq [N]$ of users. During the security game, the adversary is *not* allowed to make key-generation queries on any index $i \in S^*$. When the adversary makes its challenge query, it can specify *any* set $S \subseteq S^*$. The main difference between semi-static and *selective* security is that in selective security, the challenge query is on the committed set S^* whereas in semi-static security, the challenge ciphertext can be encrypted to any *subset* of the committed set S^*. Gentry and Waters showed how to generically compile any semi-statically-secure broadcast encryption scheme into an adaptively secure scheme in the random oracle model. On the contrary, we do not know of any generic compiler from a selectively-secure broadcast encryption scheme into an adaptively-secure one. We recall the definition below:

Definition 3.4 (Semi-Static Security for Broadcast Encryption [GW09]**).** Let $\Pi_{\mathsf{BE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a broadcast encryption scheme. For a security parameter λ, an adversary $\mathcal{A}$, and a bit $\beta \in \{0,1\}$, we define the semi-static security experiment $\mathsf{EXP}_{\mathsf{SSBE}}^{(\beta)}(1^\lambda, \mathcal{A})$ as follows:

- **Setup:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the number of users 1^N together with a set $S^* \subseteq [N]$. The challenger computes $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and gives mpk to $\mathcal{A}$.
- **Key-generation queries:** Algorithm $\mathcal{A}$ can now make (adaptive) key-generation queries. On each query, algorithm $\mathcal{A}$ specifies an index $i \in [N] \setminus S^*$ and the challenger responds with $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$.
- **Challenge query:** After $\mathcal{A}$ finishes making evaluation queries, it outputs a set $S \subseteq S^*$. The challenger computes $\mathsf{ct}_\beta \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, \beta)$ and replies to algorithm $\mathcal{A}$ with ct_β.
- **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.

We say that Π_{BE} satisfies semi-static security if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\left|\Pr[\mathsf{EXP}_{\mathsf{SSBE}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{SSBE}}^{(1)}(1^\lambda, \mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

4 Publicly-Sampleable Projective PRGs

In this section, we introduce the notion of a publicly-sampleable projective pseudorandom generator. While our notion shares a similar syntax as the notion of a projective PRG from [ABI+23], we require a different and incomparable set of security requirements. A projective PRG provides a way to take a PRG seed σ and project it to a new seed σ_S such that the output $\mathsf{PRG}(\sigma)$ and $\mathsf{PRG}(\sigma_S)$ agree on all indices $i \in S$ while ensuring pseudorandomness for the bits of $\mathsf{PRG}(\sigma)$ at indices $i \notin S$. Moreover, the projected seed σ_S should be succinct (i.e., have size $\mathsf{poly}(\lambda, \log \ell)$, where ℓ is the output length of the PRG). Importantly, for our applications, we require the projective seed to be *publicly sampleable*. Namely, there is an efficient sampler Samp that does not take in the original seed while still outputting a simulated projective seed that is indistinguishable to the honest projective seed. We give the formal definition below, and provide a more detailed comparison with the notion from [ABI+23] in Remark 4.2.

Definition 4.1 (Publicly-Sampleable Projective PRG)**.** A publicly-sampleable projective PRG is a tuple of efficient algorithms $\Pi_{\mathsf{pPRG}} = (\mathsf{Setup}, \mathsf{Samp}, \mathsf{Project}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp}, \sigma)$: On input the security parameter λ and the output length $\ell \in \mathbb{N}$, the setup algorithm outputs the public parameters pp and a seed σ.
- $\mathsf{Samp}(\mathsf{pp}, S) \to \sigma_S$: On input the public parameters pp and a set $S \subseteq [\ell]$, the public sampling algorithm outputs a (simulated) projected seed σ_S.
- $\mathsf{Project}(\mathsf{pp}, \sigma, S) \to \sigma_S$: On input the public parameters pp, the seed σ, and a set $S \subseteq [\ell]$, the projection algorithm outputs a projected seed σ_S.
- $\mathsf{Eval}(\mathsf{pp}, \sigma_S, S, i) \to y_i$: On input the public parameters pp, a projected seed σ_S, a set $S \subseteq [\ell]$, and an index $i \in S$, the evaluation algorithm outputs a bit $y_i \in \{0, 1\}$. This algorithm is *deterministic*.

The publicly-sampleable projective PRG should satisfy the following properties:

- **Correctness:** For all $\lambda, \ell \in \mathbb{N}$, all non-empty sets $S \subseteq [\ell]$, and all indices $i \in S$,

$$\Pr\left[\mathsf{Eval}(\mathsf{pp}, \sigma_{[\ell]}, [\ell], i) = \mathsf{Eval}(\mathsf{pp}, \sigma_S, S, i) \ : \ \begin{array}{c} (\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell) \\ \sigma_{[\ell]} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell]) \\ \sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, S) \end{array}\right] = 1.$$

- **Succinctness:** There exists a polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, \ell \in \mathbb{N}$, all (pp, σ) in the support of $\mathsf{Setup}(1^\lambda, 1^\ell)$, all sets $S \subseteq [\ell]$, and all σ_S in the support of $\mathsf{Samp}(\mathsf{pp}, S)$, it holds that $|\sigma_S| \le \mathsf{poly}(\lambda, \log \ell)$.
- **Sampling indistinguishability.** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $\beta \in \{0, 1\}$, we define the sampling-indistinguishability experiment $\mathsf{EXP}^{(\beta)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$:
  - **Setup:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the length parameter 1^ℓ. The challenger computes $(\mathsf{pp}, \sigma) \xleftarrow{\mathrm{R}} \mathsf{Setup}(1^\lambda, 1^\ell)$ and gives pp to $\mathcal{A}$.
  - **Challenge query:** Algorithm $\mathcal{A}$ specifies a set $S \subseteq [\ell]$. The challenger constructs the projected key σ_S as follows:
    * If $\beta = 0$, the challenger computes $\sigma_S \leftarrow \mathsf{Samp}(\mathsf{pp}, S)$.
    * If $\beta = 1$, the challenger computes $\sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, S)$.

    The challenger gives σ_S to $\mathcal{A}$.
  - **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which is the output of the experiment.

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

$$|\Pr[\mathsf{EXP}^{(0)}_{\mathsf{samp}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}_{\mathsf{samp}}(1^\lambda, \mathcal{A}) = 1]| = \mathsf{negl}(\lambda). \tag{4.1}$$

  We say that Π_{pPRG} satisfies statistical sampling indistinguishability if Eq. (4.1) holds for *all* adversaries $\mathcal{A}$, and that it satisfies perfect sampling indistinguishability if the negligible function in Eq. (4.1) is the identically-zero function.

- **Adaptive pseudorandomness:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $\beta \in \{0,1\}$, we define the pseudorandomness experiment $\mathsf{EXP}_{\mathsf{prg}}^{(\beta)}(1^\lambda, \mathcal{A})$:
  - **Setup:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the length parameter 1^ℓ. The challenger runs $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, $\sigma_{[\ell]} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell])$, and samples $\mathbf{u} \xleftarrow{\text{R}} \{0,1\}^\ell$. The challenger gives pp to $\mathcal{A}$.
  - **Evaluation queries:** Algorithm $\mathcal{A}$ can now make (adaptive) evaluation queries to the challenger. On each evaluation query, algorithm $\mathcal{A}$ specifies an index $i \in [\ell]$. The challenger responds as follows:
    - If $\beta = 0$, the challenger responds with $y_i = \mathsf{Eval}(\mathsf{pp}, \sigma_{[\ell]}, [\ell], i)$.
    - If $\beta = 1$, the challenger responds with $y_i = u_i$.
  - **Challenge query:** After $\mathcal{A}$ finishes making evaluation queries, the challenger computes the projected seed $\sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell] \setminus I)$, where $I \subseteq [\ell]$ is the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. The challenger then gives σ_S to $\mathcal{A}$.
  - **Output:** Algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$ which is the output of the experiment.

  The publicly-sampleable projective PRG satisfies adaptive pseudorandomness if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{EXP}_{\mathsf{prg}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{prg}}^{(1)}(1^\lambda, \mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Remark 4.2 (Comparison with [ABI+23])**.** The basic syntax of Definition 4.1 as well as the correctness and succinctness properties are the same as those from [ABI+23, Definition 3.1]. The key differences between our notion and the previous ones are the following:

- **Public sampleability:** In a projective PRG, the public parameters pp and the seed σ are sampled jointly. For our applications, we also require a public way to sample a projected seed (i.e., the Samp algorithm), and moreover, that the publicly-sampled seed be computationally indistinguishable from the actual projected seed (even given the public parameters). In our application to the Gentry-Waters compiler, the encryption algorithm uses the public sampling algorithm to sample a seed when encrypting. Conversely, the reduction algorithm will prepare the challenge ciphertext using a projected seed, and as such, it is important that the publicly-sampled seed appears indistinguishable from a projected seed. This is a new property to this work and with the exception of the construction from RSA, the projective PRGs in [ABI+23] from number-theoretic assumptions do *not* support this property. Namely, in the number-theoretic constructions (i.e., based on groups, pairings, or LWE) from [ABI+23], there is an efficient way to check consistency between a projected seed and the public parameters.

- **Adaptive pseudorandomness:** For our applications to adaptively-secure broadcast encryption, we require our projective PRGs to satisfy an *adaptive* pseudorandomness notion. Namely, the adversary in the pseudorandomness game is allowed to make evaluation queries *before* it sees the projected key. In [ABI+23], the adversary simply declares a challenge set S and is then given the projected seed σ_S onto S together and its goal is to distinguish the PRG values at indices $i \notin S$ from random. The ability to adaptively choose the set S after making evaluation queries is essential when using projective PRGs to realize adaptive security for broadcast encryption.

At the same time, the work of [ABI+23] consider additional properties such as robustness, reusability, and sublinear-size public parameters. These security notions are useful for their application to succinct computational secret sharing, but are not relevant in our applications to broadcast encryption.

5 Semi-Static Security to Adaptive Security via Projective PRGs

In this section, we show how to use projective PRGs to generically upgrade any semi-statically-secure broadcast encryption scheme into an adaptively-secure scheme. Our compiler follows the Gentry-Waters [GW09] strategy, except we show that the random oracle can be instantiated with a publicly-sampleable projective PRG (Definition 4.1). In Appendix A, we show how the same techniques can also be used to lift a semi-statically-secure *distributed* broadcast encryption scheme into an adaptively-secure scheme.

Construction 5.1 (Adaptively-Secure Broadcast Encryption)**.** Let $\Pi_{\mathsf{SS}} = (\mathsf{SS.Setup}, \mathsf{SS.KeyGen}, \mathsf{SS.Enc}, \mathsf{SS.Dec})$ be a semi-statically-secure broadcast encryption scheme, $\Pi_{\mathsf{pPRG}} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.Samp}, \mathsf{pPRG.Project}, \mathsf{pPRG.Eval})$ be a publicly-sampleable projective PRG. We construct an adaptively-secure broadcast encryption $\Pi_{\mathsf{BE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the number of users N, the setup algorithm proceeds as follows:
  1. Sample a random string $\mathbf{s} \xleftarrow{\mathrm{R}} \{0,1\}^N$ and $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$.
  2. Sample $(\mathsf{SS.mpk}, \mathsf{SS.msk}) \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$. For ease of exposition, we index the set $[2N]$ using a pair $(i, b) \in [N] \times \{0,1\}$.
  3. Output $\mathsf{mpk} = (\mathsf{SS.mpk}, \mathsf{pp})$ and $\mathsf{msk} = (\mathsf{SS.msk}, \mathbf{s})$.
- $\mathsf{KeyGen}(\mathsf{msk}, i)$: On input the master secret key $\mathsf{msk} = (\mathsf{SS.msk}, \mathbf{s})$, the key-generation algorithm samples $\mathsf{SS.sk}_{i,s_i} \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.msk}, (i, s_i))$. It outputs the secret key $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{i,s_i})$.
- $\mathsf{Enc}(\mathsf{mpk}, S, \mu)$: On input the master public key $\mathsf{mpk} = (\mathsf{SS.mpk}, \mathsf{pp})$, a set of users $S \subseteq [N]$, and a message $\mu \in \{0,1\}$, the encryption algorithm proceeds as follows:
  1. Sample $\sigma_S \leftarrow \mathsf{pPRG.Samp}(\mathsf{pp}, S)$.
  2. For each $i \in S$, compute $t_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_S, S, i)$. Let $S_0 = \{(i, t_i)\}_{i \in S}$ and $S_1 = \{(i, 1 - t_i)\}_{i \in S}$.
  3. Compute ciphertexts $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_0, \mu)$ and $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, \mu)$. Output the ciphertext $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$.
- $\mathsf{Dec}(\mathsf{mpk}, \mathsf{sk}, S, \mathsf{ct})$: On input the master public key $\mathsf{mpk} = (\mathsf{SS.mpk}, \mathsf{pp})$, the secret key $\mathsf{sk} = (j, b, \mathsf{SS.sk})$, a set $S \subseteq [N]$, and a ciphertext $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma)$, the decryption algorithm proceeds as follows:
  1. If $j \notin S$, then output 0.
  2. For each $i \in S$, compute $t_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma, S, i)$. Let $S_0 = \{(i, t_i)\}_{i \in S}$ and $S_1 = \{(i, 1 - t_i)\}_{i \in S}$.
  3. Finally, compute and output $\mathsf{SS.Dec}(\mathsf{SS.mpk}, \mathsf{SS.sk}, S_{b \oplus t_j}, \mathsf{SS.ct}_{b \oplus t_j})$.

Theorem 5.2 (Correctness)**.** *If Π_{SS} is correct, then Construction 5.1 is correct.*

Proof. Take any $\lambda, N \in \mathbb{N}$. Take any set $S \subseteq [N]$ and index $i \in S$, any message $\mu \in \{0,1\}$. Let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, \mu)$. By construction, the following hold:

- First, $\mathsf{mpk} = (\mathsf{SS.mpk}, \mathsf{pp})$ and $\mathsf{msk} = (\mathsf{SS.msk}, \mathbf{s})$ where $(\mathsf{SS.mpk}, \mathsf{SS.msk}) \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$, $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and $\mathbf{s} \xleftarrow{\mathrm{R}} \{0,1\}^N$.
- Next $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{i,s_i})$ where $\mathsf{SS.sk}_{i,s_i} \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.msk}, (i, s_i))$.
- Finally, $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$ where $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_0, \mu)$, $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, \mu)$, $\sigma_S \leftarrow \mathsf{pPRG.Samp}(\mathsf{pp}, S)$, $S_0 = \{(i, t_i)\}_{i \in S}$, $S_1 = \{(i, 1 - t_i)\}_{i \in S}$, and $t_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_S, S, i)$ for all $i \in S$.

Consider now the value of $\mathsf{Dec}(\mathsf{mpk}, \mathsf{sk}, S, \mathsf{ct})$. By construction, if $s_i = t_i$, then $(i, s_i) \in S_0$. Conversely, if $s_i = 1 - t_i$, then $(i, s_i) \in S_1$. This means $(i, s_i) \in S_{s_i \oplus t_i}$. By correctness of Π_{SS}, this means that

$$\mathsf{SS.Dec}\big(\mathsf{SS.mpk}, \mathsf{SS.sk}_{i,s_i}, S_{s_i \oplus t_i}, \mathsf{SS.ct}_{s_i \oplus t_i}\big) = \mu,$$

and correctness follows. □

Theorem 5.3 (Adaptive Security)**.** *Suppose Π_{SS} satisfies semi-static security and Π_{pPRG} satisfies correctness, sampling indistinguishability, and adaptive pseudorandomness. Then Construction 5.1 is adaptively secure.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the adaptive broadcast security game. We begin by defining a sequence of hybrid experiments:

- Hyb_0: This is experiment $\mathsf{EXP}_{\mathsf{BE}}^{(0)}$ from Definition 3.3:
  - **Setup:** On input the security parameter 1^λ, $\mathcal{A}$ outputs 1^N. The challenger responds by computing $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and gives $\mathsf{mpk} = (\mathsf{SS.mpk}, \mathsf{pp})$ to $\mathcal{A}$. Specifically, the challenger samples $\mathbf{s} \xleftarrow{\mathrm{R}} \{0,1\}^N$, $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$, $(\mathsf{SS.mpk}, \mathsf{SS.msk}) \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$.
  - **Key-generation queries:** On each key-generation query $i \in [N]$, the challenger responds with $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$. In particular, $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{i,s_i})$ where $\mathsf{SS.sk}_{i,s_i} \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.msk}, (i, s_i))$.
  - **Challenge query:** After $\mathcal{A}$ outputs a set $S \subseteq [N] \setminus I$ where $I \subseteq [N]$ is the set of indices on which $\mathcal{A}$ made a key-generation query. The challenger responds with $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, 0)$. Specifically, $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$ where $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_0, 0)$, $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, 0)$, $\sigma_S \leftarrow \mathsf{pPRG.Samp}(\mathsf{pp}, S)$, $S_0 = \{(i, t_i)\}_{i \in S}$, $S_1 = \{(i, 1 - t_i)\}_{i \in S}$, and $t_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_S, S, i)$ for all $i \in S$.
  - **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.
- Hyb_1: Same as Hyb_0 except when constructing the challenge ciphertext, the challenger now computes $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$.
- Hyb_2: Same as Hyb_1, except during setup, the challenger computes $\sigma_{[N]} \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, [N])$. Then, it sets $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_{[N]}, [N], i)$ for all $i \in [N]$.
- Hyb_3: Same as Hyb_2 except the challenger switches $\mathsf{SS.ct}_1$ to be an encryption of 1. Namely, the challenger now computes $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, 1)$.
- Hyb_4: Same as Hyb_3, except the challenger now samples $\mathbf{s} \xleftarrow{\mathrm{R}} \{0,1\}^N$ during setup.
- Hyb_5: Same as Hyb_4, except during setup, the challenger first computes $\sigma_{[N]} \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, [N])$. Then, it sets $s_i = 1 - \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_{[N]}, [N], i)$ for all $i \in [N]$.
- Hyb_6: Same as Hyb_5, except the challenger switches $\mathsf{SS.ct}_0$ to be an encryption of 1. Namely, the challenger now computes $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_0, 1)$.
- Hyb_7: Same as Hyb_6 except the challenger now samples $\mathbf{s} \xleftarrow{\mathrm{R}} \{0,1\}^N$ during setup.
- Hyb_8: Same as Hyb_7 except when constructing the challenge ciphertext, the challenger samples $\sigma_S \leftarrow \mathsf{pPRG.Samp}(\mathsf{pp}, S)$. This is experiment $\mathsf{EXP}_{\mathsf{BE}}^{(1)}$ from Definition 3.3.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid Hyb_i with adversary $\mathcal{A}$ (and an implicit security parameter λ). We now show that the output distributions of each adjacent pair of hybrid experiments is computationally indistinguishable.

Lemma 5.4. *Suppose* Π_{pPRG} *satisfies sampling indistinguishability. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the sampling indistinguishability game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{A}$ outputs a length parameter 1^N which algorithm $\mathcal{B}$ forwards to its challenger. The sampling indistinguishability challenger replies with pp.
2. Algorithm $\mathcal{B}$ now samples $\mathbf{s} \xleftarrow{\mathrm{R}} \{0,1\}^N$ and $(\mathsf{SS.mpk}, \mathsf{SS.msk}) \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$. It gives $\mathsf{mpk} = (\mathsf{SS.mpk}, \mathsf{pp})$ to $\mathcal{A}$.
3. When algorithm $\mathcal{A}$ makes a key-generation query on an index $i \in [N]$, algorithm $\mathcal{B}$ computes $\mathsf{SS.sk}_{i,s_i} \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.msk}, (i, s_i))$ and replies with $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{i,s_i})$.
4. When algorithm $\mathcal{A}$ makes a challenge query for the set $S \subseteq [N] \setminus I$, algorithm $\mathcal{B}$ forwards S to its challenger and receives σ_S.

5. For each $i \in S$, algorithm $\mathcal{B}$ computes $t_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_S, S, i)$. It then defines $S_0 = \{(i, t_i)\}_{i \in S}$, $S_1 = \{(i, 1-t_i)\}_{i \in S}$, $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_0, 0)$, $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, 0)$. The challenger responds with $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$.

6. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$ which algorithm $\mathcal{B}$ also outputs.

We now analyze the distribution of $\mathsf{EXP}^{(0)}_{\mathsf{samp}}(1^\lambda, \mathcal{B})$ for $\beta \in \{0,1\}$:

- Suppose $\beta = 0$. This means $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and $\sigma_S \leftarrow \mathsf{pPRG.Samp}(\mathsf{pp}, S)$. This is the distribution in Hyb_0, so algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$.
- Suppose $\beta = 1$. This means $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$. This is the distribution in Hyb_1, so algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ breaks mode indistinguishability with non-negligible advantage ε. □

Lemma 5.5. *Suppose* Π_{pPRG} *satisfies adaptive pseudorandomness. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

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 $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the adaptive pseudorandomness game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{A}$ outputs a length parameter 1^N which algorithm $\mathcal{B}$ forwards to its challenger. The sampling indistinguishability challenger replies with pp.

2. Algorithm $\mathcal{B}$ samples $(\mathsf{SS.mpk}, \mathsf{SS.msk}) \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$ and gives $\mathsf{mpk} = (\mathsf{SS.mpk}, \mathsf{pp})$ to $\mathcal{A}$.

3. When algorithm $\mathcal{A}$ makes a key-generation query on an index $i \in [N]$, algorithm $\mathcal{B}$ makes an evaluation query to its challenger on i and receives s_i. It then computes $\mathsf{SS.sk}_{i,s_i} \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.msk}, (i, s_i))$ and replies with $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{i,s_i})$.

4. When algorithm $\mathcal{A}$ makes a challenge query for the set $S \subseteq [N] \setminus I$, algorithm $\mathcal{B}$ makes an evaluation query on all indices $[N] \setminus S$. It then makes a challenge query and receives σ_S.

5. For each $i \in S$, algorithm $\mathcal{B}$ computes $t_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_S, S, i)$. It defines $S_0 = \{(i, t_i)\}_{i \in S}$, $S_1 = \{(i, 1 - t_i)\}_{i \in S}$, $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_0, 0)$, $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, 0)$. The challenger responds with $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$.

6. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$ which algorithm $\mathcal{B}$ also outputs.

We now analyze the distribution $\mathsf{EXP}^{(\beta)}_{\mathsf{prg}}(1^\lambda, \mathcal{B})$. First, let $I \subseteq [N]$ be the indices algorithm $\mathcal{A}$ makes to the key-generation oracle, and let $I_{\mathcal{B}}$ be the indices algorithm $\mathcal{B}$ makes to its evaluation oracle. From the requirements of the broadcast security definition, we have that $S \subseteq [N] \setminus I$, or equivalently, that $I \subseteq [N] \setminus S$. By construction of $\mathcal{B}$, we have that $I_{\mathcal{B}} = I \cup ([N] \setminus S) = [N] \setminus S$. Correspondingly, $[N] \setminus I_{\mathcal{B}} = S$. We now consider the distribution of of $\mathsf{EXP}^{(\beta)}_{\mathsf{prg}}(1^\lambda, \mathcal{B})$ for each $\beta \in \{0,1\}$:

- Suppose $\beta = 0$. In this case, the challenger samples $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and computes $\sigma_{[N]} \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, [N])$. It responds to each evaluation query on $i \in [N]$ with $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma, [N], i)$. Since $[N] \setminus I_{\mathcal{B}} = S$, the challenger responds with $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$ in the challenge phase. This is precisely the behavior in $\mathsf{Hyb}_2(\mathcal{A})$, so in this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.
- Suppose $\beta = 0$. In this case, the challenger samples $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and $\mathbf{u} \xleftarrow{\mathrm{R}} \{0,1\}^N$. It responds to each evaluation query on $i \in [N]$ with $s_i = u_i$. In the challenge phase, the challenger again responds with $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$. This is precisely the behavior in $\mathsf{Hyb}_1(\mathcal{A})$, so in this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ breaks adaptive pseudorandomness with non-negligible advantage ε. □

Lemma 5.6. *Suppose* Π_{SS} *satisfies semi-static security and* Π_{pPRG} *is correct. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the semi-static security game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{A}$ outputs a length parameter 1^N.
2. Algorithm $\mathcal{B}$ samples $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and computes $\sigma_{[N]} \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, [N])$. Then, for each $i \in [N]$, it computes $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_{[N]}, [N], i)$.
3. Algorithm $\mathcal{B}$ constructs the set $S^* = \{(i, 1 - s_i)\}_{i \in [N]}$. It forwards 1^{2N} together with the set S^* to the semi-static security challenger. The challenger replies with $\mathsf{SS.mpk}$. Algorithm $\mathcal{B}$ gives $\mathsf{mpk} = (\mathsf{SS.mpk}, \mathsf{pp})$ to $\mathcal{A}$.
4. When algorithm $\mathcal{A}$ makes a key-generation query on an index $i \in [N]$, algorithm $\mathcal{B}$ makes a key-generation query to its challenger on $(i, s_i) \in [2N] \setminus S^*$ to get a key $\mathsf{SS.sk}_{i,s_i}$. It replies to $\mathcal{A}$ with $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{i,s_i})$.
5. When algorithm $\mathcal{A}$ makes a challenge query for a set $S \subseteq [N]$, algorithm $\mathcal{B}$ starts by computing a projected seed $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$. Then it sets $S_0 = \{(i, s_i)\}_{i \in S}$ and $S_1 = \{(i, 1 - s_i)\}_{i \in S}$. It computes $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_0, 0)$ and forwards S_1 to the semi-static security challenger. The challenger replies with a ciphertext $\mathsf{SS.ct}_1$. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$.
6. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$ which algorithm $\mathcal{B}$ also outputs.

By construction, algorithm $\mathcal{B}$ is a *valid* adversary for the semi-static security game. Namely, algorithm $\mathcal{B}$ only makes key-generation queries on indices $(i, s_i) \in [2N] \setminus S^*$ and moreover, the challenge set $S_1 \subseteq S^*$. We now analyze the distributions of $\mathsf{EXP}^{(\beta)}_{\mathsf{SSBE}}(1^\lambda, \mathcal{B})$. We consider each component separately.

- The semi-static security challenger samples $(\mathsf{SS.mpk}, \mathsf{SS.msk}) \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$, which coincides with the distribution of $\mathsf{SS.mpk}$ in Hyb_2 and Hyb_3.
- Next, the semi-static security challenger responds to key-generation queries by computing $\mathsf{SS.sk}_{i,s_i} \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.msk}, (i, s_i))$, which again coincides with the distribution in Hyb_2 and Hyb_3.
- Next, consider the sets S_0 and S_1. In the reduction, algorithm $\mathcal{B}$ sets $S_0 = \{(i, s_i)\}_{i \in S}$ and $S_1 = \{(i, 1 - s_i)\}_{i \in S}$, where $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_{[N]}, [N], i)$. Since $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$, correctness of Π_{pPRG} means that $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_S, S, i)$ for all $i \in S$. Thus, the sets S_0 and S_1 are constructed exactly as in Hyb_2 and Hyb_3. Thus, $\mathsf{SS.ct}_0$ is distributed exactly according to the distribution in Hyb_2 and Hyb_3.
- It suffices to consider the distribution of $\mathsf{SS.ct}_1$. When $\beta = 0$, $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, 0)$ and when $\beta = 1$, $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, 1)$. The former corresponds to the distribution in Hyb_2 while the latter corresponds to the distribution in Hyb_3.

We conclude that algorithm $\mathcal{B}$ breaks semi-static security with non-negligible advantage ε. □

Lemma 5.7. *Suppose* Π_{pPRG} *satisfies adaptive pseudorandomness. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Follows by an analogous argument as the proof of Lemma 5.5. □

Lemma 5.8. *Suppose* Π_{pPRG} *satisfies adaptive pseudorandomness. 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. Follows by an analogous argument as the proof of Lemma 5.5. □

Lemma 5.9. *Suppose* Π_{SS} *satisfies semi-static security and* Π_{pPRG} *is correct. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Follows by an analogous argument as the proof of Lemma 5.6. □

Lemma 5.10. *Suppose* Π_{pPRG} *satisfies adaptive pseudorandomness. Then, there exists a negligible function* $\text{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\text{Hyb}_6(\mathcal{A}) = 1] - \Pr[\text{Hyb}_7(\mathcal{A}) = 1]| = \text{negl}(\lambda)$.

Proof. Follows by an analogous argument as the proof of Lemma 5.5. □

Lemma 5.11. *Suppose* Π_{pPRG} *satisfies sampling indistinguishability. Then, there exists a negligible function* $\text{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\text{Hyb}_7(\mathcal{A}) = 1] - \Pr[\text{Hyb}_8(\mathcal{A}) = 1]| = \text{negl}(\lambda)$.

Proof. Follows by an analogous argument as the proof of Lemma 5.4. □

Adaptive security now follows by combining Lemmas 5.4 to 5.11. □

Theorem 5.12 (Succinctness)**.** *Suppose* Π_{SS} *and* Π_{pPRG} *are succinct. Then Construction 5.1 is succinct.*

Proof. In Construction 5.1, a ciphertext for a set $S \subseteq [N]$ consists of two ciphertexts for the semi-static broadcast encryption scheme Π_{SS} as well as a projected seed σ_S for Π_{pPRG}. By succinctness of the underlying primitives, the size of the ciphertext is $2 \cdot o(|S|) \cdot \text{poly}(\lambda, \log N) + \text{poly}(\lambda, \log N)$, which satisfies the required succinctness properties. □

Application to distributed broadcast encryption. As discussed in Section 2, we can adapt Construction 5.1 to upgrade any semi-statically-secure *distributed* broadcast encryption scheme into an adaptively secure one using a projective PRG. We give the full details in Appendix A. Our compiler yields the first constructions of distributed broadcast encryption in the plain model (see Table 2) from search assumptions over groups with bilinear maps as well as from witness encryption (together with function-binding hash functions).

6 Constructing Publicly-Sampleable Projective PRGs

In the following sections, we give a variety of constructions of projective PRGs from standard number theoretic assumptions: (1) the computational Diffie-Hellman (CDH) assumption in pairing-free groups; (2) the computational bilinear Diffie-Hellman (CBDH) assumption in pairing groups; and (3) the learning with errors (LWE) assumption. Our constructions follow a similar template as the number-theoretic constructions from [ABI+23], though we will need to introduce additional randomization to support the additional public-sampleability requirement we require. We refer to Remark 4.2 for further discussion of the definitional differences between our notion of a publicly-sampleable projective PRG and the notion from [ABI+23]. Finally, in Appendix B, we also give a construction from RSA; this construction is nearly identical to the RSA-based construction from [ABI+23].

6.1 Publicly-Sampleable Projective PRGs from CDH

In this section, we show how to construct a projective PRG from the Computational Diffie-Hellman (CDH) assumption in pairing-free groups. Our construction is an adaptation of the projective PRG scheme from [ABI+23, §3.4.1]. Importantly, the original construction from [ABI+23] does not satisfy sampling indistinguishability since the adversary can use the components of the public parameters to determine whether a given seed is the output of Project or the output of Samp. Our construction introduces additional randomization (specifically, the random exponent α in Construction 6.3) to ensure sampling indistinguishability. Additionally, we use the Goldreich-Levin hardcore extractor to reduce the assumption required for adaptive pseudorandomness from DDH to CDH. We begin by recalling the CDH assumption.

Definition 6.1 (Prime-Order Group Generator)**.** A prime-order group generator PrimeGroupGen 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 $p = 2^{\Theta(\lambda)}$ and generator g. We require 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 $\rho = \rho(\lambda)$.

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 $[\mathbf{A}]_{\mathbb{G}}$ 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 $c \cdot [\mathbf{A}]_{\mathbb{G}} := [c\mathbf{A}]_{\mathbb{G}}$ and $[\mathbf{A}]_{\mathbb{G}} + [\mathbf{B}]_{\mathbb{G}} := [\mathbf{A}+\mathbf{B}]_{\mathbb{G}}$. We now define the CDH problem.

Assumption 6.2 (Computational Diffie-Hellman)**.** Let PrimeGroupGen be a prime-order group generator. The computational Diffie-Hellman (CDH) assumption holds with respect to PrimeGroupGen if for all efficient (and possibly non-uniform) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(1^\lambda, \mathcal{G}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}}) = [vw]_{\mathbb{G}} : \begin{array}{c}\mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda) \\ v, w \xleftarrow{\text{R}} \mathbb{Z}_p.\end{array}\right] = \mathsf{negl}(\lambda).$$

Construction 6.3 (Publicly-Sampleable Projective PRG from CDH)**.** Let λ be a security parameter. Let PrimeGroupGen be a prime-order group generator and let $\rho = \rho(\lambda)$ be a bound on the bit-length of the group elements associated with PrimeGroupGen. Let $\mathsf{hc} \colon \{0,1\}^\rho \times \{0,1\}^\rho \to \{0,1\}$ be the Goldreich-Levin extractor. We construct a publicly-sampleable projective PRG $\Pi_{\mathsf{pPRG}} = (\mathsf{Setup}, \mathsf{Samp}, \mathsf{Project}, \mathsf{Eval})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell)$: On input the security parameter λ and the output length $\ell \in \mathbb{N}$, the setup algorithm starts by sampling $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$. It then samples $\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$ and $\mathbf{a}, \mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_p^\ell$. It computes $\mathbf{M} \in \mathbb{Z}_p^{\ell\times\ell}$ where $M_{i,j} = s_i a_j$ for all $i \neq j$ and $M_{i,i} = 0$ for all $i \in [\ell]$. Finally, it samples the extractor seed $\mathbf{r} \xleftarrow{\text{R}} \{0,1\}^\rho$ and outputs the public parameters $\mathsf{pp} = (\mathcal{G}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{M}]_{\mathbb{G}}, \mathbf{r})$ and the seed $\sigma = (\alpha, \mathbf{s})$.
- $\mathsf{Samp}(\mathsf{pp}, S)$: On input the public parameters $\mathsf{pp} = ((\mathbb{G}, p, g), [\mathbf{a}]_{\mathbb{G}}, [\mathbf{M}]_{\mathbb{G}}, \mathbf{r})$ and the target set $S \subseteq [\ell]$, the sampling algorithm samples $x \xleftarrow{\text{R}} \mathbb{Z}_p$ and outputs $\sigma_S = x$.
- $\mathsf{Project}(\mathsf{pp}, \sigma, S)$: On input the public parameters $\mathsf{pp} = ((\mathbb{G}, p, g), [\mathbf{a}]_{\mathbb{G}}, [\mathbf{M}]_{\mathbb{G}}, \mathbf{r})$, the $\sigma = (\alpha, \mathbf{s})$, and a target set $S \subseteq [\ell]$, the projection algorithm outputs the projected seed $\sigma_S = \alpha + \sum_{i \in S} s_i$.
- $\mathsf{Eval}(\mathsf{pp}, \sigma_S, S, i)$: On input the public parameters $\mathsf{pp} = ((\mathbb{G}, p, g), [\mathbf{a}]_{\mathbb{G}}, [\mathbf{M}]_{\mathbb{G}}, \mathbf{r})$, a projected seed $\sigma_S \in \mathbb{Z}_p$, the associated set of indices $S \subseteq [\ell]$, and an index $i \in S$, the evaluation algorithm computes

  $$[y_i]_{\mathbb{G}} = \sigma_S \cdot [a_i]_{\mathbb{G}} - \sum_{j \in S \setminus \{i\}} [M_{j,i}]_{\mathbb{G}},$$

  and outputs $\mathsf{hc}([y_i]_{\mathbb{G}}, \mathbf{r})$.

Theorem 6.4 (Correctness)**.** *Construction 6.3 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, output length $\ell \in \mathbb{N}$, set of indices $S \subseteq [\ell]$, and any index $i \in S$. Let $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$ and $\sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, S)$. By construction, $\mathsf{pp} = (\mathcal{G}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{M}]_{\mathbb{G}}, \mathbf{r})$ and $\sigma_S = \alpha + \sum_{i \in S} s_i$. Consider the value of $[y_i]_{\mathbb{G}}$ computed by $\mathsf{Eval}(\mathsf{pp}, \sigma_S, S, i)$. By definition,

$$y_i = \sigma_S \cdot a_i - \sum_{j \in S \setminus \{i\}} M_{j,i} = \left(\alpha + \sum_{j \in S} s_j\right) a_i - \sum_{j \in S \setminus \{i\}} s_j a_i = (\alpha + s_i) a_i.$$

Next, let $\sigma_{[\ell]} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell])$. Then $\sigma_{[\ell]} = \alpha + \sum_{i \in [\ell]} s_i$. A similar calculation now show that $\mathsf{Eval}(\mathsf{pp}, \sigma_{[\ell]}, [\ell], i) = \mathsf{Eval}(\mathsf{pp}, \sigma_S, S, i)$. ☐

Theorem 6.5 (Succinctness)**.** *Construction 6.11 is succinct.*

Proof. Take any (pp, σ) in the support of $\mathsf{Setup}(1^\lambda, 1^\ell)$. For all sets S, the projected key $[\sigma_S]_{\mathbb{G}}$ output by $\mathsf{Project}(\mathsf{pp}, \sigma, S)$ consists of a single element of $\mathbb{Z}_p$, which has size $\Theta(\lambda)$. In particular, the size of the projected seed is independent of the size of the associated set S. ☐

Theorem 6.6 (Sampling Indistinguishability)**.** *Construction 6.3 satisfies perfect sampling indistinguishability.*

Proof. Let $\mathcal{A}$ be an adversary for the sampling indistinguishability game. We first recall the experiments in the sampling indistinguishability security definition:

- $\mathsf{EXP}^{(0)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$: This experiment proceeds as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the input length 1^ℓ.
  2. The challenger samples $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$. Concretely, the challenger samples $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$, $\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, $\mathbf{a}, \mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_p^\ell$, and $\mathbf{r} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$. It computes $\mathbf{M} \in \mathbb{Z}_p^{\ell \times \ell}$ where $M_{i,j} = s_i a_j$ for all $i \neq j$ and $M_{i,i} = 0$ for all $i \in [\ell]$. The challenger gives $\mathsf{pp} = (\mathcal{G}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{M}]_{\mathbb{G}}, \mathbf{r})$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a set $S \subseteq [\ell]$ and the challenger replies with $\sigma_S \leftarrow \mathsf{Samp}(\mathsf{pp}, S)$. Specifically, the challenger samples $\sigma_S \xleftarrow{\mathrm{R}} \mathbb{Z}_p$.
  4. Algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.
- $\mathsf{EXP}^{(1)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$: Same as $\mathsf{EXP}^{(0)}_{\mathsf{samp}}$, except the challenger sets $\sigma_S = \alpha + \sum_{i \in S} s_i$.

The only difference between $\mathsf{EXP}^{(0)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$ and $\mathsf{EXP}^{(1)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$ is the distribution of σ_S. In $\mathsf{EXP}^{(1)}_{\mathsf{samp}}$, the challenger samples $\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and the only element in the experiment that depends on α is $\sigma_S = \alpha + \sum_{i \in S} s_i$. Thus, we conclude that the distribution of σ_S is uniform over $\mathbb{Z}_p$ and independent of all other quantities. This is the distribution in $\mathsf{EXP}^{(0)}_{\mathsf{samp}}$. Since the two experiments are identically distributed, the theorem holds. □

Theorem 6.7 (Adaptive Pseudorandomness)**.** *If the CDH assumption holds with respect to* PrimeGroupGen*, then Construction 6.3 satisfies adaptive pseudorandomness.*

Proof. Before proving the theorem, we first state the following corollary of the CDH assumption (Assumption 6.2) and Theorem 3.2. This will be useful in our security analysis.

Lemma 6.8. *Suppose the CDH assumption holds with respect to* PrimeGroupGen *and let* $\mathsf{hc} \colon \{0,1\}^\rho \times \{0,1\}^\rho \to \{0,1\}$ *be the Goldreich-Levin extractor. Then, for all efficient (and possibly non-uniform) adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$:*

$$\left| \Pr\left[\mathcal{A}(1^\lambda, \mathbf{r}, (\mathcal{G}, \alpha, [v]_{\mathbb{G}}, [w]_{\mathbb{G}}), \mathsf{hc}([(\alpha+v)w]_{\mathbb{G}}, \mathbf{r})) = 1\right] - \Pr\left[\mathcal{A}(1^\lambda, \mathbf{r}, (\mathcal{G}, \alpha, [v]_{\mathbb{G}}, [w]_{\mathbb{G}}), b) = 1\right] \right| = \mathsf{negl}(\lambda),$$

where $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$*,* $\alpha, v, w \xleftarrow{\mathrm{R}} \mathbb{Z}_p$*,* $\mathbf{r} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$*, and* $b \xleftarrow{\mathrm{R}} \{0,1\}$*.*

Proof. We start by defining a joint distribution $(X, Y) = \{(X_\lambda, Y_\lambda)\}_{\lambda \in \mathbb{N}}$ as follows:

- Sample $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$ and exponents $\alpha, v, w \xleftarrow{\mathrm{R}} \mathbb{Z}_p$.
- Let $x = [(\alpha + v)w]_{\mathbb{G}}$ and $y = (\mathcal{G}, \alpha, [v]_{\mathbb{G}}, [w]_{\mathbb{G}})$. Output the pair (x, y).

Next, under the CDH assumption, X is computationally unpredictable given Y. To see this, suppose there exists an efficient algorithm $\mathcal{A}$ that can predict x given y when $(x, y) \leftarrow (X_\lambda, Y_\lambda)$ with non-negligible probability ε. We use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ for the CDH problem:

- On input the CDH challenge $(1^\lambda, \mathcal{G}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}})$, where $\mathcal{G} = (\mathbb{G}, p, g)$, algorithm $\mathcal{B}$ samples $\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and sets $y = (\mathcal{G}, \alpha, [v]_{\mathbb{G}}, [w]_{\mathbb{G}})$.
- Algorithm $\mathcal{B}$ gives 1^λ and y to algorithm $\mathcal{A}$ to obtain an output $x = [z]_{\mathbb{G}}$.
- Algorithm $\mathcal{B}$ outputs $[z]_{\mathbb{G}} - \alpha \cdot [w]_{\mathbb{G}}$.

By assumption, the CDH challenger samples $\mathcal{G} \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$ and $v, w \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Algorithm $\mathcal{B}$ then samples $\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, so algorithm $\mathcal{B}$ perfectly simulates the distribution of y in $(x, y) \leftarrow X_\lambda \times Y_\lambda$, where the associated value of x is $x = [(\alpha + v)w]_{\mathbb{G}}$. Thus, with probability ε, algorithm $\mathcal{A}$ outputs $[z]_{\mathbb{G}} = [(\alpha + v)w]_{\mathbb{G}}$. In this case, $z - \alpha w = vw$ and algorithm $\mathcal{B}$ successfully solves the CDH problem with the same non-negligible advantage ε. We conclude that under CDH, the distribution of X is computationally unpredictable given Y. Lemma 6.8 now follows from Theorem 3.2. □

Proof of Theorem 6.7. We now return to the proof of Theorem 6.7. For each index $i \in \{0\} \cup \mathbb{N}$, we define an experiment Hyb_i as follows:

- Hyb_i: This is a variant of the adaptive pseudorandomness experiment:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the length parameter 1^ℓ. The challenger runs $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$. Specifically, the challenger samples $\mathcal{G} = (\mathbb{G}, p, g) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$, $\alpha \xleftarrow{R} \mathbb{Z}_p$, $\mathbf{a}, \mathbf{s} \xleftarrow{R} \mathbb{Z}_p^\ell$, $\mathbf{r} \xleftarrow{R} \{0,1\}^\rho$ and constructs $\mathbf{M} \in \mathbb{Z}_p^{\ell \times \ell}$ where $M_{i,j} = s_i a_j$ for $i \neq j$ and $M_{i,i} = 0$ for all $i \in [\ell]$. It sets $\mathsf{pp} = (\mathbb{G}, [\mathbf{a}]_\mathbb{G}, [\mathbf{M}]_\mathbb{G}, \mathbf{r})$ and $\sigma = (\alpha, \mathbf{s})$. The challenger gives pp to $\mathcal{A}$.
  - The challenger also samples $\sigma_{[\ell]} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell])$ and $\mathbf{u} \xleftarrow{R} \{0,1\}^\ell$. In particular $\sigma_{[\ell]} = \alpha + \sum_{j \in [\ell]} s_j$.
  - When $\mathcal{A}$ makes an evaluation query on $j \in [\ell]$, the challenger replies with $\mathsf{Eval}(\mathsf{pp}, \sigma_{[\ell]}, [\ell], j)$ if $j > i$ and with u_j if $j \le i$. In particular, when $j > i$, the challenger computes
  $$[y_j]_\mathbb{G} = \sigma_{[\ell]} \cdot [a_j]_\mathbb{G} - \sum_{k \in [\ell] \setminus \{j\}} [M_{k,j}]_\mathbb{G} = [(\alpha + s_j) a_j]_\mathbb{G}$$
  and responds with $\mathsf{hc}([y_j]_\mathbb{G}, \mathbf{r})$.
  - After $\mathcal{A}$ finishes making evaluation queries, the challenger computes the seed $\sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell] \setminus I)$ where $I \subseteq [\ell]$ is the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. In particular, $\sigma_S = \alpha + \sum_{j \in [\ell] \setminus I} s_j$. The challenger gives σ_S to $\mathcal{A}$.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid Hyb_i with adversary $\mathcal{A}$ (and an implicit security parameter λ). By construction, observe that $\mathsf{EXP}_{\mathsf{prg}}^{(0)}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{EXP}_{\mathsf{prg}}^{(1)}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_\ell(\mathcal{A})$. For an index $i \in \{0\} \cup \mathbb{N}$, define $p_i := \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1]$. For each experiment, we define E_i to be the event that algorithm $\mathcal{A}$ makes an evaluation query on index i. Then, for all $i \in \{0\} \cup \mathbb{N}$, we can write

$$p_i = \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \mathsf{E}_i] + \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \neg\mathsf{E}_i].$$

By construction, the only difference between experiments Hyb_{i-1} and Hyb_i is how the challenger responds to an evaluation query on index i. If the adversary does *not* make an evaluation query on index i, then its view in the two experiments is identically distributed. Thus, we have

$$\Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \neg\mathsf{E}_i] = \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \neg\mathsf{E}_i].$$

Hence, we conclude

$$p_i - p_{i-1} = \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i]. \tag{6.1}$$

Suppose now that

$$|\Pr[\mathsf{EXP}_{\mathsf{prg}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{prg}}^{(1)}(1^\lambda, \mathcal{A}) = 1]| \ge \varepsilon(\lambda) \tag{6.2}$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the distinguishing problem from Lemma 6.8:

1. On input the challenge $(1^\lambda, \mathbf{r}, (\mathcal{G}, \alpha, [v]_\mathbb{G}, [w]_\mathbb{G}), b)$ where $\mathcal{G} = (\mathbb{G}, p, g)$, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain the output length 1^ℓ. Algorithm $\mathcal{B}$ samples an index $i \xleftarrow{R} [\ell]$.

2. For all $j \in [\ell] \setminus \{i\}$, algorithm $\mathcal{B}$ samples $a_j, s_j \xleftarrow{R} \mathbb{Z}_p$ and $u_j \xleftarrow{R} \{0,1\}$. It sets $[a_i]_\mathbb{G} = [w]_\mathbb{G}$. Next, for all $j \neq k$, algorithm $\mathcal{B}$ constructs $[M_{j,k}]_\mathbb{G}$ as follows:
$$[M_{j,k}]_\mathbb{G} = \begin{cases} [s_j a_k]_\mathbb{G} & j,k \neq i \\ [v]_\mathbb{G} \cdot a_k & j = i, k \neq i \\ s_j \cdot [w]_\mathbb{G} & k = i, j \neq i \end{cases}$$
Algorithm $\mathcal{B}$ sets $\mathsf{pp} = (\mathcal{G}, [\mathbf{a}]_\mathbb{G}, [\mathbf{M}]_\mathbb{G}, \mathbf{r})$ and gives pp to $\mathcal{A}$.

3. When algorithm $\mathcal{A}$ makes an evaluation query on an index $j \in [\ell]$, the challenger responds as follow.
   - If $j < i$, the challenger responds with u_j.
   - If $j = i$, the challenger responds with b.
   - If $j > i$, the challenger responds with $\mathsf{hc}([(\alpha + s_j)a_j]_{\mathbb{G}}, \mathbf{r})$.
4. After $\mathcal{A}$ finishes making evaluation queries, let $I \subseteq [\ell]$ be the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. If $i \notin I$, then algorithm $\mathcal{B}$ outputs 0. Otherwise algorithm $\mathcal{B}$ responds with $\sigma_S = \alpha + \sum_{j \in [\ell] \setminus I} s_j$. Since $i \in I$, this means $i \notin [\ell] \setminus I$, so algorithm $\mathcal{B}$ knows all of the exponents s_j needed to construct σ_S.
5. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ which algorithm $\mathcal{B}$ also outputs.

Since $\mathcal{A}$ is efficient, algorithm $\mathcal{B}$ is efficient, so it suffices to analyze its advantage.

Analyzing the advantage of $\mathcal{B}$. Let W_0 be the event that $\mathcal{B}$ outputs 1 when the challenger sets $b = \mathsf{hc}([(\alpha + v)w]_{\mathbb{G}}, \mathbf{r})$ and W_1 be the event that $\mathcal{B}$ outputs 1 when the challenger samples $b \xleftarrow{R} \{0, 1\}$. Suppose algorithm $\mathcal{B}$ samples $i = i^*$ in the above reduction. By construction, the challenger for the experiment in Lemma 6.8 samples $\alpha, v, w \xleftarrow{R} \mathbb{Z}_p$. Thus, algorithm $\mathcal{B}$ perfectly simulates the public parameters (where v plays the role of s_{i^*} and w plays the role of a_{i^*}) and the evaluation queries on indices $j \neq i^*$ for $\mathcal{A}$. We consider the distribution of the challenge bit b.

- Suppose $b = \mathsf{hc}([(\alpha + v)w]_{\mathbb{G}}, \mathbf{r}) = \mathsf{hc}([\alpha + s_{i^*} a_{i^*}]_{\mathbb{G}}, \mathbf{r})$. In this case, the responses to the evaluation queries are distributed according to the specification in Hyb_{i^*-1}. We consider the probability that algorithm $\mathcal{B}$ outputs 1 in this case. First, if algorithm $\mathcal{A}$ does not make an evaluation query on index i^*, then algorithm $\mathcal{B}$ always outputs 0. If algorithm $\mathcal{A}$ does make an evaluation query on index i^*, then algorithm $\mathcal{B}$ perfectly simulates the challenge according to the distribution in Hyb_{i^*-1}, and thus, outputs 1 with probability $\Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}]$. Thus, in this case, algorithm $\mathcal{B}$ outputs 1 with probability
$$\Pr[\mathcal{B} \text{ outputs } 1 \mid i = i^*] = \Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}] \cdot \Pr[\mathsf{E}_{i^*}] = \Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_{i^*}].$$
- Suppose $b \xleftarrow{R} \{0, 1\}$. In this case, the responses to the evaluation queries are distributed according to the specification in Hyb_{i^*}. By a similar reasoning as in the previous case, we conclude that in this case
$$\Pr[\mathcal{B} \text{ outputs } 1 \mid i = i^*] = \Pr[\mathsf{Hyb}_{i^*}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}] \cdot \Pr[\mathsf{E}_{i^*}] = \Pr[\mathsf{Hyb}_{i^*}(\mathcal{A}) = 1 \wedge \mathsf{E}_{i^*}].$$

Finally, algorithm $\mathcal{B}$ samples $i \xleftarrow{R} [\ell]$. Thus
$$\Pr[W_0 = 1] = \frac{1}{\ell} \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i],$$
$$\Pr[W_1 = 1] = \frac{1}{\ell} \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i}(\mathcal{A}) = 1 \wedge \mathsf{E}_i].$$

Since $\mathsf{EXP}_{\mathsf{prg}}^{(0)}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{EXP}_{\mathsf{prg}}^{(1)}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_\ell(\mathcal{A})$, we appeal to Eqs. (6.1) and (6.2) and conclude that
$$\begin{aligned} |\Pr[W_0 = 1] - \Pr[W_1 = 1]| &= \frac{1}{\ell} \left| \sum\nolimits_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \Pr[\mathsf{Hyb}_{i}(\mathcal{A}) = 1 \wedge \mathsf{E}_i] \right| \\ &= \frac{1}{\ell} \left| \sum\nolimits_{i \in [\ell]} p_i - p_{i-1} \right| \\ &= \frac{1}{\ell} |p_0 - p_\ell| = \frac{1}{\ell} |\Pr[\mathsf{EXP}_{\mathsf{prg}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{prg}}^{(1)}(1^\lambda, \mathcal{A}) = 1]| \geq \frac{\varepsilon}{\ell}, \end{aligned}$$
which is non-negligible since $\mathcal{A}$ is efficient which means $\ell = \mathsf{poly}(\lambda)$. □

6.2 Publicly-Sampleable Projective PRGs from Computational Bilinear Diffie-Hellman

A limitation of Construction 6.3 is the size of the public parameters scales *quadratically* with the output length of the projective PRG. This is because the public parameters consist of a matrix of group elements $[s_i a_j]_{\mathbb{G}}$ for all $i \neq j \in [\ell]$. In this section, we show how we can use bilinear maps to *compress* the public parameters in this construction to be *linear* in ℓ. Note that the naïve approach of simply giving out $[s_i]_{\mathbb{G}}$ and $[a_j]_{\mathbb{G}}$ for all $i, j \in [\ell]$ in the public parameters and using the pairing to compute the pairwise products $[s_i a_j]_{\mathbb{G}_T} = [s_i]_{\mathbb{G}} \cdot [a_j]_{\mathbb{G}}$ does *not* work because this would also reveal the non-cross-terms $[s_i a_i]_{\mathbb{G}_T}$. Instead, we adopt the approach from [BW10] who show how to use bilinear maps to encode $[s_i a_j]_{\mathbb{G}_T}$ for all $i \neq j \in [\ell]$ using just a linear number of group elements, while simultaneously ensuring that the non-cross-terms $[s_i a_i]_{\mathbb{G}_T}$ remain (computationally) hidden. We note that this also gives the first projective PRG scheme from bilinear maps with a linear-size public parameters; the scheme based on bilinear maps from [ABI+23] also had quadratic-size public parameters. We begin by recalling the computational bilinear Diffie-Hellman assumption over (symmetric) pairing groups.

Definition 6.9 (Prime-Order Bilinear Group Generator)**.** A (symmetric) prime-order bilinear group generator PrimeBGroupGen is an efficient algorithm that takes as input the security parameter 1^λ and outputs a description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e)$ of cyclic groups $\mathbb{G}, \mathbb{G}_T$ with prime-order $p = 2^{\Theta(\lambda)}$, a generator g of $\mathbb{G}$, and an efficiently-computable non-degenerate bilinear map $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We require that the group operation in both $\mathbb{G}$ and $\mathbb{G}_T$ to be efficiently-computable, and that each element of $\mathbb{G}$ and $\mathbb{G}_T$ can be represented by a bit-string of length at most $\rho = \rho(\lambda)$.

Notation. We also use implicit notation to represent group elements. For a symmetric-pairing group $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e)$, we write $[\mathbf{A}]_{\mathbb{G}}$ to denote $g^{\mathbf{A}}$ and $[\mathbf{A}]_{\mathbb{G}_T}$ to denote $e(g,g)^{\mathbf{A}}$. For matrices $\mathbf{A}, \mathbf{B}$ with suitable dimensions and a scalar $c \in \mathbb{Z}_p$, we write $c \cdot [\mathbf{A}]_{\mathbb{G}} := [c\mathbf{A}]_{\mathbb{G}}$, $c \cdot [\mathbf{A}]_{\mathbb{G}_T} := [c\mathbf{A}]_{\mathbb{G}_T}$, $[\mathbf{A}]_{\mathbb{G}} + [\mathbf{B}]_{\mathbb{G}} := [\mathbf{A} + \mathbf{B}]_{\mathbb{G}}$, $[\mathbf{A}]_{\mathbb{G}_T} + [\mathbf{B}]_{\mathbb{G}_T} := [\mathbf{A} + \mathbf{B}]_{\mathbb{G}_T}$, and $[\mathbf{A}]_{\mathbb{G}} \cdot [\mathbf{B}]_{\mathbb{G}} := [\mathbf{AB}]_{\mathbb{G}_T}$.

Assumption 6.10 (Computational Bilinear Diffie-Hellman Assumption)**.** Let PrimeBGroupGen be a prime-order bilinear group generator. The computational bilinear Diffie-Hellman (CBDH) assumption holds with respect to PrimeGroupGen if for all efficient (and possibly non-uniform) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(1^\lambda, \mathcal{G}, [u]_{\mathbb{G}}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}}) = [uvw]_{\mathbb{G}_T} \;:\; \begin{array}{c} \mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda) \\ u, v, w \xleftarrow{\text{R}} \mathbb{Z}_p. \end{array}\right] = \mathsf{negl}(\lambda).$$

Construction 6.11 (Publicly-Sampleable Projective PRG from CBDH)**.** Let λ be a security parameter. Next, let PrimeBGroupGen be a prime-order bilinear group generator and let $\rho = \rho(\lambda)$ be a bound on the bit-length of the group elements associated with PrimeBGroupGen. Let $\mathsf{hc} \colon \{0,1\}^\rho \times \{0,1\}^\rho \to \{0,1\}$ be the Goldreich-Levin extractor. We construct a publicly-sampleable projective PRG $\Pi_{\mathsf{pPRG}} = (\mathsf{Setup}, \mathsf{Samp}, \mathsf{Project}, \mathsf{Eval})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell)$: On input the security parameter λ and the output length $\ell \in \mathbb{N}$, the setup algorithm starts by sampling $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda)$. It then samples $\alpha, \gamma, \delta \xleftarrow{\text{R}} \mathbb{Z}_p$ and $\mathbf{a}, \mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_p^\ell$. It computes vector $\mathbf{c}, \mathbf{d} \in \mathbb{Z}_p^\ell$ such that $c_i = (\gamma i + \delta) a_i$, and $d_i = (\gamma i + \delta) s_i$ Finally, it samples the extractor seed $\mathbf{r} \xleftarrow{\text{R}} \{0,1\}^\rho$ and outputs the public parameters $\mathsf{pp} = (\mathcal{G}, [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, [\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}, \mathbf{r})$ and the seed $\sigma = (\alpha, \mathbf{s})$.
- $\mathsf{Samp}(\mathsf{pp}, S)$: On input the public parameters $\mathsf{pp} = ((\mathbb{G}, \mathbb{G}_T, p, g, e), [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, [\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}, \mathbf{r})$ and the target set $S \subseteq [\ell]$, the sampling algorithm samples $x \xleftarrow{\text{R}} \mathbb{Z}_p$ and outputs $[\sigma_S]_{\mathbb{G}} = x \cdot [\gamma]_{\mathbb{G}}$.
- $\mathsf{Project}(\mathsf{pp}, \sigma, S)$: On input the public parameters $\mathsf{pp} = ((\mathbb{G}, \mathbb{G}_T, p, g, e), [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, [\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}, \mathbf{r})$, the seed $\sigma = (\alpha, \mathbf{s})$, and a target set $S \subseteq [\ell]$, the projection algorithm outputs the projected seed $[\sigma_S]_{\mathbb{G}} = (\alpha + \sum_{i \in S} s_i) \cdot [\gamma]_{\mathbb{G}}$.
- $\mathsf{Eval}(\mathsf{pp}, [\sigma_S]_{\mathbb{G}}, S, i)$: On input the public parameters $\mathsf{pp} = ((\mathbb{G}, \mathbb{G}_T, p, g, e), [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, [\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}, \mathbf{r})$, a seed $[\sigma_S]_{\mathbb{G}} \in \mathbb{G}$, the associated set of indices $S \subseteq [\ell]$, and an index $i \in S$, the evaluation algorithm first computes the cross terms $[M_{j,i}]_{\mathbb{G}_T}$

$$[M_{j,i}]_{\mathbb{G}_T} = (i - j)^{-1} \cdot \big([s_j]_{\mathbb{G}} \cdot [c_i]_{\mathbb{G}} - [a_i]_{\mathbb{G}} \cdot [d_j]_{\mathbb{G}}\big),$$

for each $j \in S \setminus \{i\}$. Then, it computes

$$[y_i]_{\mathbb{G}_T} = [a_i]_{\mathbb{G}} \cdot [\sigma_S]_{\mathbb{G}} - \sum_{j \in S \setminus \{i\}} [M_{j,i}]_{\mathbb{G}_T},$$

and outputs $\mathsf{hc}([y_i]_{\mathbb{G}_T}, \mathbf{r})$.

Theorem 6.12 (Correctness). *Construction 6.11 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, output length $\ell \in \mathbb{N}$, set of indices $S \subseteq [\ell]$, and index $i \in S$. Take any (pp, σ) in the support of $\mathsf{Setup}(1^\lambda, 1^\ell)$. Let $[\sigma_S]_{\mathbb{G}} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, S)$. By construction,

$$\mathsf{pp} = (\mathcal{G}, [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, [\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}, \mathbf{r}) \quad \text{and} \quad [\sigma_S]_{\mathbb{G}} = \left(\alpha + \sum_{i \in S} s_i\right) \cdot [\gamma]_{\mathbb{G}}.$$

Consider the value of $M_{j,i}$ and y_i computed by $\mathsf{Eval}(\mathsf{pp}, [\sigma_S]_{\mathbb{G}}, S, i)$. By construction, for all $i \in [\ell]$, the Setup algorithm sets $c_i = (\gamma i + \delta) a_i$ and $d_i = (\gamma i + \delta) s_i$. By definition, for all $j \in S \setminus \{i\}$, we now have

$$M_{j,i} = (i-j)^{-1}(s_j c_i - a_i d_j) = (i-j)^{-1}((\gamma i + \delta) s_j a_i - (\gamma j + \delta) a_i s_j) = \gamma s_j a_i.$$

Next, $\sigma_S = \gamma(\alpha + \sum_{j \in S} s_j)$. Hence, for all $i \in S$,

$$y_i = \sigma_S \cdot a_i - \sum_{j \in S \setminus \{i\}} M_{j,i} = \left(\alpha + \sum_{j \in S} s_j\right) \gamma a_i - \sum_{j \in S \setminus \{i\}} \gamma s_j a_i = \gamma(\alpha + s_i) a_i.$$

In particular, the value of y_i for $i \in S$ is independent of the choice of set S. We conclude that $\mathsf{Eval}(\mathsf{pp}, [\sigma_{[\ell]}]_{\mathbb{G}}, [\ell], i) = \mathsf{Eval}(\mathsf{pp}, [\sigma_S]_{\mathbb{G}}, S, i)$, where $[\sigma_{[\ell]}]_{\mathbb{G}} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell])$ and correctness holds. □

Theorem 6.13 (Succinctness). *Construction 6.11 is succinct.*

Proof. Take any (pp, σ) in the support of $\mathsf{Setup}(1^\lambda, 1^\ell)$. For all sets S, the projected key $[\sigma_S]_{\mathbb{G}}$ output by $\mathsf{Project}(\mathsf{pp}, \sigma, S)$ consists of a single element of $\mathbb{G}$, which has size $\rho(\lambda) = \mathsf{poly}(\lambda)$. In particular, the size of the projected seed is independent of the size of the associated set S. □

Theorem 6.14 (Sampling Indistinguishability). *Construction 6.11 satisfies perfect sampling indistinguishability.*

Proof. Let $\mathcal{A}$ be an adversary for the sampling indistinguishability game. We first recall the experiments in the sampling indistinguishability security definition:

- $\mathsf{EXP}^{(0)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$: This experiment proceeds as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the input length 1^ℓ.
  2. The challenger samples $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$. Concretely, the challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda)$,$\alpha, \gamma, \delta \xleftarrow{\text{R}} \mathbb{Z}_p$, $\mathbf{a}, \mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_p^\ell$, and $\mathbf{r} \xleftarrow{\text{R}} \{0,1\}^\rho$. It computes $\mathbf{c}, \mathbf{d} \in \mathbb{Z}_p^\ell$ such that $c_i = (\gamma i + \delta) a_i$, $d_i = (\gamma i + \delta) s_i$. The challenger gives $\mathsf{pp} = (\mathcal{G}, [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, [\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}, \mathbf{r})$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a set $S \subseteq [\ell]$ and the challenger replies with $[\sigma_S]_{\mathbb{G}} \leftarrow \mathsf{Samp}(\mathsf{pp}, S)$. Specifically, the challenger samples $x \xleftarrow{\text{R}} \mathbb{Z}_p$ and sets $[\sigma_S]_{\mathbb{G}} = x \cdot [\gamma]_{\mathbb{G}}$.
  4. Algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.
- $\mathsf{EXP}^{(1)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$: Same as $\mathsf{EXP}^{(0)}_{\mathsf{samp}}$, except the challenger sets $[\sigma_S]_{\mathbb{G}} = (\alpha + \sum_{i \in S} s_i) \cdot [\gamma]_{\mathbb{G}}$.

The only difference between $\mathsf{EXP}^{(0)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$ and $\mathsf{EXP}^{(1)}_{\mathsf{samp}}(1^\lambda, \mathcal{A})$ is the distribution of $[\sigma_S]_{\mathbb{G}}$. In $\mathsf{EXP}^{(1)}_{\mathsf{samp}}$, the challenger samples $\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$ and the only element in the experiment that depends on α is $[\sigma_S]_{\mathbb{G}} = (\alpha + \sum_{i \in S} s_i) \cdot [\gamma]_{\mathbb{G}}$. Thus, the distribution of the multiplicative term $(\alpha + \sum_{i \in S} s_i)$ is uniform over $\mathbb{Z}_p$ and independent of all other quantities. This is the distribution of x in $\mathsf{EXP}^{(0)}_{\mathsf{samp}}$. Since the two experiments are identically distributed, the theorem holds. □

Theorem 6.15 (Adaptive Pseudorandomness)**.** *If the CBDH assumption holds with respect to* PrimeBGroupGen*, then Construction 6.11 satisfies adaptive pseudorandomness.*

Proof. The proof structure is very similar to the proof of Theorem 6.7. We first state the following corollary of the CBDH assumption (Assumption 6.10) and Theorem 3.2.

Lemma 6.16. *Suppose the CBDH assumption holds with respect to* PrimeBGroupGen *and let* $\mathsf{hc}\colon \{0,1\}^{\rho} \times \{0,1\}^{\rho} \to \{0,1\}$ *be the Goldreich-Levin extractor. Then, for all efficient (and possibly non-uniform) adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*:*

$$\left|\Pr\left[\mathcal{A}\left(1^{\lambda}, \mathbf{r}, \mathsf{chal}, \mathsf{hc}([(\alpha+u)vw]_{\mathbb{G}_T}, \mathbf{r})\right) = 1\right] - \Pr\left[\mathcal{A}\left(1^{\lambda}, \mathbf{r}, \mathsf{chal}, b\right) = 1\right]\right| = \mathsf{negl}(\lambda),$$

where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^{\lambda})$*,* $\alpha, u, v, w \xleftarrow{\text{R}} \mathbb{Z}_p$*,* $\mathbf{r} \xleftarrow{\text{R}} \{0,1\}^{\rho}$*,* $b \xleftarrow{\text{R}} \{0,1\}$*, and* $\mathsf{chal} = (\mathcal{G}, \alpha, [u]_{\mathbb{G}}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}})$

Proof. We start by defining a joint distribution $(X, Y) = \{(X_\lambda, Y_\lambda)\}_{\lambda \in \mathbb{N}}$ as follows:

- Sample $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^{\lambda})$ and exponents $\alpha, u, v, w \xleftarrow{\text{R}} \mathbb{Z}_p$.
- Let $x = [(\alpha+u)vw]_{\mathbb{G}_T}$ and $y = (\mathcal{G}, \alpha, [u]_{\mathbb{G}}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}})$. Output the pair (x, y).

Next, under the CBDH assumption, X is computationally unpredictable given Y. To see this, suppose there exists an efficient algorithm $\mathcal{A}$ that can predict x given y when $(x, y) \leftarrow (X_\lambda, Y_\lambda)$ with non-negligible probability ε. We use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ for the CBDH problem:

- On input the CBDH challenge $(1^{\lambda}, \mathcal{G}, [u]_{\mathbb{G}}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}})$, where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e)$, algorithm $\mathcal{B}$ samples $\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$ and sets $y = (\mathcal{G}, \alpha, [u]_{\mathbb{G}}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}})$.
- Algorithm $\mathcal{B}$ gives 1^{λ} and y to algorithm $\mathcal{A}$ to obtain an output $x = [z]_{\mathbb{G}_T}$.
- Algorithm $\mathcal{B}$ outputs $[z]_{\mathbb{G}_T} - \alpha \cdot [v]_{\mathbb{G}} \cdot [w]_{\mathbb{G}}$.

By assumption, the CBDH challenger samples $\mathcal{G} \leftarrow \mathsf{PrimeBGroupGen}(1^{\lambda})$ and $u, v, w \xleftarrow{\text{R}} \mathbb{Z}_p$. Algorithm $\mathcal{B}$ then samples $\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$, so algorithm $\mathcal{B}$ perfectly simulates the distribution of y in $(x, y) \leftarrow X_\lambda \times Y_\lambda$, where the associated value of x is $x = [(\alpha+u)vw]_{\mathbb{G}_T}$. Thus, with probability ε, algorithm $\mathcal{A}$ outputs $[z]_{\mathbb{G}_T} = [(\alpha+u)vw]_{\mathbb{G}_T}$. In this case, $z - \alpha vw = uvw$ and algorithm $\mathcal{B}$ successfully solves the CBDH problem with the same non-negligible advantage ε. We conclude that under CBDH, the distribution of X is computationally unpredictable given Y. Lemma 6.8 now follows from Theorem 3.2. □

Proof of Theorem 6.15. We now return to the proof of Theorem 6.15. For each index $i \in \{0\} \cup \mathbb{N}$, we define an experiment Hyb_i as follows:

- Hyb_i: This is a variant of the adaptive pseudorandomness experiment:
  - On input the security parameter 1^{λ}, algorithm $\mathcal{A}$ outputs the length parameter 1^{ℓ}. The challenger runs $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^{\lambda}, 1^{\ell})$. Specifically, it samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^{\lambda})$, $\alpha, \gamma, \delta \xleftarrow{\text{R}} \mathbb{Z}_p$, $\mathbf{a}, \mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_p^{\ell}$, $\mathbf{r} \xleftarrow{\text{R}} \{0,1\}^{\rho}$ and computes $\mathbf{c}, \mathbf{d} \in \mathbb{Z}_p^{\ell}$ such that $c_i = (\gamma i + \delta) a_i$, $d_i = (\gamma i + \delta) s_i$. It sets $\mathsf{pp} = (\mathcal{G}, [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, [\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}, \mathbf{r})$ and $\sigma = (\alpha, \mathbf{s})$. The challenger gives pp to $\mathcal{A}$.
  - The challenger also samples $[\sigma_{[\ell]}]_{\mathbb{G}} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell])$ and $\mathbf{t} \xleftarrow{\text{R}} \{0,1\}^{\ell}$. In particular $[\sigma_{[\ell]}]_{\mathbb{G}} = (\alpha + \sum_{j \in [\ell]} s_j) \cdot [\gamma]_{\mathbb{G}}$.
  - When $\mathcal{A}$ makes an evaluation query on $j \in [\ell]$, the challenger replies with $\mathsf{Eval}(\mathsf{pp}, [\sigma_{[\ell]}]_{\mathbb{G}}, [\ell], j)$ if $j > i$ and with t_j if $j \le i$. In particular, when $j > i$, the challenger computes $y_j = \gamma(\alpha + s_j) a_j$ and responds with $\mathsf{hc}([y_j]_{\mathbb{G}_T}, \mathbf{r})$.
  - After $\mathcal{A}$ finishes making evaluation queries, the challenger computes the seed $[\sigma_S]_{\mathbb{G}} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell] \setminus I)$ where $I \subseteq [\ell]$ is the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. In particular, $[\sigma_S]_{\mathbb{G}} = (\alpha + \sum_{j \in [\ell] \setminus I} s_j) \cdot [\gamma]_{\mathbb{G}}$. The challenger gives $[\sigma_S]_{\mathbb{G}}$ to $\mathcal{A}$.

– At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.

We again write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid Hyb_i with adversary $\mathcal{A}$. Following the same argument in the proof of Theorem 6.7, we again have $\mathsf{EXP}^{(0)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{EXP}^{(1)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_\ell(\mathcal{A})$, and that for all indices $i \in \{0\} \cup \mathbb{N}$,

$$\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i] = \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1], \tag{6.3}$$

where E_i is the event that algorithm $\mathcal{A}$ makes an evaluation query on index i. Suppose now that

$$|\Pr[\mathsf{EXP}^{(0)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) = 1]| \geq \varepsilon(\lambda) \tag{6.4}$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the distinguishing problem from Lemma 6.16:

1. On input the challenge $(1^\lambda, \mathbf{r}, (\mathcal{G}, \alpha, [u]_{\mathbb{G}}, [v]_{\mathbb{G}}, [w]_{\mathbb{G}}), b)$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e)$, algorithm $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ to obtain the output length 1^ℓ. Algorithm $\mathcal{B}$ samples an index $i \xleftarrow{\textsc{r}} [\ell]$.

2. For all $j \in [\ell] \setminus \{i\}$, algorithm $\mathcal{B}$ samples $a_j, s_j \xleftarrow{\textsc{r}} \mathbb{Z}_p$ and $t_j \xleftarrow{\textsc{r}} \{0,1\}$. It sets $[s_i]_{\mathbb{G}} = [u]_{\mathbb{G}}$, $[a_i]_{\mathbb{G}} = [v]_{\mathbb{G}}$, $[\gamma]_{\mathbb{G}} = [w]_{\mathbb{G}}$. It also samples $\delta^* \xleftarrow{\textsc{r}} \mathbb{Z}_p$ and (implicitly) sets $\delta := \delta^* - \gamma i$. It then computes vector $[\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}$ as follows

   - If $j \neq i$, it sets
   $$[c_j]_{\mathbb{G}} = [\delta^* a_j]_{\mathbb{G}} + (j - i) \cdot a_j \cdot [w]_{\mathbb{G}},$$
   $$[d_j]_{\mathbb{G}} = [\delta^* s_j]_{\mathbb{G}} + (j - i) \cdot s_j \cdot [w]_{\mathbb{G}}.$$
   - If $j = i$, it sets $[c_i]_{\mathbb{G}} = \delta^* \cdot [v]_{\mathbb{G}}$ and $[d_i]_{\mathbb{G}} = \delta^* \cdot [u]_{\mathbb{G}}$.

   Algorithm $\mathcal{B}$ sets $\mathsf{pp} = (\mathcal{G}, [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, [\mathbf{c}]_{\mathbb{G}}, [\mathbf{d}]_{\mathbb{G}}, \mathbf{r})$ and gives pp to $\mathcal{A}$.

3. When algorithm $\mathcal{A}$ makes an evaluation query on an index $j \in [\ell]$, the challenger responds as follow.
   - If $j < i$, the challenger responds with t_j.
   - If $j = i$, the challenger responds with b.
   - If $j > i$, the challenger responds with $\mathsf{hc}((\alpha + s_j)a_j \cdot [1]_{\mathbb{G}} \cdot [w]_{\mathbb{G}}, \mathbf{r})$.

4. After $\mathcal{A}$ finishes making evaluation queries, let $I \subseteq [\ell]$ be the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. If $i \notin I$, then algorithm $\mathcal{B}$ outputs 0. Otherwise algorithm $\mathcal{B}$ responds with $[\sigma_S]_{\mathbb{G}} = (\alpha + \sum_{j \in [\ell] \setminus I} s_j) \cdot [\gamma]_{\mathbb{G}}$. Since $i \in I$, this means $i \notin [\ell] \setminus I$, so algorithm $\mathcal{B}$ knows all of the exponents s_j needed to construct $[\sigma_S]_{\mathbb{G}}$.

5. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which algorithm $\mathcal{B}$ also outputs.

Since $\mathcal{A}$ is efficient, algorithm $\mathcal{B}$ is efficient, so it suffices to analyze its advantage.

Analyzing the advantage of $\mathcal{B}$. Let W_0 be the event that $\mathcal{B}$ outputs 1 when $b = \mathsf{hc}([(\alpha + u)vw]_{\mathbb{G}_T}, \mathbf{r})$ and W_1 be the event that $\mathcal{B}$ outputs 1 when $b \xleftarrow{\textsc{r}} \{0,1\}$. Suppose algorithm $\mathcal{B}$ samples $i = i^*$ in the above reduction. By construction, the challenger for the experiment in Lemma 6.16 samples $\alpha, u, v, w \xleftarrow{\textsc{r}} \mathbb{Z}_p$. First, we argue that algorithm $\mathcal{B}$ perfectly simulates the public parameters (where u plays the role of s_{i^*}, v plays the role of a_{i^*}, and w plays the role of γ) and the evaluation queries on indices $j \neq i^*$ for $\mathcal{A}$:

- Since the challenger samples $u, v, w \xleftarrow{\textsc{r}} \mathbb{Z}_p$, the distributions of $s_{i^*} = u$, $a_{i^*} = v$, and $\gamma = w$ are distributed exactly as in Hyb_{i^*-1} and Hyb_{i^*}. We conclude that the distribution of $\mathcal{G}, [\gamma]_{\mathbb{G}}, [\mathbf{a}]_{\mathbb{G}}, [\mathbf{s}]_{\mathbb{G}}, \mathbf{r}$ in the public parameters are distributed exactly as in Hyb_{i^*-1} and Hyb_{i^*}.

- Consider now the distribution of $\mathbf{c}$ and $\mathbf{d}$. Let $\delta = \delta^* - \gamma i$. Since algorithm $\mathcal{B}$ samples $\delta^* \xleftarrow{R} \mathbb{Z}_p$, the distribution of δ is uniform over $\mathbb{Z}_p$. Now, for $j \neq i$, we have

$$c_j = \delta^* a_j + (j-i)a_j w = \delta^* a_j + (j-i)a_j\gamma = \delta a_j + \gamma j a_j = (\gamma_j + \delta)a_j,$$
$$d_j = \delta^* s_j + (j-i)s_j w = \delta^* s_j + (j-i)s_j\gamma = \delta s_j + \gamma j s_j = (\gamma_j + \delta)s_j,$$

  which matches the specification in Hyb_{i^*-1} and Hyb_{i^*}. Similarly, for c_i and d_i, we have

$$c_i = \delta^* v = \delta^* a_i = (\gamma i + \delta)a_i,$$
$$d_i = \delta^* u = \delta^* s_i = (\gamma i + \delta)s_i,$$

  which again matches the specification in Hyb_{i^*-1} and Hyb_{i^*}.

- Finally, the evaluation queries on indices $j \neq i^*$ are answered exactly according to the specification of Hyb_{i^*-1} and Hyb_{i^*}. Specifically, when $j > i^*$, the challenger responds with the hard-core predicate on

$$(\alpha + s_j)a_j \cdot [1]_{\mathbb{G}} \cdot [w]_{\mathbb{G}} = [\gamma(\alpha + s_j)a_j]_{\mathbb{G}_T},$$

  which is precisely the behavior in Hyb_{i^*-1} and Hyb_{i^*}.

We consider the distribution of the challenge bit b.

- Suppose $b = \mathsf{hc}([(\alpha + u)vw]_{\mathbb{G}_T}, \mathbf{r}) = \mathsf{hc}([\gamma(\alpha + s_{i^*})a_{i^*}]_{\mathbb{G}_T}, \mathbf{r})$. In this case, the responses to the evaluation queries are distributed according to the specification in Hyb_{i^*-1}. We consider the probability that algorithm $\mathcal{B}$ outputs 1 in this case. First, if algorithm $\mathcal{A}$ does not make an evaluation query on index i^*, then algorithm $\mathcal{B}$ always outputs 0. If algorithm $\mathcal{A}$ does make an evaluation query on index i^*, then algorithm $\mathcal{B}$ perfectly simulates the challenge according to the distribution in Hyb_{i^*-1}, and thus, outputs 1 with probability $\Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}]$. Thus, in this case, algorithm $\mathcal{B}$ outputs 1 with probability

$$\Pr[\mathcal{B} \text{ outputs } 1 \mid i = i^*] = \Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}] \cdot \Pr[\mathsf{E}_{i^*}] = \Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_{i^*}].$$

- Suppose $b \xleftarrow{R} \{0,1\}$. In this case, the responses to the evaluation queries are distributed according to the specification in Hyb_{i^*}. By a similar reasoning as in the previous case, we conclude that in this case

$$\Pr[\mathcal{B} \text{ outputs } 1 \mid i = i^*] = \Pr[\mathsf{Hyb}_{i^*}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}] \cdot \Pr[\mathsf{E}_{i^*}] = \Pr[\mathsf{Hyb}_{i^*}(\mathcal{A}) = 1 \wedge \mathsf{E}_{i^*}].$$

Finally, algorithm $\mathcal{B}$ samples $i \xleftarrow{R} [\ell]$. Thus

$$\Pr[W_0 = 1] = \frac{1}{\ell} \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i]$$
$$\Pr[W_1 = 1] = \frac{1}{\ell} \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i}(\mathcal{A}) = 1 \wedge \mathsf{E}_i].$$

Since $\mathsf{EXP}^{(0)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{EXP}^{(1)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_\ell(\mathcal{A})$, we appeal to Eqs. (6.3) and (6.4) and conclude that

$$\begin{aligned}|\Pr[W_0 = 1] - \Pr[W_1 = 1]| &= \frac{1}{\ell}\left|\sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \mathsf{E}_i]\right| \\ &= \frac{1}{\ell}\left|\sum_{i \in [\ell]} p_i - p_{i-1}\right| \\ &= \frac{1}{\ell}|p_0 - p_\ell| = \frac{1}{\ell}|\Pr[\mathsf{EXP}^{(0)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) = 1]| \geq \frac{\varepsilon}{\ell},\end{aligned}$$

which is non-negligible since $\mathcal{A}$ is efficient which means $\ell = \mathsf{poly}(\lambda)$. □

6.3 Publicly-Sampleable Projective PRGs from LWE

In this section, we show how to construct a publicly-sampleable projective PRG from LWE. Our construction is essentially a translation of our construction from CDH (Section 6.1) to the LWE setting. Our construction has a similar structure as the projective PRG from LWE in [ABI+23], except we introduce an additional blinding term to support public sampling and sampling indistinguishability (similar to the modification made in the context of our CDH construction). We now give the full construction and security analysis.

Lattice preliminaries. We start by recalling some basic facts about lattice-based cryptography. Throughout this section, we associate elements $y \in \mathbb{Z}_q$ with its integer representative in the interval $(-q/2, q/2] \cap \mathbb{Z}$. For positive integers $n, q \in \mathbb{N}$, we define $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^{\mathsf{T}} \in \mathbb{Z}_q^{n \times m'}$ to be the gadget matrix [MP12] where $\mathbf{I}_n$ is the identity matrix of dimension n, $\mathbf{g}^{\mathsf{T}} = [1, 2, \dots, 2^{\lceil \log q \rceil - 1}]$, and $m' = n \lceil \log q \rceil$. When $m \geq m'$, we write $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ to denote the "padded gadget matrix" $[\mathbf{I}_n \otimes \mathbf{g}^{\mathsf{T}} \mid \mathbf{0}^{n \times (m-m')}]$. The inverse function $\mathbf{G}^{-1} \colon \mathbb{Z}_q^n \to \mathbb{Z}_q^{m'}$ expands each entry $x \in \mathbb{Z}_q$ into a column of size $\lceil \log q \rceil$ corresponding to the bits in the binary representation of x. Similarly, when $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ is a padded gadget matrix with dimension $m \geq m'$, we extend the output of $\mathbf{G}^{-1} \colon \mathbb{Z}_q^n \to \mathbb{Z}_q^m$ by zero-padding each column. By construction, for all $\mathbf{t} \in \mathbb{Z}_q^n$, it follows that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{t}) = \mathbf{t} \bmod q$. For a matrix $\mathbf{V}$ we write $\|\mathbf{V}\| = \max_{i,j} |V_{i,j}|$. For an element $y \in \mathbb{Z}_q$, we define the rounding function $\lfloor y \rceil : \mathbb{Z}_q \to \{0,1\}$ to be

$$\lfloor y \rceil = \begin{cases} 0 & |y| \leq q/4 \\ 1 & \text{otherwise.} \end{cases}$$

For functions $f = f(\lambda), g = g(\lambda)$, we write $f \leq O(g)$ if there exists a function $g'(\lambda) \in O(g(\lambda))$ such that for all $\lambda \in \mathbb{N}$, $f(\lambda) \leq g'(\lambda)$. We define $f \geq O(g)$ analogously. We write $D_{\mathbb{Z},\chi}$ to denote the discrete Gaussian distribution over $\mathbb{Z}$ with width parameter $\chi > 0$. Finally, we use the following standard Gaussian tail bound:

Lemma 6.17 (Gaussian Tail Bound [MP12, Lemma 2.6])**.** *For all $\chi > 0$ and all $\lambda \in \mathbb{N}$,*

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

Truncated discrete Gaussian. We use $\bar{D}_{\mathbb{Z},\chi,B}$ to denote the truncated discrete Gaussian distribution defined by the following sampling procedure:

- Sample $x \leftarrow D_{\mathbb{Z},\chi}$.
- If $|x| \leq B$, output x. Otherwise, output 0.

In this work, we will write $\bar{D}_{\mathbb{Z},\chi}$ to denote $\bar{D}_{\mathbb{Z},\chi,\sqrt{\lambda}\chi}$. By Lemma 6.17, the truncated discrete Gaussian distribution $\bar{D}_{\mathbb{Z},\chi}$ is statistically close to the discrete Gaussian distribution $D_{\mathbb{Z},\chi}$.

Homomorphic evaluation. Similar to [ABI+23], our projective PRG will rely on the lattice-based pseudorandom function (PRF) by Brakerski and Vaikuntanathan [BV15]. We first recall the lattice homomorphic evaluation procedure from [GSW13, BGG+14]

Theorem 6.18 (Homomorphic Encodings [GSW13, BGG+14])**.** *Let λ be a security parameter and $n = n(\lambda)$, $q = q(\lambda)$ be lattice parameters. Let $\ell = \ell(\lambda)$ be an input length. Take any $m \geq n \lceil \log q \rceil$ and let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of functions $f \colon \{0,1\}^\ell \to \{0,1\}$ that can be computed by a Boolean circuit of depth at most $d = d(\lambda)$. Then, there exists a pair of efficient and deterministic algorithms* $(\mathsf{EvalF}, \mathsf{EvalFX})$ *with the following properties:*

- $\mathsf{EvalF}(\mathbf{A}, f) \to \mathbf{A}_f$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$ and a function $f \in \mathcal{F}$, the input-independent evaluation algorithm outputs a matrix $\mathbf{A}_f \in \mathbb{Z}_q^{n \times m}$.*
- $\mathsf{EvalFX}(\mathbf{A}, f, \mathbf{x}) \to \mathbf{H}_{\mathbf{A},f,\mathbf{x}}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$, a function $f \in \mathcal{F}$, and an input $\mathbf{x} \in \{0,1\}^\ell$, the input-dependent evaluation algorithm outputs a matrix $\mathbf{H}_{\mathbf{A},f,\mathbf{x}} \in \mathbb{Z}_q^{\ell m \times m}$.*

Moreover, for all security parameters $\lambda \in \mathbb{N}$, matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$, all functions $f \in \mathcal{F}$, and all inputs $\mathbf{x} \in \{0,1\}^\ell$, the matrices $\mathbf{A}_f = \mathsf{EvalF}(\mathbf{A}, f)$ and $\mathbf{H}_{\mathbf{A},f,\mathbf{x}} = \mathsf{EvalFX}(\mathbf{A}, f, \mathbf{x})$ satisfy the following properties:

- $\|\mathbf{H}_{\mathbf{A},f,\mathbf{x}}\| \le (n \log q)^{O(d)}$.
- $(\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{A},f,\mathbf{x}} = \mathbf{A}_f - f(\mathbf{x}) \cdot \mathbf{G}$.

Learning with errors. We now recall the learning with errors (LWE) assumption [Reg05]:

Assumption 6.19 (Learning with Errors [Reg05])**.** Let λ be a security parameter, and $n = n(\lambda)$, $m = m(\lambda)$, $q = q(\lambda)$, and $\chi = \chi(\lambda)$ be lattice parameters. We say the learning with errors problem $\mathsf{LWE}_{n,m,q,\chi}$ holds if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

The Brakerski-Vaikuntanathan constrained PRF. We now describe the key-homomorphic constrained PRF by Brakerski and Vaikuntanathan [BV15] for the special case of puncturing constraints.

Theorem 6.20 (Key-Homomorphic Puncturable PRF [BV15, adapted])**.** *Let λ be a security parameter and ℓ be an input length. Let $n, m, q, \chi, \hat{\chi}$ be parameters (which are functions of λ, ℓ) and suppose $m \ge n \lceil \log q \rceil$. For every $\mathbf{x} \in \{0,1\}^\ell$, let $\delta_\mathbf{x} \colon \{0,1\}^\ell \to \{0,1\}$ be the indicator function*

$$\delta_\mathbf{x}(\mathbf{y}) := \begin{cases} 1 & \mathbf{x} = \mathbf{y} \\ 0 & \mathbf{x} \ne \mathbf{y}. \end{cases}$$

For a (public) matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$, vector $\mathbf{d} \in \mathbb{Z}_q^n$, and inputs $\mathbf{x}, \mathbf{y} \in \{0,1\}^\ell$, define

$$\mathbf{A}_\mathbf{y} = \mathsf{EvalF}(\mathbf{A}, \delta_\mathbf{y}) \quad \text{and} \quad \mathbf{H}_{\mathbf{A},\mathbf{y},\mathbf{x}} = \mathsf{EvalFX}(\mathbf{A}, \delta_\mathbf{y}, \mathbf{x}).$$

Then, the following properties hold:

- **Correctness.** *For all $\mathbf{y} \ne \mathbf{x}$, $(\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{A},\mathbf{y},\mathbf{x}} \mathbf{G}^{-1}(\mathbf{d}) = \mathbf{A}_\mathbf{y} \mathbf{G}^{-1}(\mathbf{d})$. In addition, $\|\mathbf{H}_{\mathbf{A},\mathbf{u},\mathbf{x}}\| \le m^{O(\log \ell)}$.*
- **Punctured pseudorandomness.** *For a security parameter $\lambda \in \mathbb{N}$, an adversary $\mathcal{A}$, and a bit $\beta \in \{0,1\}$, we define the punctured pseudorandomness experiment $\mathsf{EXP}_{\mathsf{PPRF}}^{(\beta)}(1^\lambda, \mathcal{A})$ as follows:*
  - *On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a point $\mathbf{x} \in \{0,1\}^\ell$.*
  - *The challenger samples the public parameters $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times \ell m}$ and $\mathbf{d} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, a PRF key $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, and error terms $\mathbf{e} \leftarrow D_{\mathbb{Z},\chi}^{\ell m}$, $\hat{e} \leftarrow D_{\mathbb{Z},\hat{\chi}}$, and sets $\mathbf{c}_\mathbf{x} = \mathbf{s}^\intercal (\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal$, $t_0 = \mathbf{s}^\intercal \mathbf{A}_\mathbf{x} \mathbf{G}^{-1}(\mathbf{d}) + \hat{e}$, and $t_1 \xleftarrow{\mathrm{R}} \mathbb{Z}_q$. The challenger gives $(\mathbf{A}, \mathbf{d}, \mathbf{c}_\mathbf{x}, t_\beta)$ to the adversary.*
  - *The adversary outputs a bit $b \in \{0,1\}$, which is the output of the experiment.*

  Suppose $\hat{\chi} \ge \lambda^{\omega(1)} \cdot \chi m^{O(\log \ell)} \ell \log \ell$. Then, under the $\mathsf{LWE}_{n,\hat{m},q,\chi}$ assumption with $\hat{m} = \mathsf{poly}(n, \log q)$, for all efficient adversaries $\mathcal{A}$, there exist a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left| \Pr[\mathsf{EXP}_{\mathsf{PPRF}}^{(0)}(1^\lambda, \mathcal{A})] - \Pr[\mathsf{EXP}_{\mathsf{PPRF}}^{(1)}(1^\lambda, \mathcal{A})] \right| = \mathsf{negl}(\lambda).$$

Construction 6.21 (Publicly-Sampleable Projective PRG from LWE)**.** Let λ be a security parameter and ℓ be an output length parameter. Let $n, m, q, \chi, \hat{\chi}, B$ be scheme parameters (which are functions of λ and ℓ) and suppose $m \ge n \lceil \log q \rceil$ and $q > 4B + 2$. For each integer $i \in [\ell]$, we associate with it a unique canonical vector $\mathbf{u}_i \in \{0,1\}^{\ell'}$ where $\ell' = \lceil \log \ell \rceil$; for example, we can take $\mathbf{u}_i$ to be the binary representation of $i - 1$. We now construct a projective PRG as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp}, \sigma)$: On input the security parameter λ and the output length $\ell \in \mathbb{N}$, the setup algorithm sets $\ell' = \lceil \log \ell \rceil$. Then, it samples public components $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times \ell' m}$ and $\mathbf{d} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$. Next, the algorithm samples $\mathbf{r}, \mathbf{s}_1, \ldots, \mathbf{s}_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$. Then it defines the set

$$\mathsf{NR}_{q,B} \coloneqq \mathbb{Z}_q \setminus \Big([\lfloor q/4 \rfloor - B, \lfloor q/4 \rfloor + B] \cup [-\lfloor q/4 \rfloor - B, -\lfloor q/4 \rfloor + B] \Big).$$

The set $\mathsf{NR}_{q,B}$ contains all elements of $\mathbb{Z}_q$ that are not contained in a rounding boundary: that is, for all $y \in \mathsf{NR}_{q,B}$ and all $e \in \mathbb{Z}$ where $|e| \le B$, it holds that $\lfloor y + e \rceil = \lfloor y \rceil$. Note that this set is non-empty as long as $q > 4B + 2$. The setup algorithm now samples $y_i \xleftarrow{\mathrm{R}} \mathsf{NR}_{q,B}$ for all $i \in [\ell]$. Finally, for each $i \in [\ell]$, the setup algorithm samples $\mathbf{e}_i \leftarrow \bar{D}_{\mathbb{Z},\chi}^{\ell' m}$ and $\hat{e}_i \leftarrow \bar{D}_{\mathbb{Z},\hat{\chi}}$. It then computes $\mathbf{A}_i = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_i})$ and

$$\mathbf{c}_i = \mathbf{s}_i^{\mathsf{T}}(\mathbf{A} - \mathbf{u}_i^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e}_i^{\mathsf{T}} \quad \text{and} \quad z_i = (\mathbf{r} + \mathbf{s}_i)^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) + \hat{e}_i + y_i.$$

The algorithm outputs

$$\mathsf{pp} = \big(\mathbf{A}, \mathbf{d}, \{\mathbf{c}_i, z_i\}_{i \in [\ell]}\big) \quad \text{and} \quad \sigma = (\mathbf{r}, \mathbf{s}_1, \ldots, \mathbf{s}_\ell).$$

- $\mathsf{Samp}(\mathsf{pp}, S) \to \sigma_S$: On input the public parameters $\mathsf{pp} = (\mathbf{A}, \mathbf{d}, \{\mathbf{c}_i, z_i\}_{i \in [\ell]})$ and a target set S, the sampling algorithm samples a random vector $\mathbf{k} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and outputs $\sigma_S = \mathbf{k}$.

- $\mathsf{Project}(\mathsf{pp}, \sigma, S) \to \sigma_S$: On input the public parameters $\mathsf{pp} = (\mathbf{A}, \mathbf{d}, \{\mathbf{c}_i, z_i\}_{i \in [\ell]})$, the seed $\sigma = (\mathbf{r}, \mathbf{s}_1, \ldots, \mathbf{s}_\ell)$, and a target set $S \subseteq [\ell]$, the projection algorithm computes and outputs $\sigma_S = \mathbf{r} + \sum_{i \in S} \mathbf{s}_i$.

- $\mathsf{Eval}(\mathsf{pp}, \sigma, S, i)$: On input the public parameters $\mathsf{pp} = (\mathbf{A}, \mathbf{d}, \{\mathbf{c}_i, z_i\}_{i \in [\ell]})$, the projected seed $\sigma = \mathbf{k}$, the associated set of indices $S \subseteq [\ell]$, and an index $i \in S$, the evaluation algorithm computes $\mathbf{A}_i = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_i})$, $\mathbf{H}_{\mathbf{A},i,j} = \mathsf{EvalFX}(\mathbf{A}, \delta_{\mathbf{u}_i}, \mathbf{u}_j)$, and

$$y_i' = z_i + \sum_{j \in S \setminus \{i\}} \mathbf{c}_j^{\mathsf{T}} \mathbf{H}_{\mathbf{A},i,j} \mathbf{G}^{-1}(\mathbf{d}) - \mathbf{k}^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}).$$

It outputs $\lfloor y_i' \rceil$.

Theorem 6.22 (Correctness). *Suppose $B \ge \sqrt{\lambda}\hat{\chi} + \sqrt{\lambda}\chi m^{O(\log\log \ell)} \ell \log \ell$. Then, Construction 6.21 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, output length $\ell \in \mathbb{N}$, any set of indices $S \subseteq [\ell]$, and index $i \in S$. Let $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$ and $\sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, S)$. Then $\mathsf{pp} = (\mathbf{A}, \mathbf{d}, \{\mathbf{c}_i, z_i\}_{i \in [\ell]})$, $\sigma = (\mathbf{r}, \mathbf{s}_1, \ldots, \mathbf{s}_\ell)$ and $\sigma_S = \mathbf{r} + \sum_{i \in S} \mathbf{s}_i$. By construction, $\mathbf{c}_i = \mathbf{s}_i^{\mathsf{T}}(\mathbf{A} - \mathbf{u}_i^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e}_i^{\mathsf{T}}$, $z_i = (\mathbf{r} + \mathbf{s}_i)^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) + \hat{e}_i + y_i$, and $\mathbf{A}_i = \mathsf{Eval}(\mathbf{A}, \delta_{\mathbf{u}_i})$. Consider $y_i = \mathsf{Eval}(\mathsf{pp}, \sigma_S, S, i)$. Since $\mathbf{e}_i$ and $\hat{e}_i$ is sampled from the truncated discrete Gaussian distributions $\bar{D}_{\mathbb{Z},\chi}^{\ell' m}$ and $\bar{D}_{\mathbb{Z},\hat{\chi}}$, we have $\|\mathbf{e}_i\| \le \sqrt{\lambda}\chi$ and $|\hat{e}_i| \le \sqrt{\lambda}\hat{\chi}$. From Theorem 6.20, we have for all $j \in S \setminus \{i\}$,

$$\mathbf{c}_j^{\mathsf{T}} \mathbf{H}_{\mathbf{A},i,j} \mathbf{G}^{-1}(\mathbf{d}) = (\mathbf{s}_j^{\mathsf{T}}(\mathbf{A} - \mathbf{u}_j^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e}_j^{\mathsf{T}}) \cdot \mathbf{H}_{\mathbf{A},i,j} \mathbf{G}^{-1}(\mathbf{d}) = \mathbf{s}_j^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) + e_j^*,$$

where

$$|e_j^*| = |\mathbf{e}_j^{\mathsf{T}} \mathbf{H}_{\mathbf{A},i,j} \mathbf{G}^{-1}(\mathbf{d})| \le \|\mathbf{e}_j\| \cdot \ell' m^{O(\log\log \ell)} \le \ell' m^{O(\log\log \ell)} \chi\sqrt{\lambda}.$$

The evaluation algorithm computes

$$\begin{aligned} y_i' &= z_i + \sum_{j \in S \setminus \{i\}} \mathbf{c}_j^{\mathsf{T}} \mathbf{H}_{\mathbf{A},i,j} \mathbf{G}^{-1}(\mathbf{d}) - \mathbf{k}^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) \\ &= (\mathbf{r} + \mathbf{s}_i)^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) + \hat{e}_i + y_i + \sum_{j \in S \setminus \{i\}} \mathbf{c}_j^{\mathsf{T}} \mathbf{H}_{\mathbf{A},i,j} \mathbf{G}^{-1}(\mathbf{d}) - \mathbf{r}^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) - \sum_{j \in S} \mathbf{s}_j^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) \\ &= y_i + \hat{e}_i + \sum_{j \in S \setminus \{i\}} e_j^*. \end{aligned}$$

Next,

$$\left| \hat{e}_i + \sum_{j \in S \setminus \{i\}} e_j^* \right| \le \sqrt{\lambda}\hat{\chi} + |S| \cdot \ell' m^{O(\log\log \ell)} \chi \sqrt{\lambda}.$$

Taking $B \ge \sqrt{\lambda}\hat{\chi} + \ell \lceil \log \ell \rceil m^{O(\log\log\ell)} \sqrt{\lambda}\chi$ and using the fact that $y_i' \in \mathsf{NR}_{q,B}$, we conclude that $\lfloor y_i' \rceil = \lfloor y_i \rceil$. Finally, this holds for all $i \in S$, independent of the choice of S. Correctness holds. □

Theorem 6.23 (Succinctness). *Suppose $n \log q = \mathrm{poly}(\lambda, \log \ell)$. Then, Construction 6.21 is succinct.*

Proof. The projected key σ_S is a vector in $\mathbb{Z}_q^n$, which can be described by $n \log q$ bits. □

Theorem 6.24 (Sampling Indistinguishability). *Suppose $q > \ell B \cdot \lambda^{\omega(1)}$. Then, Construction 6.21 satisfies statistical sampling indistinguishability.*

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

- Hyb_0: This is experiment $\mathsf{EXP}_{\mathsf{samp}}^{(0)}(1^\lambda, \mathcal{A})$. Namely, after $\mathcal{A}$ outputs the input length 1^ℓ, the challenger provides $\mathsf{pp} = (\mathbf{A}, \mathbf{d}, \{\mathbf{c}_i, z_i\}_{i \in [\ell]})$ to the adversary, where $\mathbf{c}_i^\mathsf{T} = \mathbf{s}_i^\mathsf{T}(\mathbf{A} - \mathbf{u}_i^\mathsf{T} \otimes \mathbf{G}) + \mathbf{e}_i^\mathsf{T}, z_i = (\mathbf{r} + \mathbf{s}_i)^\mathsf{T} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) + \hat{e}_i + y_i$, $\mathbf{A}_i = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_i})$, and $y_i \xleftarrow{\mathrm{R}} \mathsf{NR}_{q,B}$. Algorithm $\mathcal{A}$ then outputs a set $S \subseteq [\ell]$ and the challenger replies with $\sigma_S \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$. Algorithm $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, which is the output of the experiment.
- Hyb_1: Same as Hyb_0 except the challenger samples $y_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q$ for all $i \in [\ell]$.
- Hyb_2: Same as Hyb_1 except the challenger samples $z_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q$ for all $i \in [\ell]$. In this experiment, the distribution of each z_i is *independent* of $\mathbf{r}$.
- Hyb_3: Same as Hyb_2 except the challenger sets $\sigma_S = \mathbf{r} + \sum_{i \in S} \mathbf{s}_i$.
- Hyb_4: Same as Hyb_3 except the challenger samples $z_i = (\mathbf{r} + \mathbf{s}_i)^\mathsf{T} \mathbf{a}_i + \hat{e}_i + y_i$ for all $i \in [\ell]$.
- Hyb_5: Same as Hyb_4 except the challenger samples $y_i \xleftarrow{\mathrm{R}} \mathsf{NR}_{q,B}$ for all $i \in [\ell]$. This is experiment $\mathsf{EXP}_{\mathsf{samp}}^{(1)}(1^\lambda, \mathcal{A})$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the random variable corresponding to an output of an execution of Hyb_i with adversary $\mathcal{A}$. We now show that each pair of adjacent distributions is statistically indistinguishable.

- The only difference between $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ is the distribution of y_i. The statistical distance between the uniform distribution over $\mathbb{Z}_q$ and $\mathsf{NR}_{q,B}$ is $(4B + 2)/q$. Thus, the statistical distance between the tuple $(y_1, \ldots, y_\ell)$ in the two experiments is at most $\ell(4B + 2)/q = \mathsf{negl}(\lambda)$.
- The only difference between $\mathsf{Hyb}_1(\mathcal{A})$ and $\mathsf{Hyb}_2(\mathcal{A})$ is the distribution of z_i. In Hyb_1, the challenger samples $y_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q$ and the only element in the experiment that depends on y_i is z_i. Thus, we conclude that the distribution of z_i is uniform over $\mathbb{Z}_q$ and independent of all other quantities. This is identical to the distribution in Hyb_2.
- The only difference between $\mathsf{Hyb}_2(\mathcal{A})$ and $\mathsf{Hyb}_3(\mathcal{A})$ is the distribution of σ_S. In Hyb_3, the challenger samples $\mathbf{r} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, and the only element in the experiment that depends on $\mathbf{r}$ is σ_S. Thus, we conclude that the distribution of σ_S is uniform over $\mathbb{Z}_p$ and independent of all other quantities. This is the distribution in Hyb_2.
- $\mathsf{Hyb}_3(\mathcal{A})$ and $\mathsf{Hyb}_4(\mathcal{A})$ are identically distributed by the same argument used to argue that $\mathsf{Hyb}_1(\mathcal{A})$ and $\mathsf{Hyb}_2(\mathcal{A})$ are identically distributed.
- $\mathsf{Hyb}_4(\mathcal{A})$ and $\mathsf{Hyb}_5(\mathcal{A})$ are statistically close by the same argument used to argue statistical indistinguishability of $\mathsf{Hyb}_1(\mathcal{A})$ and $\mathsf{Hyb}_2(\mathcal{A})$.

Since each pair of adjacent distributions has negligible statistical distance, the theorem holds. □

Theorem 6.25 (Adaptive Pseudorandomness). *Suppose $n \log q = \mathrm{poly}(\lambda, \ell)$, $\hat{\chi} \ge \lambda^{\omega(1)} \cdot \chi m^{O(\log\log\ell)} \ell \log \ell$, $B \ge \sqrt{\lambda}\hat{\chi} + \sqrt{\lambda}\chi m^{O(\log\log\ell)} \ell \log \ell$, and $q > \lambda^{\omega(1)} + 4B$. Then, under the $\mathsf{LWE}_{n,\hat{m},q,\chi}$ for $\hat{m} = \mathrm{poly}(n, \log q)$. Then, Construction 6.21 satisfies adaptive pseudorandomness.*

Proof. The proof follows a similar structure as that of Theorem 6.7. We first state a corollary of the pseudorandomness property from Theorem 6.20, which is tailored to our construction.

Lemma 6.26. *For a security parameter λ, an adversary $\mathcal{A}$ and a bit $\beta \in \{0,1\}$, we define the following distinguishing game $\mathsf{EXP}^{(\beta)}(1^\lambda, \mathcal{A})$:*

- *On input the security parameter 1^λ, the adversary $\mathcal{A}$ chooses an output length 1^ℓ and an index $i \in [\ell]$.*
- *The challenger computes $\ell' = \lceil \log \ell \rceil$ and samples a set of public parameters using a similar procedure as that of the* Setup *algorithm in Construction 6.21. Specifically, the challenger proceeds as follows:*
  - *Sample $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell' m}$ and $\mathbf{d} \xleftarrow{R} \mathbb{Z}_q^n$.*
  - *Sample $\mathbf{r}, \mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \bar{D}_{\mathbb{Z},\chi}^{\ell' m}$, $\hat{e} \leftarrow \bar{D}_{\mathbb{Z},\hat{\chi}}$, and $y \xleftarrow{R} \mathsf{NR}_{q,B}$.*
  - *Compute $\mathbf{A}_i = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_i})$, $\mathbf{c} = \mathbf{s}^\intercal(\mathbf{A} - \mathbf{u}_i^\intercal \otimes \mathbf{G}) + \mathbf{e}^\intercal$, and $z = (\mathbf{r}+\mathbf{s})^\intercal \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) + \hat{e} + y$.*

  Finally, the challenger computes $b_0 = \lfloor y \rceil$, $b_1 \xleftarrow{R} \{0,1\}$, and gives $(\mathbf{A}, \mathbf{d}, \mathbf{r}, \mathbf{c}, z, b_\beta)$ to the adversary.
- *The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.*

Suppose $\hat{\chi} \geq \lambda^{\omega(1)} \cdot \chi m^{O(\log\log \ell)} \ell \log \ell$ and $n \log q = \mathrm{poly}(\lambda)$. Then, under the $\mathsf{LWE}_{n,\hat{m},q,\chi}$ assumption with $\hat{m} = \mathrm{poly}(n, \log q)$, for all efficient adversaries $\mathcal{A}$, there exist a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$,

$$\left| \Pr[\mathsf{EXP}^{(0)}(1^\lambda, \mathcal{A})] - \Pr[\mathsf{EXP}^{(1)}(1^\lambda, \mathcal{A})] \right| \leq \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an efficient distinguisher for $\mathsf{EXP}^{(0)}$ and $\mathsf{EXP}^{(1)}$. We define a sequence of hybrid experiments:

- Hyb_0: This is experiment $\mathsf{EXP}^{(0)}$, where the challenger sets $b = \lfloor y \rceil$.
- Hyb_1: Same as Hyb_0 except the challenger samples $\mathbf{e} \leftarrow D_{\mathbb{Z},\chi}^{\ell' m}$ and $\hat{e} \leftarrow D_{\mathbb{Z},\hat{\chi}}$. In particular, the errors $\mathbf{e}$ and $\hat{e}$ are sampled from the discrete Gaussian distribution rather than the truncated discrete Gaussian distribution.
- Hyb_2: Same as Hyb_1 except the challenger samples $z \xleftarrow{R} \mathbb{Z}_q$.
- Hyb_3: Same as Hyb_2 except the challenger samples $b \xleftarrow{R} \{0,1\}$.
- Hyb_4: Same as Hyb_3 except the challenger samples $z = (\mathbf{r}+\mathbf{s})^\intercal \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) + e + y$.
- Hyb_5: Same as Hyb_4 except the challenger samples $\mathbf{e} \leftarrow \bar{D}_{\mathbb{Z},\chi}^{\ell' m}$ and $\hat{e} \leftarrow \bar{D}_{\mathbb{Z},\hat{\chi}}$. Specifically, the errors $\mathbf{e}$ and $\hat{e}$ are now sampled from the *truncated* discrete Gaussian distribution. This is experiment $\mathsf{EXP}^{(1)}$.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid Hyb_i with adversary $\mathcal{A}$ (and an implicit security parameter λ). We now show that the output distributions of each adjacent pair of hybrid experiments is indistinguishable.

- The only difference between $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ is the distribution of $\mathbf{e}$ and $\hat{e}$. By Lemma 6.17, the distributions $D_{\mathbb{Z},\hat{\chi}}$ and $\bar{D}_{\mathbb{Z},\hat{\chi}}$ are statistically close to $D_{\mathbb{Z},\chi}$ and $\bar{D}_{\mathbb{Z},\chi}$. Therefore $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ are statistically indistinguishable (since $m = \mathrm{poly}(n \log q) = \mathrm{poly}(\lambda)$ and $\ell = \mathrm{poly}(\lambda)$ by the efficiency requirement of $\mathcal{A}$).
- We appeal to Theorem 6.20. Specifically, when $\hat{\chi} > \lambda^{\omega(1)} \cdot \chi m^{O(\log\log \ell)} \ell \log \ell$ and the $\mathsf{LWE}_{n,\hat{m},q,\chi}$ assumption holds, the punctured pseudorandomness property of Theorem 6.20 holds. Suppose now that $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks the punctured pseudorandomness property from Theorem 6.20.
  - On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ to obtain 1^ℓ and $i \in [\ell]$.
  - Algorithm $\mathcal{B}$ sets $\ell' = \lceil \log \ell \rceil$ and gives $1^{\ell'}$ and $\mathbf{u}_i \in \{0,1\}^{\ell'}$ to the challenger. The challenger replies with $(\mathbf{A}, \mathbf{d}, \mathbf{c}, t)$.

  - Algorithm $\mathcal{B}$ samples a vector $\mathbf{r} \xleftarrow{R} \mathbb{Z}_q^n$ and a scalar $y \xleftarrow{R} \mathsf{NR}_{q,B}$. It computes $\mathbf{A}_i = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_i})$ and $z = \mathbf{r}^\mathsf{T}\mathbf{A}_i\mathbf{G}^{-1}(\mathbf{d}) + t + y$. It send $(\mathbf{A}, \mathbf{d}, \mathbf{r}, \mathbf{c}, z, t)$ to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

  By construction, the challenger samples $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell' m}$, $\mathbf{d} \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{c} = \mathbf{s}^\mathsf{T}(\mathbf{A} - \mathbf{u}_i^\mathsf{T} \otimes \mathbf{G}) + \mathbf{e}^\mathsf{T}$, exactly as in the punctured pseudorandomness experiment. When $t = \mathbf{s}^\mathsf{T}\mathbf{A}_i\mathbf{G}^{-1}(\mathbf{d}) + \hat{e}$, then algorithm $\mathcal{B}$ perfectly simulates Hyb_1. On the other hand, when $t \xleftarrow{R} \mathbb{Z}_p$, the distribution of $z = \mathbf{r}^\mathsf{T}\mathbf{A}_i\mathbf{G}^{-1}(\mathbf{d}) + t + y$ also uniformly random since no other element in the experiment depends on t. Therefore $\mathcal{B}$ perfectly simulates Hyb_2 when $t \xleftarrow{R} \mathbb{Z}_p$. Hence $\mathcal{B}$ breaks the pseudorandomness property of Theorem 6.20 with the same non-negligible probability ε, which is a contradiction.

- The only difference between $\mathsf{Hyb}_2(\mathcal{A})$ and $\mathsf{Hyb}_3(\mathcal{A})$ is the distribution of b. In Hyb_2, the challenger sets $b = \lfloor y \rceil$ where $y \leftarrow \mathsf{NR}_{q,B}$. By construction of $\mathsf{NR}_{q,B}$, this means,

$$\left| \Pr[\lfloor y \rceil = 1 : y \xleftarrow{R} \mathsf{NR}_{q,B}] - \frac{1}{2} \right| = \frac{O(1)}{|\mathsf{NR}_{q,B}|} = \mathsf{negl}(\lambda),$$

  since $|\mathsf{NR}_{q,B}| \geq q - (4B+2) = \lambda^{\omega(1)}$. Thus, the marginal distribution of $b \in \{0,1\}$ in Hyb_2 is statistically close to uniform. Finally, the only variable in Hyb_2 that depends on y is b. We conclude that $\mathsf{Hyb}_2(\mathcal{A})$ and $\mathsf{Hyb}_3(\mathcal{A})$ are statistically indistinguishable.

- $\mathsf{Hyb}_3(\mathcal{A})$ and $\mathsf{Hyb}_4(\mathcal{A})$ are computationally indistinguishable by the same argument used to argue indistinguishability of $\mathsf{Hyb}_1(\mathcal{A})$ and $\mathsf{Hyb}_2(\mathcal{A})$.

- $\mathsf{Hyb}_4(\mathcal{A})$ and $\mathsf{Hyb}_5(\mathcal{A})$ are statistically indistinguishable by the same argument used to argue indistinguishability of $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$. $\square$

Proof of Theorem 6.25. We now return to the proof of Theorem 6.25. The proof follows a similar strategy as the proof of Theorem 6.7. For each index $i \in \{0\} \cup \mathbb{N}$, we define an experiment Hyb_i as follows:

- Hyb_i: This is a variant of the adaptive pseudorandomness experiment:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs 1^ℓ. The challenger runs $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$. Specifically, the challenger samples $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell' m}$, $\mathbf{d} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{r}, \mathbf{s}_1, \ldots, \mathbf{s}_\ell \xleftarrow{R} \mathbb{Z}_q^n$, $y_1, \ldots, y_\ell \xleftarrow{R} \mathsf{NR}_{q,B}$, $\mathbf{e}_1, \ldots, \mathbf{e}_\ell \leftarrow \bar{D}_{\mathbb{Z},\chi}^{\ell' m}$ and $\hat{e}_1, \ldots, \hat{e}_\ell \leftarrow \bar{D}_{\mathbb{Z},\hat{\chi}}$. Then, it computes $\mathbf{c}_i^\mathsf{T} = \mathbf{s}_i^\mathsf{T}(\mathbf{A} - \mathbf{u}_i^\mathsf{T} \otimes \mathbf{G}) + \mathbf{e}_i^\mathsf{T}$, $\mathbf{A}_i = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_i})$, and $z_i = (\mathbf{r} + \mathbf{s}_i)^\mathsf{T}\mathbf{A}_i\mathbf{G}^{-1}(\mathbf{d}) + \hat{e}_i + y_i$ for all $i \in [\ell]$. The challenger gives $\mathsf{pp} = (\mathbf{A}, \mathbf{d}, \{\mathbf{c}_i, z_i\}_{i \in [\ell]})$ to $\mathcal{A}$.
  - Next, the challenger computes $\sigma_{[\ell]} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell])$ and samples $\mathbf{t} \xleftarrow{R} \{0,1\}^\ell$. In particular $\sigma_{[\ell]} = \mathbf{r} + \sum_{j \in [\ell]} \mathbf{s}_j$.
  - When $\mathcal{A}$ makes an evaluation query on $j \in [\ell]$, the challenger replies with $\mathsf{Eval}(\mathsf{pp}, \sigma_{[\ell]}, [\ell], j)$ if $j > i$ and with t_j if $j \leq i$. In particular, when $j > i$, the challenger computes $\mathbf{A}_j = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_j})$ and

$$y_j' = z_j + \sum_{k \in [\ell] \setminus \{j\}} \mathbf{c}_k^\mathsf{T} \cdot \mathbf{H}_{\mathbf{A},j,k}\mathbf{G}^{-1}(\mathbf{d}) - \sigma_{[\ell]}^\mathsf{T}\mathbf{A}_j\mathbf{G}^{-1}(\mathbf{d})$$

    and responds with $\lfloor y_j' \rceil$.
  - After $\mathcal{A}$ finishes making evaluation queries, the challenger computes the seed $\sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell] \setminus I)$ where $I \subseteq [\ell]$ is the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. In particular, $\sigma_S = \mathbf{r} + \sum_{j \in [\ell] \setminus I} \mathbf{s}_j$. The challenger gives σ_S to $\mathcal{A}$.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid Hyb_i with adversary $\mathcal{A}$. By construction, observe that $\mathsf{EXP}_{\mathsf{prg}}^{(0)}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{EXP}_{\mathsf{prg}}^{(1)}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_\ell(\mathcal{A})$. Following the same argument in the proof of Theorem 6.7, for all indices $i \in \{0\} \cup \mathbb{N}$,

$$\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i] = \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1], \tag{6.5}$$

where E_i is the event that algorithm $\mathcal{A}$ makes an evaluation query on index i. Suppose now that

$$|\Pr[\mathsf{EXP}_{\mathsf{prg}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{prg}}^{(1)}(1^\lambda, \mathcal{A}) = 1]| \geq \varepsilon(\lambda) \tag{6.6}$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the distinguishing problem from Lemma 6.26:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ to obtain the output length 1^ℓ. Algorithm $\mathcal{B}$ samples a random index $i \xleftarrow{\text{R}} [\ell]$ and send $(1^\ell, i)$ to the challenger to receive $(\mathbf{A}, \mathbf{d}, \mathbf{r}, \mathbf{c}, z, b)$.

2. For all $j \in [\ell] \setminus \{i\}$, algorithm $\mathcal{B}$ samples $\mathbf{s}_j \xleftarrow{\text{R}} \mathbb{Z}_q^n$, $t_j \xleftarrow{\text{R}} \{0,1\}$, $y_j \xleftarrow{\text{R}} \mathsf{NR}_{q,B}$, $\mathbf{e}_j \leftarrow \bar{D}_{\mathbb{Z},\chi}^{\ell' m}$, $\hat{e}_j \leftarrow \bar{D}_{\mathbb{Z},\hat{\chi}}$ and computes $\mathbf{c}_j^{\mathsf{T}} = \mathbf{s}_j^{\mathsf{T}}(\mathbf{A} - \mathbf{u}_j^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e}_j^{\mathsf{T}}$, $\mathbf{A}_j = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_j})$, and $z_j = (\mathbf{r} + \mathbf{s}_j)^{\mathsf{T}} \mathbf{A}_j \mathbf{G}^{-1}(\mathbf{d}) + \hat{e}_j + y_j$. Algorithm $\mathcal{B}$ also sets $\mathbf{c}_i = \mathbf{c}$, $z_i = z$. Algorithm $\mathcal{B}$ gives $\mathsf{pp} = (\mathbf{A}, \mathbf{d}, \{\mathbf{c}_j, z_j\}_{j \in [\ell]})$ to $\mathcal{A}$.

3. When algorithm $\mathcal{A}$ makes an evaluation query on an index $j \in [\ell]$, the challenger responds as follow.
   - If $j < i$, the challenger responds with t_j.
   - If $j = i$, the challenger responds with b.
   - If $j > i$, the challenger responds with $\lfloor y_j \rceil$.

4. After $\mathcal{A}$ finishes making evaluation queries, let $I \subseteq [\ell]$ be the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. If $i \notin I$, then algorithm $\mathcal{B}$ outputs 0. Otherwise algorithm $\mathcal{B}$ responds with $\sigma_S = \mathbf{r} + \sum_{j \in [\ell] \setminus I} \mathbf{s}_j$. Since $i \in I$, this means $i \notin [\ell] \setminus I$, so algorithm $\mathcal{B}$ knows all of the $\mathbf{s}_j$ needed to construct σ_S.

5. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which algorithm $\mathcal{B}$ also outputs.

Since $\mathcal{A}$ is efficient, algorithm $\mathcal{B}$ is efficient, so it suffices to analyze its advantage.

Analyzing the advantage of $\mathcal{B}$. By construction, $\mathbf{c} = \mathbf{s}^{\mathsf{T}}(\mathbf{A} - \mathbf{u}_i^{\mathsf{T}} \otimes \mathbf{G}) + \mathbf{e}^{\mathsf{T}}$ and $z = (\mathbf{r} + \mathbf{s})^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) + \hat{e} + y$. In the following analysis, let

$$\sigma_{[\ell]} = \mathbf{r} + \mathbf{s} + \sum_{k \in [\ell] \setminus \{i\}} \mathbf{s}_k.$$

Note that algorithm $\mathcal{B}$ does not (and cannot) compute $\sigma_{[\ell]}$; however, it will be useful in our analysis below. Let W_0 be the event that $\mathcal{B}$ outputs 1 when the challenger sets $b = \lfloor y \rceil$ and W_1 be the event that $\mathcal{B}$ outputs 1 when the challenger samples $b \xleftarrow{\text{R}} \{0,1\}$. Suppose algorithm $\mathcal{B}$ samples $i = i^*$ in the above reduction. Then, the following holds:

- By definition, algorithm $\mathcal{B}$ perfectly simulates the public parameters pp for algorithm $\mathcal{A}$ as well as the evaluation queries for all $j < i^*$ according to the specification in hybrid Hyb_{i^*-1} and Hyb_{i^*}.

- Consider the evaluation queries for $j > i^*$

- In Hyb_{i^*-1} and Hyb_{i^*}, the challenger would first compute $\mathbf{A}_j = \mathsf{EvalF}(\mathbf{A}, \delta_{\mathbf{u}_j})$ and

$$y_j' = z_j + \sum_{k \in [\ell] \setminus \{j\}} \mathbf{c}_k^{\mathsf{T}} \mathbf{H}_{\mathbf{A},j,k} \mathbf{G}^{-1}(\mathbf{d}) - \sigma_{[\ell]}^{\mathsf{T}} \mathbf{A}_j \mathbf{G}^{-1}(\mathbf{d}).$$

  By the same analysis as in the proof of Theorem 6.22, $\lfloor y_j' \rceil = \lfloor y_j \rceil$, and we conclude that algorithm $\mathcal{B}$ perfectly simulates the evaluation queries for $j > i^*$ according to the specification of Hyb_{i^*-1} and Hyb_{i^*}.

Next, we consider the distribution of the challenge bit b:

- Suppose $b = \lfloor y \rceil$. By the same argument as in the proof of Theorem 6.22, we have

$$\lfloor y \rceil = \left\lfloor z + \sum_{k \in [\ell] \setminus \{i\}} \mathbf{c}_k^{\mathsf{T}} \mathbf{H}_{\mathbf{A},i,k} \mathbf{G}^{-1}(\mathbf{d}) - \sigma_{[\ell]}^{\mathsf{T}} \mathbf{A}_i \mathbf{G}^{-1}(\mathbf{d}) \right\rceil .$$

  In this case, the response to an evaluation query on index i is distributed according to the specification in Hyb_{i^*-1}. We consider the probability that algorithm $\mathcal{B}$ outputs 1 in this case. First, if algorithm $\mathcal{A}$ does not make an evaluation query on index i^*, then algorithm $\mathcal{B}$ always outputs 0. If algorithm $\mathcal{A}$ does make an evaluation query on index i^*, then algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_{i^*-1}, and thus, outputs 1 with probability $\Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}]$. Thus, in this case, algorithm $\mathcal{B}$ outputs 1 with probability

$$\Pr[\mathcal{B} \text{ outputs } 1 \mid i = i^*] = \Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}] \cdot \Pr[\mathsf{E}_{i^*}] = \Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_{i^*}].$$

- Suppose $b \xleftarrow{\mathrm{R}} \{0,1\}$. In this case, the responses to the evaluation queries are distributed according to the specification in Hyb_{i^*}. By a similar reasoning as in the previous case, we conclude that in this case

$$\Pr[\mathcal{B} \text{ outputs } 1 \mid i = i^*] = \Pr[\mathsf{Hyb}_{i^*}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}] \cdot \Pr[\mathsf{E}_{i^*}] = \Pr[\mathsf{Hyb}_{i^*}(\mathcal{A}) = 1 \wedge \mathsf{E}_{i^*}].$$

Finally, algorithm $\mathcal{B}$ samples $i \xleftarrow{\mathrm{R}} [\ell]$. Thus

$$\begin{aligned} \Pr[W_0 = 1] &= \frac{1}{\ell} \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i], \\ \Pr[W_1 = 1] &= \frac{1}{\ell} \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i}(\mathcal{A}) = 1 \wedge \mathsf{E}_i]. \end{aligned}$$

Since $\mathsf{EXP}_{\mathrm{prg}}^{(0)}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{EXP}_{\mathrm{prg}}^{(1)}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_\ell(\mathcal{A})$, we appeal to Eqs. (6.5) and (6.6) and conclude that

$$\begin{aligned} |\Pr[W_0 = 1] - \Pr[W_1 = 1]| &= \frac{1}{\ell} \left| \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \mathsf{E}_i] \right| \\ &= \frac{1}{\ell} \left| \sum_{i \in [\ell]} p_i - p_{i-1} \right| \\ &= \frac{1}{\ell} |p_0 - p_\ell| = \frac{1}{\ell} |\Pr[\mathsf{EXP}_{\mathrm{prg}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathrm{prg}}^{(1)}(1^\lambda, \mathcal{A}) = 1]| \geq \frac{\varepsilon}{\ell}, \end{aligned}$$

which is non-negligible since $\mathcal{A}$ is efficient which means $\ell = \mathrm{poly}(\lambda)$. □

Parameter instantiations. We now show how to instantiate the parameters for Construction 6.21 to satisfy the requirements in Theorems 6.22 to 6.25. Let λ be a security parameter and ℓ be the output length. We instantiate the lattice parameters as follows:

- We set the lattice dimension to be $n = (\lambda \log \ell)^{1/\varepsilon}$ for some small constant $\varepsilon \in (0,1)$. In the following, we will assume $\log q = \tilde{O}(\lambda \log \ell)$, where $\tilde{O}(\cdot)$ suppresses $\mathrm{poly}(\log \lambda, \log \log \ell)$ factors. Next, we set $m = O(n \log q) = \mathrm{poly}(\lambda, \log \ell)$.
- We set $\chi = \mathrm{poly}(\lambda)$. We set $\hat{\chi} = 2^\lambda \cdot \chi m^{O(\log \log \ell)} \ell \log \ell = 2^{\tilde{O}(\lambda \log \ell)}$. We set $B = 2\sqrt{\lambda}\hat{\chi} = 2^{\tilde{O}(\lambda \log \ell)}$
- Finally, we set $q = 2^\lambda \cdot \ell B = 2^{\tilde{O}(\lambda \log \ell)} = 2^{\tilde{O}(n^\varepsilon)}$ so that the $\mathsf{LWE}_{n,\hat{m},q,\chi}$ assumption holds where $\hat{m} = \mathrm{poly}(n, \log q)$.

This yields a projective PRG from polynomial hardness of LWE with a sub-exponential modulus-to-noise ratio.

7 Nearly-Linear Semi-Static Broadcast Encryption from Bilinear Maps

In this section, we show how to construct a semi-statically secure broadcast encryption with a nearly-linear-size public key from a search assumption (in the target group). The construction combines ideas from the semi-static broadcast encryption scheme from [GW09, §3.1] with the recent cross-term-compression technique from [GLWW24]. Security relies on the search version of the set-consistent bilinear Diffie-Hellman exponent assumption from [GLWW24, Assumption 4.2]. We first recall the notion of a progression-free and double-free set as well as the set-consistent bilinear Diffie Hellman exponent assumption [GLWW24].

Progression-free sets. A progression-free set [ET36] is a set of natural numbers that does not contain any arithmetic progressions of length 3. Similar to the applications from [GLWW24], we also require the set system to be double-free, which means the set does not contain any integer which is twice that of another integer. The work of [GLWW24] describes a simple way to convert any progression-free set into a progression-free and double-free set. We provide the formal definitions below:

Definition 7.1 (Progression-Free Set [ET36])**.** A set $\mathcal{D} \subset \mathbb{N}$ is progression-free if for all $i, j, k \in \mathcal{D}$ where $i \neq j$, it follows that $i + j \neq 2 \cdot k$.

Definition 7.2 (Double-Free Set [GLWW24])**.** A set $\mathcal{D} \subset \mathbb{N}$ is double-free if for all $i, j \in \mathcal{D}$, it holds that $i \neq 2 \cdot j$.

Lemma 7.3 (Progression-Free and Double-Free Sets [Beh46, Elk10, GLWW24])**.** *There exists a family of progression-free and double-free sets $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ where $|\mathcal{D}_n| = n$ and $\max(\mathcal{D}_n) = n^{1+o(1)}$. Moreover, there exists an explicit and efficient algorithm that takes as input 1^n and outputs $\mathcal{D}_n$.*

Set-consistent bilinear Diffie-Hellman exponent assumption. We prove security from a search version of the set-consistent bilinear Diffie-Hellman exponent assumption from [GLWW24]. While [GLWW24, Assumption 4.2] formulates the assumption as a *decisional* assumption in the target group, for our applications, it suffices to use the search version of the assumption. The formulation in [GLWW24] gives out some additional group elements compared to our formulation. We exclude these additional elements to simplify the statement of the assumption as they are not needed for our construction. We now give the formal statement of the assumption:

Assumption 7.4 (Search Set-Consistent Bilinear Diffie-Hellman Exponent [GLWW24, Assumption 4.2, adapted])**.** Let $\mathsf{PrimeBGroupGen}$ be a prime-order bilinear group generator. For a security parameter λ and an adversary $\mathcal{A}$, we define the search computational q-set-consistent bilinear Diffie-Hellman exponent experiment $q\text{-}\mathsf{SC\text{-}BDHE}(1^\lambda, \mathcal{A})$ as follow:

- On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a parameter 1^q and two sets $S_0, S_1 \subseteq [2q]$.
- The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda)$ and random exponents $a, t \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. The challenger gives $(\mathcal{G}, [t]_\mathbb{G}, \{[a^j]_\mathbb{G}\}_{j\in S_0}, \{[t \cdot a^j]_\mathbb{G}\}_{j\in S_1}, [a^q]_{\mathbb{G}_T})$ to the adversary.
- The adversary outputs a group element $[Z]_{\mathbb{G}_T} \in \mathbb{G}_T$. The experiment outputs 1 if $[Z]_{\mathbb{G}_T} = [t \cdot a^q]_{\mathbb{G}_T}$ and outputs 0 otherwise.

We say an adversary is admissible if it outputs a triple (q, S_0, S_1) where for every $s_0 \in S_0 \cup \{0\}$ and $s_1 \in S_1 \cup \{0\}$, it holds that $s_0 + s_1 \neq q$. We say the search q-set-consistent bilinear Diffie-Hellman exponent assumption holds with respect to $\mathsf{PrimeBGroupGen}$ if for every efficient and admissible adversary $\mathcal{A}$, there exist a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr[q\text{-}\mathsf{SC\text{-}BDHE}(1^\lambda, \mathcal{A}) = 1] = \mathsf{negl}(\lambda).$$

Semi-static broadcast encryption. We now give our construction of a semi-static broadcast encryption scheme from the search set-consistent bilinear Diffie-Hellman exponent assumption. As noted before, our construction can be viewed as a combination of the semi-static broadcast encryption scheme from [GW09, §3.1] with the cross-term compression techniques based on progression-free sets from [GLWW24]. At a high level, our construction replaces the cross-terms $h_j^{r_i}$ in the [GW09] construction with $h_j = [\beta a^{d_j}]_\mathbb{G}$ and $r_i = a^{d_i}$, where the exponents d_i are drawn from a progression free set $\mathcal{D}_N$. This way, there are many pairs of indices (i, j) that share a common cross-term $h_j^{r_i} = [\beta a^{d_i + d_j}]_{\mathbb{G}}$ (e.g., every i, j with a common value of $d_i + d_j$ share a cross term). At the same time, the non-cross-terms terms $h_i^{r_i} = [\beta a^{2d_i}]_{\mathbb{G}}$ remain hidden by the progression-free structure of $\mathcal{D}_N$. We give the construction below:

Construction 7.5 (Semi-Static Broadcast Encryption)**.** Let PrimeBGroupGen be a prime-order bilinear group generator and let $\rho = \rho(\lambda)$ be a bound on the bit-length of the group elements associated with PrimeBGroupGen. Let $\mathsf{hc}\colon \{0,1\}^\rho \times \{0,1\}^\rho \to \{0,1\}$ be the Goldreich-Levin hardcore extractor. Let $\{\mathcal{D}_n\}_{n \in \mathbb{N}}$ be the efficiently-computable family of progression-free and double-free sets from Lemma 7.3. We construct a broadcast encryption scheme $\Pi_{\mathsf{BE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the number of parties N, the setup algorithm samples a prime-order pairing group $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda)$ and sets $\mathcal{D} = \mathcal{D}_N$ to be a progression-free and double-free set of size N. It then computes the set $\mathcal{E}$ of all distinct pairwise sums
$$\mathcal{E} = \{d_i + d_j \mid i, j \in [N] : i \neq j\}.$$
Let $|\mathcal{E}| = M$ and denote the distinct elements of $\mathcal{E}$ by $e_1, e_2, \ldots, e_M \in \mathcal{E}$. We define a canonical function $f\colon [N] \times [N] \to [M]$ where $e_{f(i,j)} = d_i + d_j$ for all $i \neq j$. Finally, the setup algorithm samples random exponents $\alpha, \beta, a, t \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and compute vectors $\mathbf{u} = (a^{d_1}, \ldots, a^{d_N})$, $\mathbf{v} = (a^{e_1}, \ldots, a^{e_M})$. Finally, the algorithm samples the extractor seed $\mathbf{r} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and outputs
$$\mathsf{mpk} = (\mathcal{G}, \mathcal{D}, [\beta]_{\mathbb{G}}, [\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{v}]_{\mathbb{G}}, [\alpha]_{\mathbb{G}_T}, \mathbf{r}) \quad \text{and} \quad \mathsf{msk} = (\mathcal{G}, \alpha, \beta, \mathbf{u}).$$
- $\mathsf{KeyGen}(\mathsf{msk}, i)$: On input the master secret key $\mathsf{msk} = (\mathcal{G}, \alpha, \beta, \mathbf{u})$ and an index $i \in [N]$, the key-generation algorithm outputs the secret key $\mathsf{sk}_i = (i, [\alpha - \beta u_i^2]_{\mathbb{G}})$.
- $\mathsf{Enc}(\mathsf{mpk}, S, \mu)$: On input the master public key $\mathsf{mpk} = (\mathcal{G}, \mathcal{D}, [\beta]_{\mathbb{G}}, [\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{v}]_{\mathbb{G}}, [\alpha]_{\mathbb{G}_T}, \mathbf{r})$, a set $S \subseteq [N]$, and a message $\mu \in \{0,1\}$, the encryption algorithm samples a random $t \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and computes
$$C_1 = \mathsf{hc}(t \cdot [\alpha]_{\mathbb{G}_T}, \mathbf{r}) \oplus \mu, \quad [C_2]_{\mathbb{G}_T} = [t]_{\mathbb{G}}, \quad [C_3]_{\mathbb{G}_T} = t \cdot \sum_{i \in S} [\beta u_i]_{\mathbb{G}}.$$
The encryption algorithm outputs the ciphertext $\mathsf{ct} = (C_1, [C_2]_{\mathbb{G}_T}, [C_3]_{\mathbb{G}_T})$.
- $\mathsf{Dec}(\mathsf{mpk}, S, \mathsf{sk}, \mathsf{ct})$: On input the master public key $\mathsf{mpk} = (\mathcal{G}, \mathcal{D}, [\beta]_{\mathbb{G}}, [\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{v}]_{\mathbb{G}}, [\alpha]_{\mathbb{G}_T}, \mathbf{r})$, a set $S \subseteq [N]$, a secret key $\mathsf{sk} = (i, [\gamma]_{\mathbb{G}})$ and a ciphertext $\mathsf{ct} = (C_1, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}})$, the decryption algorithm computes
$$[Z]_{\mathbb{G}_T} = [C_2]_{\mathbb{G}} \cdot [\gamma]_{\mathbb{G}} + [C_3]_{\mathbb{G}} \cdot [u_i]_{\mathbb{G}} - \sum_{j \in S \setminus \{i\}} [C_2]_{\mathbb{G}} \cdot [\beta v_{f(i,j)}]_{\mathbb{G}}.$$
Then it outputs $\mu = C_1 \oplus \mathsf{hc}([Z]_{\mathbb{G}_T}, \mathbf{r})$.

Theorem 7.6 (Correctness)**.** *Construction 7.5 is correct.*

Proof. Take any $\lambda, N \in \mathbb{N}$ any set $S \subseteq [N]$, any index $i \in S$, and any message $\mu \in \{0,1\}$. Take any $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, any secret key $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$, and any $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, \mu)$. By construction,
$$\mathsf{mpk} = (\mathcal{G}, \mathcal{D}, [\beta]_{\mathbb{G}}, [\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{v}]_{\mathbb{G}}, [\alpha]_{\mathbb{G}_T}, \mathbf{r}) \quad \text{and} \quad \mathsf{msk} = (\mathcal{G}, \alpha, \beta, \mathbf{u}),$$
$\mathsf{sk}_i = (i, [\gamma]_{\mathbb{G}})$ where $\gamma = \alpha - \beta u_i^2$, and $\mathsf{ct} = (C_1, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}})$. By construction of Setup, $u_i = a^{d_i}$ and $v_i = a^{e_i}$. Since $e_{f(i,j)} = d_i + d_j$, this means $v_{f(i,j)} = a^{d_i + d_j} = u_i u_j$. Now, by construction of Setup, KeyGen, and Enc, we can write
$$\begin{aligned} Z &= C_2 \cdot \gamma + C_3 \cdot u_i - \sum_{j \in S \setminus \{i\}} C_2 \cdot \beta v_{f(i,j)} \\ &= t(\alpha - \beta u_i^2) + t \sum_{j \in S} \beta u_j u_i - \sum_{j \in S \setminus \{i\}} \beta t v_{f(i,j)} \\ &= t\alpha + \sum_{j \in S} t\beta u_i u_j - t\beta u_i^2 - \sum_{j \in S \setminus \{i\}} t\beta u_i u_j \\ &= t\alpha. \end{aligned}$$

In particular, this means

$$C_1 \oplus \mathsf{hc}([Z]_{\mathbb{G}_T}, \mathbf{r}) = \mathsf{hc}([t\alpha]_{\mathbb{G}_T}, \mathbf{r}) \oplus \mu \oplus \mathsf{hc}([Z]_{\mathbb{G}_T}, \mathbf{r}) = \mu,$$

and correctness holds. □

Theorem 7.7 (Semi-Static Security)**.** *Suppose the search set-consistent bilinear Diffie-Hellman exponent assumption holds with respect to* PrimeBGroupGen*. Then Construction 7.5 is semi-statically secure (Definition 3.4).*

Proof. We first state a corollary of the search set-consistent bilinear Diffie-Hellman exponent assumption (Assumption 7.4) and Theorem 3.2.

Lemma 7.8. *Let* PrimeBGroupGen *be a prime-order bilinear group generator and let* $\mathsf{hc}\colon \{0,1\}^\rho \times \{0,1\}^\rho \to \{0,1\}$ *be the Goldreich-Levin extractor. For a security parameter λ and an adversary $\mathcal{A}$, define the following distinguishing game* $\mathsf{EXP}^{(\beta)}(1^\lambda, \mathcal{A})$*, which can be viewed as a decisional version of the q-*SC-BDHE *game:*

- *On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a parameter 1^q and two sets $S_0, S_1 \subseteq [2q]$.*
- *The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda)$ and random exponents $s, a, t \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Additionally, the challenger samples extractor seed $\mathbf{r} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and computes bits $b_0 = \mathsf{hc}([sta^q]_{\mathbb{G}_T}, \mathbf{r})$ and $b_1 \xleftarrow{\mathrm{R}} \{0,1\}$. The challenger provides $(\mathcal{G}, s, [t]_{\mathbb{G}}, \{[a^j]_{\mathbb{G}}\}_{j \in S_0}, \{[t \cdot a^j]_{\mathbb{G}}\}_{j \in S_1}, [a^q]_{\mathbb{G}_T}, \mathbf{r}, b_\beta)$ to the adversary.*
- *The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.*

*Similar to the q-*SC-BDHE *game, we say an adversary is admissible if it outputs a triple (q, S_0, S_1) where for every $s_0 \in S_0 \cup \{0\}$ and $s_1 \in S_1 \cup \{0\}$, it holds that $s_0 + s_1 \neq q$. Suppose the q-*SC-BDHE *assumption holds with respect to* PrimeBGroupGen*. Then, for all efficient and admissible adversaries $\mathcal{A}$, there exist a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{EXP}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}(1^\lambda, \mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose there exists an efficient and admissible adversary $\mathcal{A}$ such that

$$|\Pr[\mathsf{EXP}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}(1^\lambda, \mathcal{A}) = 1]| = \varepsilon(\lambda)$$

for some non-negligible ε. For simplicity, we decompose $\mathcal{A}$ into two algorithms $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, where $\mathcal{A}_0$ takes as input 1^λ and outputs $(1^q, S_0, S_1)$ along with a private state st, and $\mathcal{A}_1$ takes as input the private state st along with the challenger's response and outputs the bit b'. We now define a joint distribution $(X, Y) = \{(X_\lambda, Y_\lambda)\}_{\lambda \in \mathbb{N}}$ as follows:

- Run $\mathcal{A}_0(1^\lambda)$ to obtain a triple $(1^q, S_0, S_1)$ and private state st. Then, sample a bilinear group $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda)$ along with exponents $s, a, t \xleftarrow{\mathrm{R}} \mathbb{Z}_p$.
- Let $x = [sta^q]_{\mathbb{G}_T}$ and $y = (\mathrm{st}, \mathcal{G}, s, [t]_{\mathbb{G}}, \{[a^j]_{\mathbb{G}}\}_{j \in S_0}, \{[t \cdot a^j]_{\mathbb{G}}\}_{j \in S_1})$. Output the pair (x, y).

We now show that under the q-SC-BDHE assumption, X is computationally unpredictable given Y. Suppose there exist an efficient algorithm $\mathcal{A}'$ that can predict x given y with some non-negligible probability ε'. We use $\mathcal{A}'$ to construct an efficient algorithm $\mathcal{B}$ for the q-SC-BDHE problem:

- On input the security parameter 1^λ, algorithm $\mathcal{B}$ first runs $\mathcal{A}_0(1^\lambda)$ to obtain a triple $(1^q, S_0, S_1)$ and state st. It forwards $(1^q, S_0, S_1)$ to the q-SC-BDHE challenger.
- On receiving the q-SC-BDHE challenge $(\mathcal{G}, [t]_{\mathbb{G}}, \{[a^j]_{\mathbb{G}}\}_{j \in S_0}, \{[t \cdot a^j]_{\mathbb{G}}\}_{j \in S_1}, [a^q]_{\mathbb{G}_T})$, algorithm $\mathcal{B}$ samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and set $y = (\mathrm{st}, \mathcal{G}, s, [t]_{\mathbb{G}}, \{[a^j]_{\mathbb{G}}\}_{j \in S_0}, \{[t \cdot a^j]_{\mathbb{G}}\}_{j \in S_1}, [a^q]_{\mathbb{G}_T})$.
- Algorithm $\mathcal{B}$ runs $\mathcal{A}'(1^\lambda, y)$ and obtains an output $x = [z]_{\mathbb{G}_T}$.
- Algorithm $\mathcal{B}$ outputs $s^{-1} \cdot [z]_{\mathbb{G}_T}$.

By construction, algorithm $\mathcal{B}$ perfectly simulates the distribution of y in $(x, y) \leftarrow (X_\lambda, Y_\lambda)$, where $x = [sta^q]_{\mathbb{G}_T}$. Thus, with probability ε', algorithm $\mathcal{A}'$ outputs $[z]_{\mathbb{G}_T} = [sta^q]_{\mathbb{G}_T}$. In this case, algorithm $\mathcal{B}$ outputs $s^{-1} \cdot [sta^q]_{\mathbb{G}_T} = [ta^q]_{\mathbb{G}}$, successfully solving the q-SC-BDHE challenge. Finally, $\mathcal{B}$ is admissible as long as $\mathcal{A}$ is admissible. This contradicts the q-SC-BDHE assumption. We conclude that under the q-SC-BDHE assumption, X is computationally unpredictable given Y. The lemma now follows by Theorem 3.2:

- Since X is computationally unpredictable given Y, Theorem 3.2 now states that the distributions $(1^\lambda, \mathbf{r}, \mathsf{hc}(x, \mathbf{r}), y)$ and $(1^\lambda, \mathbf{r}, b, y)$ are computationally indistinguishable when $\mathbf{r} \xleftarrow{R} \{0,1\}^\rho$, $(x, y) \leftarrow (X_\lambda, Y_\lambda)$, and $b \xleftarrow{R} \{0,1\}$.
- If algorithm $\mathcal{A}$ distinguishes $\mathsf{EXP}^{(0)}$ and $\mathsf{EXP}^{(1)}$ with advantage ε, then we can use $\mathcal{A}_1$ to construct a distinguisher for the distributions $(1^\lambda, \mathbf{r}, \mathsf{hc}(x, \mathbf{r}), y)$ and $(1^\lambda, \mathbf{r}, b, y)$ as follows:
  - On input $(1^\lambda, \mathbf{r}, b, y)$ where $y = (\mathsf{st}, \mathcal{G}, s, [t]_{\mathbb{G}}, \{[a^j]_{\mathbb{G}}\}_{j \in S_0}, \{[t \cdot a^j]_{\mathbb{G}}\}_{j \in S_1}, [a^q]_{\mathbb{G}_T})$, compute and output $\mathcal{A}_1(\mathsf{st}, \mathcal{G}, s, [t]_{\mathbb{G}}, \{[a^j]_{\mathbb{G}}\}_{j \in S_0}, \{[t \cdot a^j]_{\mathbb{G}}\}_{j \in S_1}, [a^q]_{\mathbb{G}_T}, \mathbf{r}, b)$.

  When $b = \mathsf{hc}(x, \mathbf{r})$, this process perfectly simulates an execution of $\mathsf{EXP}^{(0)}$ for $\mathcal{A}_1$ and when $b \xleftarrow{R} \{0,1\}$, this perfectly simulates an execution of $\mathsf{EXP}^{(1)}$ for $\mathcal{A}_1$. Thus, this algorithm distinguishes between $(1^\lambda, \mathbf{r}, \mathsf{hc}(x, \mathbf{r}), y)$ and $(1^\lambda, \mathbf{r}, b, y)$ with non-negligible advantage ε, which is a contradiction. □

Proof of Theorem 7.7. We now return to the proof of Theorem 7.7. We first define a hybrid experiment $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$, which is almost identical to $\mathsf{EXP}_{\mathsf{SSBE}}^{(\nu)}$, but in the challenge query, after computing $\mathsf{ct} = (C_1, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}}) \leftarrow \mathsf{Enc}(\mathsf{mpk}, S, \mu)$, replace C_1 with $C_1' \xleftarrow{R} \{0,1\}$ and output $\mathsf{ct}' = (C_1', [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}})$. Suppose there exist $\nu \in \{0,1\}$ and an efficient adversary $\mathcal{A}$ where

$$\left|\Pr[\mathsf{EXP}_{\mathsf{SSBE}}^{(\nu)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}(1^\lambda, \mathcal{A}) = 1]\right| = \varepsilon(\lambda).$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the distinguishing problem from Lemma 7.8.

- On input the security parameter 1^λ, run $\mathcal{A}(1^\lambda)$ to obtain the number of user 1^N and a set $S^* \subseteq [N]$.
- Let $\mathcal{D} = \mathcal{D}_N$ be the progression-free and double-free set of size N and let $\mathcal{E} = \{d_i + d_j \mid i, j \in [N] : i \neq j\}$ be the set of pairwise sums. Let $f\colon [N] \times [N] \to [M]$ be the canonical function where $e_{f(i,j)} = d_i + d_j$ for all $i \neq j$. Let $d_{\max} = \max(\mathcal{D})$ and set $q = 4d_{\max} + 2$.
- Let $C = \{2d_i - d_j : i \in [N] \setminus S^*, j \in S^*\}$ and define

$$S_0 = [2q] \setminus (C \cup \{q\})$$
$$S_1 = \{q - c : c \in C\} \cup [q+1, 2q].$$

  Algorithm $\mathcal{B}$ gives $(1^q, S_0, S_1)$ to the challenger and receives $(\mathcal{G}, s, [t]_{\mathbb{G}}, \{[a^j]_{\mathbb{G}}\}_{j \in S_0}, \{[t \cdot a^j]_{\mathbb{G}}\}_{j \in S_1}, [a^q]_{\mathbb{G}_T}, \mathbf{r}, b)$.
- Algorithm $\mathcal{B}$ samples $s' \xleftarrow{R} \mathbb{Z}_p$ and implicitly set $\alpha = sa^q$ and $\beta = s'a^{q+1} + \sum_{i \in [N \setminus S^*]} sa^{q - 2d_i}$. It then computes
  - $[\alpha]_{\mathbb{G}_T} = s \cdot [a^q]_{\mathbb{G}_T}$;
  - $[\beta]_{\mathbb{G}} = s' \cdot [a^{q+1}]_{\mathbb{G}} + \sum_{j \in [N] \setminus S^*} s \cdot [a^{q-2d_j}]_{\mathbb{G}}$;
  - for all $i \in [N]$, $[u_i]_{\mathbb{G}} = [a^{d_i}]_{\mathbb{G}}$ and $[\beta u_i]_{\mathbb{G}} = s' \cdot [a^{q+1+d_i}]_{\mathbb{G}} + \sum_{j \in [N] \setminus S^*} s \cdot [a^{q-2d_j+d_i}]_{\mathbb{G}}$;
  - for all $i \in [M]$, $[\beta v_i]_{\mathbb{G}} = s' \cdot [a^{q+1+e_i}]_{\mathbb{G}} + \sum_{j \in [N] \setminus S^*} s \cdot [a^{q-2d_j+e_i}]_{\mathbb{G}}$.

  Algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with $\mathsf{mpk} = (\mathcal{G}, \mathcal{D}, [\beta]_{\mathbb{G}}, [\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{u}]_{\mathbb{G}}, [\beta\mathbf{v}]_{\mathbb{G}}, [\alpha]_{\mathbb{G}_T}, \mathbf{r})$.
- For each key-generation query $i \in [N] \setminus S^*$ from $\mathcal{A}$, the algorithm $\mathcal{B}$ computes

$$[\gamma_i]_{\mathbb{G}} = -s' \cdot [a^{q+1+2d_i}]_{\mathbb{G}} - \sum_{j \in [N] \setminus (S^* \cup \{i\})} s \cdot [a^{q-2d_j+2d_i}]_{\mathbb{G}} \tag{7.1}$$

  and sends $\mathsf{sk}_i = [\gamma_i]_{\mathbb{G}}$ to $\mathcal{A}$.

- When $\mathcal{A}$ makes a challenge query $S \subseteq S^*$, algorithm $\mathcal{B}$ sets

$$C_1 = b \oplus v, \quad [C_2]_{\mathbb{G}} = [t]_{\mathbb{G}}, \quad [C_3]_{\mathbb{G}} = \sum_{k \in S} \left(s' \cdot [ta^{q+1+d_k}]_{\mathbb{G}} + \sum_{j \in [N] \setminus S^*} s \cdot [ta^{q-2d_j+d_k}]_{\mathbb{G}} \right) \tag{7.2}$$

  and replies to $\mathcal{A}$ with $\mathsf{ct} = (C_1, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}})$.

- At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which algorithm $\mathcal{B}$ also outputs.

We first argue that $\mathcal{B}$ is an admissible algorithm.

- Since $\mathcal{D}$ is double free, $0 \notin C$. Thus, by definition $q \notin S_0$, $q \notin S_1$.
- Furthermore, for all $y \in S_1 \cap [q-1]$, y is of the form $y = q - c$ for some $c \in C$. By definition of S_0, $c \notin S_0$. Hence, there does not exist $s_0 \in S_0$ and $s_1 \in S_1$ where $q = s_0 + s_1$.

We conclude that $\mathcal{B}$ is admissible. Now we show that $\mathcal{B}$ correctly simulates either $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$ or $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$.

- Algorithm $\mathcal{B}$ sets $\alpha = sa^q$ and $\beta = s'a^{q+1} + \sum_{j \in [N] \setminus S^*} sa^{q-2d_j}$. Since the challenger samples $s \xleftarrow{\text{R}} \mathbb{Z}_p$ and algorithm $\mathcal{B}$ samples $s' \xleftarrow{\text{R}} \mathbb{Z}_p$, both α and β are independently uniform in $\mathbb{Z}_p$ as long as $a^q, a^{q+1} \neq 0$. Since the challenger samples $a \xleftarrow{\text{R}} \mathbb{Z}_p$, with probability at least $1 - (q+1)/p$, it will be the case that a^q and a^{q+1} are both non-zero. Finally, $q = \mathsf{poly}(N) = \mathsf{poly}(\lambda)$, so with overwhelming probability over the choice of a, the values of α, β are distributed as in $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$ and $\mathsf{EXP}_{\mathsf{SSBE}}^{(\beta)}$.
- We now consider each component of the master public key and show that they can be constructed from components in the challenge and that they are correctly distributed. First, by definition $\max(C) \leq 2d_{\max} < q/2$. Thus $[q/2, q-1] \cup [q+1, 2q] \subseteq S_0$. We now consider each component in the master public key:
  - First, we argue that algorithm $\mathcal{B}$ can compute $[\beta]_{\mathbb{G}}$ from the elements in the challenge. First, $q - 2d_i \in S_0$ for all $i \in [N]$ because $d_i \in \mathbb{N}$ and $q - 2d_i > q/2$. We also have $q+1 \in S_0$. Therefore all of the components $[a^{q-2d_i}]_{\mathbb{G}}$ and $[a^{q+1}]_{\mathbb{G}}$ are included in the challenge so algorithm $\mathcal{B}$ can successfully construct $[\beta]_{\mathbb{G}}$. As argued previously, the distribution of β is uniform over $\mathbb{Z}_p$ with overwhelming probability because algorithm $\mathcal{B}$ samples $s' \xleftarrow{\text{R}} \mathbb{Z}_p$.
  - Consider the elements $[u_i]_{\mathbb{G}}$. We first show that $d_i \notin C$ which means $d_i \in S_0$. Suppose that $d_i \in C$. This means $d_i = 2d_j - d_k$ for some $j \neq k$. We consider a few sub-cases:
    * If $d_i \neq d_j, d_k$, then $d_i + d_k = 2d_j$, which contradicts the assumption that $\mathcal{D}$ is progression-free.
    * If $d_i = d_j$, then $d_j = d_k$ which is a contradiction since $j \neq k$ and the elements of $\mathcal{D}$ are distinct.
    * If $d_i = d_k$, then again we have $d_j = d_k$, which contradicts the requirement $j \neq k$.

    We conclude that $d_i \notin C$, which means $d_i \in S_0$. Thus, algorithm $\mathcal{B}$ correctly simulates $[u_i]_{\mathbb{G}}$ for all $i \in [N]$.
  - Consider the elements $[\beta u_i]_{\mathbb{G}}$. First, for all $j \neq i$, we have $q - 2d_j + d_i > q - 2d_{\max} > q/2$. Moreover, $q - 2d_j + d_i \neq q$ since $\mathcal{D}$ is double-free. This means $q - 2d_j + d_i \in S_0$. We also have $q+1+d_i \in [q+1, 2q] \subset S_0$. Thus, algorithm $\mathcal{B}$ simulates these terms exactly as in $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$ and $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$.
  - Finally, consider $[\beta v_i]_{\mathbb{G}}$. First, for all $j \neq i$, $q - 2d_j + e_i > q - 2d_{\max} > q/2$. Now, we show that $q - 2d_j + e_i \neq q$. By construction $e_i = d_{i_1} + d_{i_2}$ for some distinct $i_1, i_2 \in [N]$. We consider several possibilities:
    * Suppose $j \neq i_1, i_2$. In this case, if $q - 2d_j + e_i = q$, then $d_{i_1} + d_{i_2} = 2d_j$, which contradicts the assumption that $\mathcal{D}$ is progression-free.
    * Suppose $j = i_1$ or $j = i_2$. In this case, if $q - 2d_j + e_i = q$, then $d_{i_1} = d_{i_2}$, which contradicts the fact that i_1 and i_2 are distinct indices.

    We conclude that $q - 2d_j + e_i > q/2$ and $q - 2d_j + e_i \neq q$. In this case, $q - 2d_j + e_i \in S_0$, so algorithm $\mathcal{B}$ correctly simulates these terms exactly as in $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$ and $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$.

We conclude that all of the components in the master public key are constructed according to the specification of $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$ and $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$.

- Next, consider the key-generation queries. First, we argue that algorithm $\mathcal{B}$ can compute Eq. (7.1) using the terms from the challenge. Consider Eq. (7.1) for some index $i \in [N] \setminus S^*$. First, $q - 2d_j + 2d_i > q - 2d_{\max} > q/2$. Moreover, when $j \neq i$, we have that $q - 2d_j + 2d_i \neq q$, so this means $q - 2d_j + 2d_i \in S_0$. Similarly, $q + 1 + 2d_i \in [q+1, 2q] \subset S_0$. Thus, algorithm $\mathcal{B}$ is able to compute $[\gamma_i]_{\mathbb{G}}$ from the challenge components. Furthermore, from Eq. (7.1) and using the fact that $i \notin S^*$, we can write

$$\begin{aligned}\gamma_i &= -s' a^{q+1+2d_i} - \sum_{j \in [N] \setminus (S^* \cup \{i\})} s a^{q-2d_j+2d_i} \\ &= s a^q - s' a^{q+1+2d_i} - \sum_{j \in [N] \setminus S^*} s a^{q-2d_j+2d_i} \\ &= s a^q - \left(s' a^{q+1} + \sum_{j \in [N] \setminus S^*} s a^{q-2d_j} \right) a^{2d_i} = \alpha - \beta u_i^2 .\end{aligned}$$

  We conclude that algorithm $\mathcal{B}$ simulates the secret keys exactly as in $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$ and $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$.

- Finally, consider the challenge ciphertext. As usual, we start by showing that algorithm $\mathcal{B}$ can compute Eq. (7.2) using the terms from the challenge. Observe that for all $k \in S \subseteq S^*$ and all $j \in [N] \setminus S^*$, we have that $2d_j - d_k \in C$ by construction. Thus $q - 2d_j + d_k \in S_1$. We also have $q + 1 + 2d_k \in [q+1, 2q] \subset S_1$. Hence $\mathcal{B}$ receives all $[t a^{q-2d_j+d_k}]_{\mathbb{G}}$ and $[t a^{q+1+d_k}]_{\mathbb{G}}$ from the challenge and can simulate C_3 as described. Since the challenger samples $t \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, the distribution of C_2 is exactly as in $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$ and $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$. Consider the distribution of C_3. By definition, we have

$$\begin{aligned}C_3 &= \sum_{k \in S} \left(s' \cdot t a^{q+1+d_k} + \sum_{j \in [N] \setminus S^*} s t a^{q-2d_j+d_k} \right) \\ &= \sum_{k \in S} t a^{d_k} \left(s' a^{q+1} + \sum_{j \in [N] \setminus S^*} s a^{q-2d_j} \right) \\ &= \sum_{k \in S} t a^{d_k} \beta = t \sum_{k \in S} \beta u_k ,\end{aligned}$$

  which is exactly the distribution in $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$ and $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$. Finally, consider the distribution of C_1:

  - If $b = \mathsf{hc}([s t a^q]_{\mathbb{G}_T}, \mathbf{r}) = \mathsf{hc}([t\alpha]_{\mathbb{G}_T}, \mathbf{r})$, then $C_1 = b \oplus v$, which is the distribution of C_1 in $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$.
  - If $b \xleftarrow{\mathrm{R}} \{0,1\}$, $C_1 = b \oplus v$ is uniformly random, which is the distribution in $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$.

Thus, depending on the distribution of b (and with overwhelming probability over the choice of the challenge), algorithm $\mathcal{B}$ either simulates an execution of $\mathsf{EXP}_{\mathsf{SSBE}}^{(v)}$ or $\mathsf{EXP}_{\mathsf{SSBE}}^{(\mathsf{rand})}$. Thus, we conclude that $\mathcal{B}$ succeeds in distinguishing the distributions in Lemma 7.8 with advantage $\varepsilon(\lambda) - \mathsf{negl}(\lambda)$. As argued above, algorithm $\mathcal{B}$ is also admissible, which concludes the proof of semi-static security. □

Instantiations. Instantiating the double-free and progression-free set family in Construction 7.5 with Lemma 7.3, we obtain a semi-statically-secure broadcast encryption scheme where the public parameters contain $N^{1+o(1)}$ group elements and the secret keys and ciphertexts contain a constant number of group elements. Security follows from a *search* assumption. We summarize the instantiation below:

Corollary 7.9 (Semi-Statically-Secure Broadcast Encryption Scheme)**.** *Let N be the number of users. Under the search set-consistent bilinear Diffie-Hellman exponent assumption, there exists a semi-statically-secure broadcast encryption where the public keys contain $N^{1+o(1)}$ group elements, the secret keys contain one group element, and the ciphertexts contain two group elements.*

Acknowledgments

We thank Hoeteck Wee for several insightful discussions on broadcast encryption. Brent Waters is supported by NSF CNS-1908611, CNS-2318701, and a Simons Investigator award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[ABI+23] Benny Applebaum, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tianren Liu, and Vinod Vaikuntanathan. Succinct computational secret sharing. In *STOC*, 2023.

[AT24] Nuttapong Attrapadung and Junichi Tomida. A modular approach to registered ABE for unbounded predicates. In *CRYPTO*, 2024.

[AWY20] Shweta Agrawal, Daniel Wichs, and Shota Yamada. Optimal broadcast encryption from LWE and pairings in the standard model. In *TCC*, 2020.

[AY20] Shweta Agrawal and Shota Yamada. Optimal broadcast encryption from pairings and LWE. In *EUROCRYPT*, 2020.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

[Beh46] F. Behrend. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences*, 32(12), 1946.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, 2005.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1), 2003.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC*, 2015.

[BV22] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-inspired broadcast encryption and succinct ciphertext-policy ABE. In *ITCS*, 2022.

[BW06] Dan Boneh and Brent Waters. A fully collusion resistant broadcast, trace, and revoke system. In *ACM CCS*, 2006.

[BW10] Xavier Boyen and Brent Waters. Shrinking the keys of discrete-log-type lossy trapdoor functions. In *ACNS*, 2010.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.

[BWZ14] Dan Boneh, Brent Waters, and Mark Zhandry. Low overhead broadcast encryption from multilinear maps. In *CRYPTO*, 2014.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CGW15] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system ABE in prime-order groups via predicate encodings. In *EUROCRYPT*, 2015.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. *IACR Cryptol. ePrint Arch.*, 2025. Available at https://eprint.iacr.org/2025/044.pdf.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[DF02] Yevgeniy Dodis and Nelly Fazio. Public key broadcast encryption for stateless receivers. In *ACM CCS*, 2002.

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

[Elk10] Michael Elkin. An improved construction of progression-free sets. In *SODA*, 2010.

[ET36] Paul Erdös and Paul Turán. On some sequences of integers. *Journal of the London Mathematical Society*, 1(4), 1936.

[FN93] Amos Fiat and Moni Naor. Broadcast encryption. In *CRYPTO*, 1993.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In *CRYPTO*, 2023.

[GKSW10] Sanjam Garg, Abishek Kumarasubramanian, Amit Sahai, and Brent Waters. Building efficient fully collusion-resilient traitor tracing and revocation schemes. In *ACM CCS*, 2010.

[GKW18] Romain Gay, Lucas Kowalczyk, and Hoeteck Wee. Tight adaptively secure broadcast encryption with short ciphertexts and keys. In *SCN*, 2018.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *STOC*, 1989.

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS size in registered ABE systems. In *CRYPTO*, 2024.

[GQWW19] Rishab Goyal, Willy Quach, Brent Waters, and Daniel Wichs. Broadcast and trace with n^ε ciphertext size from standard assumptions. In *CRYPTO*, 2019.

[GST04] Michael T. Goodrich, Jonathan Z. Sun, and Roberto Tamassia. Efficient tree-based revocation in groups of low-state devices. In *CRYPTO*, 2004.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[GVW19] Rishab Goyal, Satyanarayana Vusirikala, and Brent Waters. Collusion resistant broadcast and trace from positional witness encryption. In *PKC*, 2019.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In *EUROCRYPT*, 2009.

[HLR07] Chun-Yuan Hsiao, Chi-Jen Lu, and Leonid Reyzin. Conditional computational entropy, or toward separating pseudoentropy from compressibility. In *EUROCRYPT*, 2007.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

[HS02] Dani Halevy and Adi Shamir. The LSD broadcast encryption scheme. In *CRYPTO*, 2002.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, 2012.

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

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO*, 2001.

[NP00] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In *Financial Cryptography*, 2000.

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

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2), 1978.

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In *CRYPTO*, 2022.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-io from evasive LWE. In *ASIACRYPT*, 2022.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO*, 2009.

[Wee21] Hoeteck Wee. Broadcast encryption with size $n^{1/3}$ and more from k-lin. In *CRYPTO*, 2021.

[Wee22] Hoeteck Wee. Optimal broadcast encryption and CP-ABE from evasive lattice assumptions. In *EUROCRYPT*, 2022.

[Wee24] Hoeteck Wee. Circuit ABE with $\mathsf{poly}(\mathsf{depth}, \lambda)$-sized ciphertexts and keys from lattices. In *CRYPTO*, 2024.

[Wee25] Hoeteck Wee. Almost optimal KP and CP-ABE for circuits from succinct LWE. In *EUROCRYPT*, 2025.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS*, 2010.

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via predicate encodings. In *ASIACRYPT*, 2023.

A Distributed Broadcast Encryption

In this section, we give the definition of distributed broadcast encryption along with the semi-static to adaptive transformation for distributed broadcast encryption schemes. The construction is almost identical to the construction for centralized broadcast encryption given in Construction 5.1. We start by recalling the definition of distributed broadcast encryption.

Definition A.1 (Distributed Broadcast Encryption [BZ14, KMW23]). A distributed broadcast encryption scheme is a tuple of efficient algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{pp}$: On input the security parameter λ and the number of users N, the setup algorithm outputs the public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: On input the public parameters pp and an index $i \in [N]$, the key-generation algorithm outputs a public key and a secret key $(\mathsf{pk}_i, \mathsf{sk}_i)$.

- $\mathsf{IsValid}(\mathsf{pp}, i, \mathsf{pk}_i) \to b$: On input the public parameters pp, an index $i \in [N]$, and a public key pk_i, the validity-checking algorithm outputs a bit $b \in \{0, 1\}$.
- $\mathsf{Enc}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mu) \to \mathsf{ct}$: On input the public parameters pp, a collection of public keys pk_i and a message $\mu \in \{0, 1\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mathsf{ct}, (j, \mathsf{sk}_j)) \to \mu$: On input the public parameters pp, a collection of public keys pk_i, a ciphertext ct, and a secret key sk_j for an index j, the decryption algorithm outputs a message $\mu \in \{0, 1\}$.

We require that (Setup, KeyGen, IsValid, Enc, Dec) satisfy the following properties:

- **Correctness:** For all $\lambda, N \in \mathbb{N}$, $i \in [N]$, all $S \subseteq [N]$ such that $i \in S$, all pp in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp}, i)$, all $\{\mathsf{pk}_j\}_{j \in S \setminus \{i\}}$ such that $\mathsf{IsValid}(\mathsf{pp}, j, \mathsf{pk}_j) = 1$ for all $j \in S \setminus \{i\}$, and all $\mu \in \{0, 1\}$, we have
$$\Pr[\mathsf{Dec}(\mathsf{pp}, \{(j, \mathsf{pk}_k)\}_{j \in S}, \mathsf{ct}, (i, \mathsf{sk}_i)) = \mu \ : \ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \{(j, \mathsf{pk}_j)\}_{i \in S}, \mu)] = 1.$$
- **Verifiable keys:** For all $\lambda, N \in \mathbb{N}$, and all indices $i \in [N]$, it holds that
$$\Pr\left[\mathsf{IsValid}(\mathsf{pp}, i, \mathsf{pk}_i) = 1 : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N) \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i) \end{array}\right] = 1.$$
- **Succinctness:** There exists a fixed polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda, N \in \mathbb{N}$, all subsets $S \subseteq [N]$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda 1^N)$, all key-pairs $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp}, i)$ for $i \in S$, all messages $\mu \in \{0, 1\}$, and all ciphertexts ct in the support of $\mathsf{Enc}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in S}, \mu, S)$, it holds that $|\mathsf{ct}| \le o(|S|) \cdot \mathsf{poly}(\lambda, \log N)$.

We also define adaptive security and semi-static security as follows:

- **Adaptive security:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $\beta \in \{0, 1\}$, we define the adaptive security experiment $\mathsf{EXP}^{(\beta)}_{\mathsf{DBE}}(1^\lambda, \mathcal{A})$:
  - **Setup:** On input the security parameter 1^λ, the adversary outputs the number of users 1^N. The challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and sends pp to $\mathcal{A}$. The challenger also initializes two (initially empty) sets Q and C to keep track of the set of identities on which $\mathcal{A}$ makes a key-generation query and the set of identities on which $\mathcal{A}$ makes a corruption query, respectively.
  - **Query phase:** The adversary $\mathcal{A}$ can (adaptively) make the following queries.
    * **Key generation queries:** On input index $i \in [N]$, if $i \in Q$, the query is invalid and the challenger responds with $\perp$. Otherwise, the challenger samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$, sends pk_i to $\mathcal{A}$, and add i to Q.
    * **Key corruption queries:** On input index $i \in [N]$, if $i \notin Q \setminus C$, the query is invalid and the challenger responds with $\perp$. Otherwise the challenger responds with sk_i and add i to C.
  - **Challenge query:** After $\mathcal{A}$ finishes making evaluation queries, it sends a set $S \subseteq Q \setminus C$ to the challenger. The challenger then computes $\mathsf{ct}_\beta \leftarrow \mathsf{Enc}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in S}, \beta)$ and sends ct_β to $\mathcal{A}$.
  - **Output:** At the end of the game, $\mathcal{A}$ outputs $b \in \{0, 1\}$, which is the output of the experiment.

  We say that Π_{DBE} satisfies adaptive security if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\left|\Pr[\mathsf{EXP}^{(0)}_{\mathsf{DBE}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}_{\mathsf{DBE}}(1^\lambda, \mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$
- **Semi-static security:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $\beta \in \{0, 1\}$, we define the semi-static security experiment $\mathsf{EXP}^{(\beta)}_{\mathsf{SSDBE}}(1^\lambda, \mathcal{A})$:

- **Setup:** On input the security parameter 1^λ, the adversary outputs the number of users 1^N, along with a set S^*. The challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and sends pp to $\mathcal{A}$. Next, the challenger samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$ for all $i \in S^*$ and sends $\{\mathsf{pk}_i\}_{i \in S^*}$ to $\mathcal{A}$.
- **Challenge query:** The adversary $\mathcal{A}$ sends a set $S \subseteq S^*$ to the challenger. The challenger computes $\mathsf{ct}_\beta \leftarrow \mathsf{Enc}(\mathsf{pp}, \{\mathsf{pk}_i\}_{i \in S}, \beta)$ and sends ct_β to $\mathcal{A}$.
- **Output:** At the end of the game, $\mathcal{A}$ outputs $b \in \{0, 1\}$, which is the output of the experiment.

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

$$\left| \Pr[\mathsf{EXP}^{(0)}_{\mathsf{SSDBE}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}_{\mathsf{SSDBE}}(1^\lambda, \mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Semi-static to adaptive security. We now show how to use publicly-sampleable projective PRGs to generically compile any semi-statically-secure distributed broadcast encryption scheme into an adaptively-secure one. Our approach is an adaptation of Construction 5.1 to the setting of distributed broadcast encryption. Previously, [KMW23] showed how to apply the Gentry-Waters compiler [GW09] to achieve an analogous transformation in the *random oracle* model. Our approach is the plain-model analog where we replace the random oracle with a publicly-sampleable projective PRG.

Construction A.2 (Adaptively-Secure Distributed Broadcast Encryption)**.** Let $\Pi_{\mathsf{SS}} = (\mathsf{SS.Setup}, \mathsf{SS.KeyGen}, \mathsf{SS.IsValid}, \mathsf{SS.Enc}, \mathsf{SS.Dec})$ be a semi-statically-secure distributed broadcast encryption scheme. Let $\Pi_{\mathsf{pPRG}} = (\mathsf{pPRG.Setup}, \mathsf{pPRG.Samp}, \mathsf{pPRG.Project}, \mathsf{pPRG.Eval})$ be a publicly-sampleable projective PRG. We construct an adaptively-secure broadcast encryption $\Pi_{\mathsf{DBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the number of users ℓ, the setup algorithm proceeds as follows:
  1. Samples $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$.
  2. Sample $\mathsf{SS.pp} \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$. For ease of exposition, we index the set $[2N]$ using a pair $(i, b) \in [\ell] \times \{0, 1\}$.
  3. Output $\mathsf{pp} = (\mathsf{SS.pp}, \mathsf{pPRG.pp})$.
- $\mathsf{KeyGen}(\mathsf{pp}, i)$: On input the public parameters $\mathsf{pp} = (\mathsf{SS.pp}, \mathsf{pPRG.pp})$, the key-generation algorithm samples $(\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.sk}_{(i,0)}) \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.pp}, (i, 0))$ and $(\mathsf{SS.pk}_{(i,1)}, \mathsf{SS.sk}_{(i,1)}) \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.pp}, (i, 1))$. It then samples a random bit $s_i \xleftarrow{\text{R}} \{0, 1\}$ and output key pair $\mathsf{pk}_i = (\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.pk}_{(i,1)})$, $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{(i,s_i)})$.
- $\mathsf{IsValid}(\mathsf{pp}, i, \mathsf{pk}_i)$: On input the public parameters $\mathsf{pp} = (\mathsf{SS.pp}, \mathsf{pPRG.pp})$, an index $i \in [N]$, and a public key $\mathsf{pk}_i = (\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.pk}_{(i,1)})$ the validity-checking algorithm output 1 if $\mathsf{SS.IsValid}(\mathsf{SS.pp}, (i, 0), \mathsf{SS.pk}_{(i,0)}) = 1$ and $\mathsf{SS.IsValid}(\mathsf{SS.pp}, (i, 1), \mathsf{SS.pk}_{(i,1)}) = 1$. The algorithm outputs 0 otherwise.
- $\mathsf{Enc}(\mathsf{pp}, \{i, \mathsf{pk}_i\}_{i \in S}, \mu)$: On input the public parameters $\mathsf{pp} = (\mathsf{SS.pp}, \mathsf{pPRG.pp})$, and a set $S \subseteq [N]$ of public keys $\mathsf{pk}_i = (\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.pk}_{(i,1)})$, and a message $\mu \in \{0, 1\}$, the encryption algorithm proceeds as follows:
  1. Sample $\sigma_S \leftarrow \mathsf{pPRG.Samp}(\mathsf{pp}, S)$.
  2. For each $i \in S$, compute $t_i = \mathsf{pPRG.Eval}(\mathsf{pPRG.pp}, \sigma_S, S, i)$.
  3. Compute ciphertexts
  $$\begin{aligned} \mathsf{SS.ct}_0 &\leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, \{(i, t_i), \mathsf{SS.pk}_{(i,t_i)}\}_{i \in S}, \mu), \\ \mathsf{SS.ct}_1 &\leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, \{(i, 1 - t_i), \mathsf{SS.pk}_{(i,1-t_i)}\}_{i \in S}, \mu). \end{aligned}$$
  Output the ciphertext $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$.
- $\mathsf{Dec}(\mathsf{pp}, \{i, \mathsf{pk}_i\}_{i \in S}, \mathsf{ct}, (j, \mathsf{sk}_j))$: On input the public parameters $\mathsf{pp} = (\mathsf{SS.pp}, \mathsf{pPRG.pp})$, a set $S \subseteq [N]$ of public keys $\mathsf{pk}_i = (\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.pk}_{(i,1)})$, the secret key $\mathsf{sk}_j = (s_j, \mathsf{SS.sk}_{(j,s_j)})$ for some $j \in S$, and a ciphertext $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma)$, the decryption algorithm proceeds as follows:

1. For each $i \in S$, compute $t_i = \text{pPRG.Eval}(\text{pPRG.pp}, \sigma, S, i)$. Let $b = t_j \oplus s_j$.
2. Finally, compute and output $\text{SS.Dec}(\text{SS.pp}, \{(i, t_i \oplus b), \text{SS.pk}_{(i,t_i \oplus b)}\}_{i \in S}, \text{SS.sk}_{(j,s_j)}, \text{SS.ct}_b)$.

Theorem A.3 (Correctness). *If Π_{SS} is correct, then Construction A.2 is correct.*

Proof. Take any $\lambda, N \in \mathbb{N}$. Take any set $S \subseteq [N]$ and index $i \in S$, any message $\mu \in \{0,1\}$. Let $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}(1^\lambda, 1^N)$, $\text{sk}_i \leftarrow \text{KeyGen}(\text{msk}, i)$, and $\text{ct} \leftarrow \text{Enc}(\text{mpk}, S, \mu)$. By construction, the following hold:

- First, $\text{pp} = (\text{SS.mpk}, \text{pPRG.pp})$, where $\text{SS.pp} \leftarrow \text{SS.Setup}(1^\lambda, 1^{2N})$ and $(\text{pPRG.pp}, \sigma) \leftarrow \text{pPRG.Setup}(1^\lambda, 1^N)$.
- Next, $\text{pk}_i = (\text{SS.pk}_{(i,0)}, \text{SS.pk}_{(i,1)})$, $\text{sk}_i = (i, s_i, \text{SS.sk}_{(i,s_i)})$ where $s_i \xleftarrow{\text{R}} \{0,1\}$ and

$$\begin{aligned}(\text{SS.pk}_{(i,0)}, \text{SS.sk}_{(i,0)}) &\leftarrow \text{SS.KeyGen}(\text{SS.pp}, (i,0)) \\ (\text{SS.pk}_{(i,1)}, \text{SS.sk}_{(i,1)}) &\leftarrow \text{SS.KeyGen}(\text{SS.pp}, (i,1)).\end{aligned}$$

- Finally, $\text{ct} = (\text{SS.ct}_0, \text{SS.ct}_1, \sigma_S)$, where

$$\begin{aligned}\sigma_S &\leftarrow \text{pPRG.Samp}(\text{pPRG.pp}, S) \\ \forall i \in S : t_i &= \text{pPRG.Eval}(\text{pPRG.pp}, \sigma_S, S, i) \\ \text{SS.ct}_0 &\leftarrow \text{SS.Enc}(\text{SS.mpk}, \{(i, t_i)\text{SS.pk}_{(i,t_i)}\}_{i \in S}, \mu) \\ \text{SS.ct}_1 &\leftarrow \text{SS.Enc}(\text{SS.mpk}, \{(i, 1-t_i), \text{SS.pk}_{(i,1-t_i)}\}_{i \in S}, \mu).\end{aligned}$$

Consider now the value of $\text{Dec}(\text{pp}, \{i, \text{pk}_i\}_{i \in S}, \text{ct}, (j, \text{sk}_j))$. By construction, $(j, t_j \oplus b) = (j, t_j \oplus (t_j \oplus s_j)) = (j, s_j)$. Hence by the correctness of Π_{SS}, this means that with probability 1,

$$\mu = \text{SS.Dec}(\text{SS.pp}, \{(i, t_i \oplus b), \text{SS.pk}_{(i,t_i \oplus b)}\}_{i \in S}, \text{SS.sk}_{(j,s_j)}, \text{SS.ct}_b),$$

and correctness follows. □

Theorem A.4 (Verifiable Keys). *If Π_{SS} has verifiable keys, then so does Construction A.2.*

Proof. By construction, the public key in Π_{DBE} consists of two public keys of Π_{SS}. The property now follows by the verifiable keys property of the underlying scheme:

$$\begin{aligned}&\Pr\left[\text{IsValid}(\text{pp}, i, \text{pk}_i) = 1 : \begin{array}{c}\text{pp} \leftarrow \text{Setup}(1^\lambda, 1^N) \\ (\text{pk}_i, \text{sk}_i) \leftarrow \text{KeyGen}(\text{pp}, i)\end{array}\right] \\ = &\Pr\left[\begin{array}{c}\text{SS.IsValid}(\text{SS.pp}, (i,0), \text{SS.pk}_{(i,0)}) = 1 \\ \text{and } \text{SS.IsValid}(\text{SS.pp}, (i,1), \text{SS.pk}_{(i,1)}) = 1\end{array} : \begin{array}{c}\text{SS.pp} \leftarrow \text{SS.Setup}(1^\lambda, 1^{2N}) \\ (\text{SS.pk}_{i,0}, \text{SS.sk}_{i,0}) \leftarrow \text{SSKeyGen}(\text{SS.pp}, (i,0)) \\ (\text{SS.pk}_{i,1}, \text{SS.sk}_{i,0}) \leftarrow \text{SSKeyGen}(\text{SS.pp}, (i,1))\end{array}\right] = 1.\end{aligned}$$

The claim follows. □

Theorem A.5 (Adaptive Security). *Suppose Π_{SS} satisfies semi-static security and Π_{pPRG} satisfies correctness, sampling indistinguishability, and adaptive pseudorandomness. Then Construction A.2 is adaptively secure.*

Proof. Let $\mathcal{A}$ be an efficient adversary for the adaptive broadcast security game. We begin by defining a sequence of hybrid experiments:

- Hyb_0: This is the adaptive experiment $\text{EXP}^{(0)}_{\text{DBE}}$ from Definition A.1:
  - **Setup:** On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the number of slots 1^N. The challenger responds by computing $\text{pp} \leftarrow \text{Setup}(1^\lambda, 1^N)$ and gives $\text{pp} = (\text{SS.pp}, \text{pPRG.pp})$ to $\mathcal{A}$. Specifically, the challenger samples $(\text{pPRG.pp}, \sigma) \leftarrow \text{pPRG.Setup}(1^\lambda, 1^N)$, $\text{SS.pp} \leftarrow \text{SS.Setup}(1^\lambda, 1^{2N})$. The challenger also initializes sets Q and C to keep track of key-generation queries and corruption queries, respectively.

- **Query phase:** The challenger handles queries as follows:
  - **Key-generation queries:** On input index $i \in [N]$, the challenger sends $\perp$ to $\mathcal{A}$ if $i \in Q$. Otherwise, the challenger samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$, sends pk_i to $\mathcal{A}$, and adds i to Q. In particular, $\mathsf{pk}_i = (\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.pk}_{(i,1)}), \mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{(i,s_i)})$ where $s_i \xleftarrow{\mathrm{R}} \{0,1\}$ and
$$(\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.sk}_{(i,0)}) \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.pp}, (i,0))$$
$$(\mathsf{SS.pk}_{(i,1)}, \mathsf{SS.sk}_{(i,1)}) \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.pp}, (i,1)).$$
  - **Key-corruption queries:** On input index $i \in [N]$, the challenger sends $\perp$ to $\mathcal{A}$ if $i \notin Q \setminus C$. Otherwise the challenger sends sk_i to $\mathcal{A}$ and add i to C.
- **Challenge query:** After $\mathcal{A}$ outputs $S \subseteq Q \setminus C$, the challenger responds with $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \{i, \mathsf{pk}_i\}_{i \in S}, 0)$. Specifically, the challenger first computes $t_i = \mathsf{pPRG.Eval}(\mathsf{pPRG.pp}, \sigma_S, S, i)$ for all $i \in S$ and set ciphertext $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$ where
$$\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, \{(i, t_i)\mathsf{SS.pk}_{(i,t_i)}\}_{i \in S}, 0)$$
$$\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, \{(i, 1-t_i), \mathsf{SS.pk}_{(i,1-t_i)}\}_{i \in S}, 0).$$
- **Output:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.

- Hyb_1; Same as Hyb_0 except the challenger samples $\mathbf{s} = (s_1, \ldots, s_N) \xleftarrow{\mathrm{R}} \{0,1\}^N$ in the setup phase. Furthermore, it delays the computation of sk_i to the key-corruption queries. In particular, when responding to key-generation queries, the challenger samples the key pairs $(\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.sk}_{(i,0)}), (\mathsf{SS.pk}_{(i,1)}, \mathsf{SS.sk}_{(i,1)})$ exactly as in Hyb_0 and replies with $\mathsf{pk}_i = (\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.pk}_{(i,1)})$. It also stores $(\mathsf{SS.sk}_{(i,0)}, \mathsf{SS.sk}_{(i,0)})$. If $\mathcal{A}$ later makes a key-corruption query on index i, then the challenger samples $s_i \xleftarrow{\mathrm{R}} \{0,1\}$ and replies with $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{(i,s_i)})$ Note that in this hybrid, the challenger's behavior in a key-generation query does not depend on the bit s_i.

- Hyb_2: Same as Hyb_1 except when constructing the challenge ciphertext, the challenger now computes $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pPRG.pp}, \sigma, S)$.

- Hyb_3: Same as Hyb_2, except during setup, the challenger computes $\sigma_{[N]} \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, [N])$, and on each key-corruption query, the challenger sets $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_{[N]}, [N], i)$.

- Hyb_4: Same as Hyb_3 except the challenger switches $\mathsf{SS.ct}_1$ to be an encryption of 1. Namely, the challenger now computes $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_1, 1)$.

- Hyb_5: Same as Hyb_4, except on every key-corruption query, the challenger samples $s_i \xleftarrow{\mathrm{R}} \{0,1\}$.

- Hyb_6: Same as Hyb_5, except during setup, the challenger first computes $\sigma_{[N]} \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, [N])$, and on every key-corruption query, the challenger sets $s_i = 1 - \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_{[N]}, [N], i)$.

- Hyb_7: Same as Hyb_6, except the challenger switches $\mathsf{SS.ct}_0$ to be an encryption of 1. Namely, the challenger now computes $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, S_0, 1)$.

- Hyb_8: Same as Hyb_7 except the challenger now samples $\mathbf{s} \xleftarrow{\mathrm{R}} \{0,1\}^N$ during the setup phase.

- Hyb_9: Same as Hyb_8 except when constructing the challenge ciphertext, the challenger samples $\sigma_S \leftarrow \mathsf{pPRG.Samp}(\mathsf{pp}, S)$.

- Hyb_{10}: Same as Hyb_9 except the challenger now samples s_i in key-generation queries. In particular, when responding to a key-generation query, the challenger samples $(\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.sk}_{(i,0)}), (\mathsf{SS.pk}_{(i,1)}, \mathsf{SS.sk}_{(i,1)})$ as in Hyb_9. Then, it samples $s_i \xleftarrow{\mathrm{R}} \{0,1\}$ and sets $\mathsf{pk}_i = (\mathsf{SS.pk}_{(i,0)}, \mathsf{SS.pk}_{(i,1)})$, $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{(i,s_i)})$. This is experiment $\mathsf{EXP}^{(1)}_{\mathsf{DBE}}$ from Definition A.1.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid Hyb_i with adversary $\mathcal{A}$ (and an implicit security parameter λ). We now show that the output distributions of each adjacent pair of hybrid experiments is computationally indistinguishable.

Lemma A.6. *For all* $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

Proof. Note that in Hyb_0, the bit s_i is never revealed to the adversary $\mathcal{A}$ unless it makes a key corruption query on index i. Therefore the view of $\mathcal{A}$ is identical in both hybrids and the lemma follows. □

Lemma A.7. *Suppose* Π_{pPRG} *satisfies sampling indistinguishability. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| \ge \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the sampling indistinguishability game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{A}$ outputs a length parameter 1^N which algorithm $\mathcal{B}$ forwards to its challenger. The sampling indistinguishability challenger replies with pPRG.pp.
2. Algorithm $\mathcal{B}$ samples $\mathbf{s} \xleftarrow{\text{R}} \{0,1\}^N$, $\mathsf{SS.pp} \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$ and gives $\mathsf{pp} = (\mathsf{SS.pp}, \mathsf{pPRG.pp})$ to $\mathcal{A}$.
3. Algorithm $\mathcal{B}$ implements all of the key-generation and key-corruption queries from $\mathcal{A}$ using the challenger specification of Hyb_1.
4. When algorithm $\mathcal{A}$ makes a challenge query for the set $S \subseteq Q \setminus C$, algorithm $\mathcal{B}$ forwards S to its challenger and receives σ_S.
5. For each $i \in S$, algorithm $\mathcal{B}$ computes $t_i = \mathsf{pPRG.Eval}(\mathsf{pPRG.pp}, \sigma_S, S, i)$. It then sets
$$\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, \{(i, t_i), \mathsf{SS.pk}_{(i,t_i)}\}_{i \in S}, 0)$$
$$\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, \{(i, 1-t_i), \mathsf{SS.pk}_{(i,1-t_i)}\}_{i \in S}, 0).$$
The challenger responds with $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$.
6. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$ which algorithm $\mathcal{B}$ also outputs.

We now analyze the distribution of $\mathsf{EXP}^{(0)}_{\mathsf{samp}}(1^\lambda, \mathcal{B})$ for $\beta \in \{0,1\}$:

- Suppose $\beta = 0$. This means $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and $\sigma_S \leftarrow \mathsf{pPRG.Samp}(\mathsf{pp}, S)$. This is the distribution in Hyb_1, so algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.
- Suppose $\beta = 1$. This means $(\mathsf{pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$. This is the distribution in Hyb_2, so algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ breaks mode indistinguishability with non-negligible advantage ε. □

Lemma A.8. *Suppose* Π_{pPRG} *satisfies adaptive pseudorandomness. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \ge \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the adaptive pseudorandomness game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{A}$ outputs a length parameter 1^N which algorithm $\mathcal{B}$ forwards to its challenger. The sampling indistinguishability challenger replies with pPRG.pp.
2. Algorithm $\mathcal{B}$ samples $\mathsf{SS.pp} \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$ and gives $\mathsf{pp} = (\mathsf{SS.mpk}, \mathsf{pPRG.pp})$ to $\mathcal{A}$.

3. When algorithm $\mathcal{A}$ makes a key-corruption query on an index $i \in Q \setminus C$, algorithm $\mathcal{B}$ makes an evaluation query to its challenger on i and receives s_i. It then returns $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{(i,s_i)})$. Algorithm $\mathcal{B}$ implements all key-generation queries using the challenger specification of Hyb_2.

4. When algorithm $\mathcal{A}$ makes a challenge query for the set $S \subseteq Q \setminus \mathcal{S}$, algorithm $\mathcal{B}$ makes an evaluation query on all indices $[N] \setminus S$. It then makes a challenge query and receives σ_S.

5. For each $i \in S$, algorithm $\mathcal{B}$ computes $t_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_S, S, i)$. It then sets

$$\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, \{(i, t_i), \mathsf{SS.pk}_{(i,t_i)}\}_{i \in S}, 0)$$
$$\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.mpk}, \{(i, 1 - t_i), \mathsf{SS.pk}_{(i,1-t_i)}\}_{i \in S}, 0).$$

The challenger responds with $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$.

6. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$ which algorithm $\mathcal{B}$ also outputs.

We now analyze the distribution $\mathsf{EXP}_{\mathsf{prg}}^{(\beta)}(1^\lambda, \mathcal{B})$. First, let $I \subseteq [N]$ be the indices algorithm $\mathcal{B}$ makes to its evaluation oracle. First, from the requirements of the distributed broadcast security definition, we have that $S \subseteq Q \setminus C$. This means $C \subseteq Q \setminus S \subseteq [N] \setminus S$. Then by construction of $\mathcal{B}$, we have that $I = C \cup ([N] \setminus S) = [N] \setminus S$. Correspondingly, $[N] \setminus I = S$. We now consider the distribution of of $\mathsf{EXP}_{\mathsf{prg}}^{(\beta)}(1^\lambda, \mathcal{B})$ for each $\beta \in \{0,1\}$:

- Suppose $\beta = 0$. In this case, the challenger samples $(\mathsf{pPRG.pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and computes $\sigma_{[N]} \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, [N])$. It responds to each evaluation query on $i \in [N]$ with $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma, [N], i)$. Since $[N] \setminus I_{\mathcal{B}} = S$, the challenger responds with $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$ in the challenge phase. This is precisely the behavior in $\mathsf{Hyb}_3(\mathcal{A})$, so in this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]$.
- Suppose $\beta = 1$. In this case, the challenger samples $(\mathsf{pPRG.pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and $\mathbf{u} \xleftarrow{\mathrm{R}} \{0,1\}^N$. It responds to each evaluation query on $i \in [N]$ with $s_i = u_i$. In the challenge phase, the challenger again responds with $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$. This is precisely the behavior in $\mathsf{Hyb}_2(\mathcal{A})$, so in this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]$.

We conclude that algorithm $\mathcal{B}$ breaks adaptive pseudorandomness with non-negligible advantage ε. $\square$

Lemma A.9. *Suppose* Π_{SS} *satisfies semi-static security and* Π_{pPRG} *is correct. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]| \geq \varepsilon(\lambda)$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the semi-static security game:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{A}$ outputs a length parameter 1^N. Algorithm $\mathcal{B}$ then initializes two (empty) sets Q and C.

2. Algorithm $\mathcal{B}$ samples $(\mathsf{pPRG.pp}, \sigma) \leftarrow \mathsf{pPRG.Setup}(1^\lambda, 1^N)$ and $\sigma_{[N]} \leftarrow \mathsf{pPRG.Project}(\mathsf{pPRG.pp}, \sigma, [N])$. Then, for each $i \in [N]$, it computes $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_{[N]}, [N], i)$.

3. Algorithm $\mathcal{B}$ constructs the set $S^* = \{(i, 1 - s_i)\}_{i \in [N]}$. It forwards 1^{2N} together with the set S^* to the semi-static security challenger. The challenger replies with $\mathsf{SS.pp}$ and a collection of public keys $\{\mathsf{SS.pk}_{(i,1-s_i)}\}_{i \in [N]}$. Algorithm $\mathcal{B}$ gives $\mathsf{pp} = (\mathsf{SS.pp}, \mathsf{pPRG.pp})$ to $\mathcal{A}$.

4. When algorithm $\mathcal{A}$ makes a key-generation query on an index $i \in [N]$, algorithm $\mathcal{B}$ replies with $\perp$ if $i \in Q$. Otherwise, it samples
$$(\mathsf{SS.pk}_{i,s_i}, \mathsf{SS.sk}_{i,s_i}) \leftarrow \mathsf{KeyGen}(\mathsf{SS.pp}, (i, s_i))$$
and replies with $\mathsf{pk}_i = (\mathsf{SS.pk}_{i,0}, \mathsf{SS.pk}_{i,1})$. Algorithm $\mathcal{B}$ then adds i to Q.

5. When algorithm $\mathcal{A}$ makes a key-corruption query on an index $i \in [N]$, algorithm $\mathcal{B}$ replies with $\perp$ if $i \notin Q \setminus C$. Otherwise, it replies with $\mathsf{sk}_i = (i, s_i, \mathsf{SS.sk}_{i,s_i})$ and adds i to C.

6. When algorithm $\mathcal{A}$ makes a challenge query for a set $S \subseteq [N]$, algorithm $\mathcal{B}$ starts by computing a projected seed $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$. Then it sets $S_0 = \{(i, s_i)\}_{i \in S}$ and $S_1 = \{(i, 1 - s_i)\}_{i \in S}$. It computes $\mathsf{SS.ct}_0 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.pp}, (i, s_i), \mathsf{SS.pk}_{i,s_i\,(i,s_i) \in S_0}, 0)$ and forwards S_1 to the semi-static security challenger. The challenger replies with a ciphertext $\mathsf{SS.ct}_1$. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with $\mathsf{ct} = (\mathsf{SS.ct}_0, \mathsf{SS.ct}_1, \sigma_S)$.
7. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$ which algorithm $\mathcal{B}$ also outputs.

By construction, algorithm $\mathcal{B}$ is a *valid* adversary for the semi-static security game. Specifically, the challenge set S_1 always satisfies $S_1 \subseteq S^*$. We now analyze the distributions of $\mathsf{EXP}^{(\beta)}_{\mathsf{SSBE}}(1^\lambda, \mathcal{B})$. We consider each component separately.

- The semi-static security challenger samples $\mathsf{SS.pp} \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^{2N})$, which coincides with the distribution of $\mathsf{SS.pp}$ in Hyb_3 and Hyb_4. In addition, the challenger samples the public keys as $(\mathsf{SS.pk}_{i,1-s_i}, \mathsf{SS.sk}_{i,1-s_i}) \leftarrow \mathsf{SS.KeyGen}(\mathsf{SS.pp}, (i, s_i))$. This also coincides with the distribution in Hyb_3 and Hyb_4.
- Next, consider the sets S_0 and S_1. In the reduction, algorithm $\mathcal{B}$ uses public key sets $\{\mathsf{pk}_{(i,s_i)}\}_{i \in S}$ for $\mathsf{SS.ct}_0$ and $\{\mathsf{pk}_{(i,1-s_i)}\}_{i \in S}$ for $\mathsf{SS.ct}_1$, where $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_{[N]}, [N], i)$. Since $\sigma_S \leftarrow \mathsf{pPRG.Project}(\mathsf{pp}, \sigma, S)$, correctness of Π_{pPRG} means that $s_i = \mathsf{pPRG.Eval}(\mathsf{pp}, \sigma_S, S, i)$ for all $i \in S$. Thus, the public key sets are constructed exactly as in Hyb_3 and Hyb_4. Thus, $\mathsf{SS.ct}_0$ is distributed exactly according to the distribution in Hyb_3 and Hyb_4.
- It suffices to consider the distribution of $\mathsf{SS.ct}_1$. When $\beta = 0$, $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.pp}, \{\mathsf{pk}_{(i,1-s_i)}\}_{i \in S}, 0)$ and when $\beta = 1$, $\mathsf{SS.ct}_1 \leftarrow \mathsf{SS.Enc}(\mathsf{SS.pp}, \{\mathsf{pk}_{(i,1-s_i)}\}_{i \in S}, 1)$. The former corresponds to the distribution in Hyb_3 while the latter corresponds to the distribution in Hyb_4.

We conclude that algorithm $\mathcal{B}$ breaks semi-static security with non-negligible advantage ε. □

Lemma A.10. *Suppose Π_{pPRG} satisfies adaptive pseudorandomness. 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. Follows by an analogous argument as the proof of Lemma A.8. □

Lemma A.11. *Suppose Π_{pPRG} satisfies adaptive pseudorandomness. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Follows by an analogous argument as the proof of Lemma A.8. □

Lemma A.12. *Suppose Π_{SS} satisfies semi-static security and Π_{pPRG} is correct. Then, 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. Follows by an analogous argument as the proof of Lemma A.9. □

Lemma A.13. *Suppose Π_{pPRG} satisfies adaptive pseudorandomness. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Follows by an analogous argument as the proof of Lemma A.8. □

Lemma A.14. *Suppose Π_{pPRG} satisfies sampling indistinguishability. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Follows by an analogous argument as the proof of Lemma A.7. □

Lemma A.15. *For all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{10}(\mathcal{A}) = 1]| = 0$.*

Proof. Follows by an analogous argument as the proof of Lemma A.6. □

Adaptive security now follows by combining Lemmas A.6 to A.15. □

Theorem A.16 (Succinctness). *Suppose Π_{SS} and Π_{pPRG} are succinct. Then Construction A.2 is succinct.*

Proof. In Construction A.2, a ciphertext for a set $S \subseteq [N]$ consists of two ciphertexts for the semi-static distributed broadcast encryption scheme Π_{SS} as well as a projected seed σ_S for Π_{pPRG}. By succinctness of the underlying primitives, the size of the ciphertext is $2 \cdot o(|S|) \cdot \mathsf{poly}(\lambda, \log N) + \mathsf{poly}(\lambda, \log N)$, which satisfies the required succinctness properties. □

B Publicly-Sampleable Projective PRGs from RSA

In this section, we recall the projective PRG scheme from [ABI+23, §3.2] based on the RSA assumption [RSA78]. In the following, we write $\mathsf{Primes}_\lambda \subset \mathbb{N}$ to denote the set of λ-bit primes. Similar to [ABI+23], we state the RSA assumption with respect to *prime* exponents (as opposed to random exponents over $\mathbb{Z}^*_{\varphi(N)}$). Since the density of primes is $\Theta(1/\log N)$, the standard RSA assumption implies the variant with prime exponents. We define this as follows:

Definition B.1 (Composite Modulus Sampler)**.** Let λ be a security parameter. A composite-modulus sampler is an efficient algorithm $\mathsf{SampleN}$ that takes as input the security parameter 1^λ and outputs (N, p, q) where $N = pq$ and $p, q \in \mathsf{Primes}_\lambda$ are distinct λ-bit primes.

Assumption B.2 (RSA with Prime Exponents)**.** Let $\mathsf{SampleN}$ be a composite-modulus sampler. We say the RSA assumption with prime exponents holds with respect to $\mathsf{SampleN}$ if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathcal{A}(1^\lambda, N, e, u^e \bmod N) = u : \begin{array}{c} (N, p, q) \leftarrow \mathsf{SampleN}(1^\lambda) \\ e \xleftarrow{\mathrm{R}} \mathbb{Z}^*_{\varphi(N)} \cap \mathsf{Primes}_\lambda \,,\, u \xleftarrow{\mathrm{R}} \mathbb{Z}^*_N \end{array}\right] = \mathsf{negl}(\lambda).$$

Publicly-sampleable projective PRG from RSA. We now show how to construct a publicly-sampleable projective PRGs from the RSA assumption. The construction is nearly identical with the one from [ABI+23, §3.2], except we introduce an explicit public sampling algorithm. We then prove that the construction satisfies the security properties from Definition 4.1. As discussed in Definition 4.1, our security properties do not follow as a consequence of the security properties considered in [ABI+23] which is why we include the full analysis here.

Construction B.3 (Projective PRG from RSA)**.** Let $\mathsf{SampleN}$ be a composite-modulus sampler. Let $\mathsf{hc}\colon \{0,1\}^{2\lambda} \times \{0,1\}^{2\lambda} \to \{0,1\}$ be the Goldreich-Levin extractor. We construct a publicly-sampleable projective PRG $\Pi_{\mathsf{pPRG}} = (\mathsf{Setup}, \mathsf{Samp}, \mathsf{Project}, \mathsf{Eval})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp}, \sigma)$: On input the security parameter λ and the output length $\ell \in \mathbb{N}$, the setup algorithm starts by sampling $(N, p, q) \leftarrow \mathsf{SampleN}(1^\lambda)$. It then samples $x \xleftarrow{\mathrm{R}} \mathbb{Z}^*_N$, extractor randomness $\mathbf{r} \xleftarrow{\mathrm{R}} \{0,1\}^{2\lambda}$, and primes $e_1, \ldots, e_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}^*_{\varphi(N)} \cap \mathsf{Primes}_\lambda$. It outputs the public parameters $\mathsf{pp} = (N, e_1, \ldots, e_\ell, \mathbf{r})$ and the seed $\sigma = (x, \varphi(N))$.
- $\mathsf{Samp}(\mathsf{pp}, S) \to \sigma_S$: On input the parameter $\mathsf{pp} = (N, e_1, \ldots, e_\ell, \mathbf{r})$ and a target set $S \subseteq [\ell]$, the sampling algorithm samples a random element $y \xleftarrow{\mathrm{R}} \mathbb{Z}^*_N$ and output the simulated projected seed $\sigma_S = y$.
- $\mathsf{Project}(\mathsf{pp}, \sigma, S) \to \sigma_S$: On input the public parameters $\mathsf{pp} = (N, e_1, \ldots, e_\ell, \mathbf{r})$, the key $\sigma = (x, \varphi(N))$, and a target set $S \subseteq [\ell]$, the projection algorithm computes $x_S = x^{\prod_{j \in [\ell] \setminus S} e_j} \bmod N$. It outputs the projected seed $\sigma_S = x_S$.
- $\mathsf{Eval}(\mathsf{pp}, \sigma, S, i)$: On input the public parameters $\mathsf{pp} = (N, e_1, \ldots, e_\ell, \mathbf{r})$, a projected seed $\sigma = x_S$, the associated set of indices $S \subseteq [\ell]$, and an index $i \in S$, the evaluation algorithm computes $x_i = x^{\prod_{j \in S \setminus \{i\}} e_j} \bmod N$ and outputs $\mathsf{hc}(x_i, \mathbf{r})$.

Theorem B.4 (Correctness)**.** *Construction B.3 is correct.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, output length $\ell \in \mathbb{N}$, set of indices $S \subseteq [\ell]$, and index $i \in S$. Let $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$ and $\sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, S)$. By construction, this means $\mathsf{pp} = (N, e_1, \ldots, e_\ell, \mathbf{r})$ and $\sigma = x \in \mathbb{Z}^*_N$. Now, observe that $\sigma_S = x_S = x^{\prod_{j \in [\ell] \setminus S} e_j} \bmod N$, therefore in $\mathsf{Eval}(\mathsf{pp}, \sigma_S, S, i)$,

$$x_i = x_S^{\prod_{j \in S \setminus \{i\}} e_j} = x^{\prod_{j \in [\ell] \setminus \{i\}} e_j},$$

which is a fixed value independent of the choice of S. Therefore, $\mathsf{Eval}(\mathsf{pp}, \sigma_S, S, i) = \mathsf{Eval}(\mathsf{pp}, \sigma_{[\ell]}, [\ell], i)$. □

Theorem B.5 (Succinctness). *Construction B.3 is succinct.*

Proof. A projected seed is a single element of $\mathbb{Z}_N$, which can be described by at most 2λ bits. □

Theorem B.6 (Sampling Indistinguishability). *Construction B.3 satisfies perfect sampling indistinguishability.*

Proof. Since $e_i \in \mathbb{Z}^*_{\varphi(N)}$, for all sets $S \subseteq [\ell]$, the mapping $x \mapsto x_S = x^{\prod_{j\in[\ell]\setminus S} e_j}$ is a permutation over $\mathbb{Z}_N^*$. Therefore, for all S, the distribution of $\sigma_S = x_S$ for random sampled $x \xleftarrow{R} \mathbb{Z}_N^*$ is identical to a random $y \xleftarrow{R} \mathbb{Z}_N^*$. Hence $\mathsf{EXP}^{(0)}_{\mathsf{samp}} \equiv \mathsf{EXP}^{(1)}_{\mathsf{samp}}$ and that Construction B.3 satisfies perfect sampling indistinguishability. □

Theorem B.7 (Adaptive Pseudorandomness). *If the RSA assumption with prime exponents hold, then Construction B.3 satisfies adaptive pseudorandomness.*

Proof. Before proving the theorem, we first state the following corollary of the RSA assumption (Assumption B.2) and Theorem 3.2. This will be useful in our security analysis.

Lemma B.8. *Let* $\mathsf{hc}\colon \{0,1\}^{2\lambda} \times \{0,1\}^{2\lambda} \to \{0,1\}$ *be the Goldreich-Levin extractor. For a security parameter* λ *and an adversary* $\mathcal{A}$*, define the following distinguishing game* $\mathsf{EXP}^{(\beta)}(1^\lambda, \mathcal{A})$*:*

- *On input the security parameter* 1^λ*, the adversary outputs a length parameter* 1^ℓ *and an index* $i \in [\ell]$*.*
- *The challenger samples* $(N,p,q) \leftarrow \mathsf{SampleN}(1^\lambda)$*,* $u \xleftarrow{R} \mathbb{Z}_N^*$*, extractor randomness* $\mathbf{r} \xleftarrow{R} \{0,1\}^{2\lambda}$*, and random primes* $e_1,\dots,e_\ell \xleftarrow{R} \mathsf{Primes}_\lambda$*. Additionally, the challenger computes bits* $b_0 = \mathsf{hc}(u^{\prod_{j\neq i} e_j} \bmod N, \mathbf{r})$ *and* $b_1 \xleftarrow{R} \{0,1\}$*. The challenger gives* $(N, u^{e_i}, e_1, \dots, e_\ell, \mathbf{r}, b_\beta)$ *to the adversary.*
- *The adversary outputs a bit* $b' \in \{0,1\}$*, which is the output of the experiment.*

Suppose the RSA assumption holds. Then, for all efficient adversaries $\mathcal{A}$*, there exist a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr[\mathsf{EXP}^{(0)}(1^\lambda,\mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}(1^\lambda,\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose there exists some efficient adversary $\mathcal{A}$ where

$$|\Pr[\mathsf{EXP}^{(0)}(1^\lambda,\mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}(1^\lambda,\mathcal{A}) = 1]| = \varepsilon(\lambda)$$

for some non-negligible ε. For simplicity, we split $\mathcal{A}$ into two algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, where $\mathcal{A}_0$ takes as input 1^λ and outputs a pair $(1^\ell, i)$ along with a private state st, and $\mathcal{A}_1$ takes as input the private state st along with the challenger's response and outputs the bit b'. We now define a joint distribution $(X,Y) = \{(X_\lambda, Y_\lambda)\}_{\lambda\in\mathbb{N}}$ as follows:

- Run $\mathcal{A}_0(1^\lambda)$ to receive a pair $(1^\ell, i)$ and private state st. Sample $(N,p,q) \leftarrow \mathsf{SampleN}(1^\lambda)$, $u \xleftarrow{R} \mathbb{Z}_N^*$, $\mathbf{r} \xleftarrow{R} \{0,1\}^{2\lambda}$, and $e_1,\dots,e_\ell \xleftarrow{R} \mathsf{Primes}_\lambda$.
- Let $x = u^{\prod_{j\neq i} e_j}$ and $y = (\mathsf{st}, i, N, u^{e_i}, e_1, \dots, e_\ell)$. Output the pair (x,y).

We now show that under the RSA assumption, X is computationally unpredictable given Y. Suppose there exist an efficient algorithm $\mathcal{A}'$ that can predict x given y with some non-negligible probability ε'. We construct an efficient algorithm $\mathcal{B}$ for the RSA problem:

- On input $(1^\lambda, N, e, v)$, where $(N,p,q) \leftarrow \mathsf{SampleN}(1^\lambda)$, $e \xleftarrow{R} \mathbb{Z}^*_{\varphi(N)} \cap \mathsf{Primes}_\lambda$, $v = u^e \bmod N$, and $u \xleftarrow{R} \mathbb{Z}_N^*$, algorithm $\mathcal{B}$ starts running $\mathcal{A}_0(1^\lambda)$ to receive a pair $(1^\ell, i)$ and state st. For each $i \in [\ell] \setminus \{i\}$, it samples $e_i \xleftarrow{R} \mathsf{Primes}_\lambda$. It sets $e_i = e$. Finally, it sets $y = (\mathsf{st}, i, N, v, e_1, \dots, e_\ell)$
- Algorithm $\mathcal{B}$ runs $\mathcal{A}'(1^\lambda, y)$ to obtain x. It then computes the Bezout coefficients $\alpha, \beta \in \mathbb{Z}$ where $\alpha \prod_{j\neq i} e_j + \beta e = 1$, which exists and are efficiently computable as long as $\gcd(e_j, e) = 1$ for all $j \neq i$. The algorithm $\mathcal{B}$ aborts if the coefficients do not exist.
- Algorithm $\mathcal{B}$ outputs $x^\alpha \cdot v^\beta$.

The advantage of $\mathcal{B}$ can be analyzed as follows:

- With overwhelming probability, a random prime $e_i \xleftarrow{R} \mathsf{Primes}_\lambda$ will satisfy $\gcd(e_i, \varphi(N)) = 1$, so the uniform distribution over Primes_λ and $\mathbb{Z}^*_{\varphi(N)} \cap \mathsf{Primes}_\lambda$ is statistically indistinguishable. In this case, algorithm $\mathcal{B}$ correctly simulates the distribution of y in $(x, y) \leftarrow (X_\lambda, Y_\lambda)$.
- By assumption, with probability $\varepsilon' - \mathsf{negl}(\lambda)$, algorithm $\mathcal{A}'$ outputs $x = u^{\prod_{j \neq i} e_j}$.
- With overwhelming probability over the choice of $e_j \xleftarrow{R} \mathsf{Primes}_\lambda$, it holds that $e_j \neq e$. Since $e_1, \ldots, e_\ell, e$ are all prime, this means $\gcd(e_j, e) = 1$ for all $j \neq i$, and correspondingly, that $\gcd(\prod_{j \neq i} e_j, e) = 1$. Thus, with overwhelming probability over the choice of $e_1, \ldots, e_\ell, e$, algorithm $\mathcal{B}$ successfully computes the Bezout coefficients α, β.
- If $x = u^{\prod_{j \neq i} e_j}$ and the Bezout coefficients exist, algorithm $\mathcal{B}$ outputs $x^\alpha \cdot v^\beta = u^{\alpha \cdot \prod_{j \neq i} e_j} \cdot u^{\beta e} = u$, which is the solution to the RSA challenge.

Therefore the advantage of $\mathcal{B}$ is $\varepsilon' - \mathsf{negl}(\lambda)$, which is non-negligible. Therefore, under the RSA assumption with prime exponents, we conclude that X is computationally unpredictable given Y. The lemma now follows from Theorem 3.2:

- Since X is computationally unpredictable given Y, Theorem 3.2 now states that the distributions $(1^\lambda, \mathbf{r}, \mathsf{hc}(x, \mathbf{r}), y)$ and $(1^\lambda, \mathbf{r}, b, y)$ are computationally indistinguishable when $\mathbf{r} \xleftarrow{R} \{0,1\}^{2\lambda}$, $(x, y) \leftarrow (X_\lambda, Y_\lambda)$ and $b \xleftarrow{R} \{0,1\}$.
- If algorithm $\mathcal{A}$ distinguishes $\mathsf{EXP}^{(0)}$ and $\mathsf{EXP}^{(1)}$ with advantage ε, then we can use $\mathcal{A}_1$ to construct a distinguisher for the distributions $(1^\lambda, \mathbf{r}, \mathsf{hc}(x, \mathbf{r}), y)$ and $(1^\lambda, \mathbf{r}, b, y)$ as follows:
  - On input $(1^\lambda, \mathbf{r}, b, y)$ where $y = (\mathsf{st}, i, N, v, e_1, \ldots, e_\ell)$, compute and output $\mathcal{A}_1(\mathsf{st}, N, v, e_1, \ldots, e_\ell, \mathbf{r}, b)$.

  When $b = \mathsf{hc}(x, \mathbf{r})$, this process perfectly simulates an execution of $\mathsf{EXP}^{(0)}$ for $\mathcal{A}_1$ and when $b \xleftarrow{R} \{0,1\}$, this perfectly simulates an execution of $\mathsf{EXP}^{(1)}$ for $\mathcal{A}_1$. Thus, this algorithm distinguishes between $(1^\lambda, \mathbf{r}, \mathsf{hc}(x, \mathbf{r}), y)$ and $(1^\lambda, \mathbf{r}, b, y)$ with non-negligible advantage ε. □

Proof of Theorem B.7. We now return to the proof of Theorem B.7. The proof follows a similar strategy as the proof of Theorem 6.7. For each index $i \in \{0\} \cup \mathbb{N}$, we define an experiment Hyb_i as follows:

- Hyb_i: This is a variant of the adaptive pseudorandomness experiment:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the length parameter 1^ℓ. The challenger runs $(\mathsf{pp}, \sigma) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$. Specifically, the challenger samples $(N, p, q) \leftarrow \mathsf{SampleN}(1^\lambda)$, $x \xleftarrow{R} \mathbb{Z}_N^*$, $\mathbf{r} \xleftarrow{R} \{0,1\}^{2\lambda}$, and $e_1, \ldots, e_\ell \xleftarrow{R} \mathbb{Z}^*_{\varphi(N)} \cap \mathsf{Primes}_\lambda$. It sets $\mathsf{pp} = (N, e_1, \ldots, e_\ell, \mathbf{r})$ and $\sigma = x$. The challenger gives pp to $\mathcal{A}$.
  - The challenger samples $\sigma_{[\ell]} \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell])$ and $\mathbf{t} \xleftarrow{R} \{0,1\}^\ell$. In particular $\sigma_{[\ell]} = x$.
  - When $\mathcal{A}$ makes an evaluation query on $j \in [\ell]$, the challenger replies with $\mathsf{Eval}(\mathsf{pp}, \sigma_{[\ell]}, [\ell], j)$ if $j > i$ and with t_j if $j \leq i$. In particular, when $j > i$, the challenger computes $x_j = x^{\prod_{k \in [\ell] \setminus \{j\}} e_k} \bmod N$ and responds with $\mathsf{hc}(x_j, \mathbf{r})$.
  - After $\mathcal{A}$ finishes making evaluation queries, the challenger computes the seed $\sigma_S \leftarrow \mathsf{Project}(\mathsf{pp}, \sigma, [\ell] \setminus I)$ where $I \subseteq [\ell]$ is the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. In particular, $\sigma_S = x^{\prod_{j \in [\ell] \setminus S} e_j} \bmod N$. The challenger gives σ_S to $\mathcal{A}$.
  - At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b \in \{0,1\}$, which is the output of the experiment.

We write $\mathsf{Hyb}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid Hyb_i with adversary $\mathcal{A}$ (and an implicit security parameter λ). By construction, observe that $\mathsf{EXP}^{(0)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{EXP}^{(1)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_\ell(\mathcal{A})$. Following the same argument in the proof of Theorem 6.7, for all indices $i \in \{0\} \cup \mathbb{N}$,

$$\Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i] = \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1], \tag{B.1}$$

where E_i is the event that algorithm $\mathcal{A}$ makes an evaluation query on index i. Suppose now that

$$|\Pr[\mathsf{EXP}_{\mathsf{prg}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{prg}}^{(1)}(1^\lambda, \mathcal{A}) = 1]| \geq \varepsilon(\lambda) \tag{B.2}$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the distinguishing problem from Lemma B.8:

1. On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ to obtain the output length 1^ℓ. Algorithm $\mathcal{B}$ samples a random index $i \xleftarrow{R} [\ell]$ and sends $(1^\ell, i)$ to the challenger.
2. The challenger responds with $(N, v, e_1, \ldots, e_\ell, \mathbf{r}, b)$, where $v = u^{e_i}$ and $u \xleftarrow{R} \mathbb{Z}_N^*$. Algorithm $\mathcal{B}$ sets $\mathsf{pp} = (N, e_1, \ldots, e_\ell, \mathbf{r})$ and gives pp to $\mathcal{A}$. Algorithm $\mathcal{B}$ also samples $\mathbf{t} \xleftarrow{R} \{0,1\}^\ell$.
3. When algorithm $\mathcal{A}$ makes an evaluation query on an index $j \in [\ell]$, the challenger responds as follows:
   - If $j < i$, the challenger responds with t_j.
   - If $j = i$, the challenger responds with b.
   - If $j > i$, the challenger computes $x_j = v^{\prod_{k \in [\ell] \setminus \{i,j\}} e_k} \bmod N$ and responds with $\mathsf{hc}(x_j, \mathbf{r})$.
4. After $\mathcal{A}$ finishes making evaluation queries, let $I \subseteq [\ell]$ be the set of indices on which algorithm $\mathcal{A}$ made an evaluation query. If $i \notin I$, then algorithm $\mathcal{B}$ outputs 0. Otherwise algorithm $\mathcal{B}$ responds with $\sigma_S = v^{\prod_{j \in I \setminus \{i\}} e_j}$.
5. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$ which algorithm $\mathcal{B}$ also outputs.

Since $\mathcal{A}$ is efficient, algorithm $\mathcal{B}$ is efficient, so it suffices to analyze its advantage.

Analyzing the advantage of $\mathcal{B}$. Let W_0 be the event that $\mathcal{B}$ outputs 1 when the challenger sets $b = \mathsf{hc}(u^{\prod_{j \neq i} e_j}, \mathbf{r})$ and W_1 be the event that $\mathcal{B}$ outputs 1 when the challenger samples $b \xleftarrow{R} \{0,1\}$. Suppose algorithm $\mathcal{B}$ samples $i = i^*$ in the above reduction. By construction, the challenger for the experiment in Lemma B.8 samples $N, u, e_1, \ldots, e_\ell$ as in the normal setup algorithm, except it samples the exponents $e_i \xleftarrow{R} \mathsf{Primes}_\lambda$ instead of $e_i \xleftarrow{R} \mathbb{Z}_{\varphi(N)}^* \cap \mathsf{Primes}_\lambda$. However, with overwhelming probability over the choice of $e_i \xleftarrow{R} \mathsf{Primes}_\lambda$, it holds that $\gcd(e_i, \varphi(N)) = 1$, so these two distributions are statistically close. Correspondingly, the public parameters simulated by algorithm $\mathcal{B}$ are statistically close to the distribution of public parameters in $\mathsf{EXP}_{\mathsf{prg}}^{(0)}$ and $\mathsf{EXP}_{\mathsf{prg}}^{(1)}$. By construction, algorithm $\mathcal{B}$ also simulates the evaluation queries for $j > i$ and the projected seed exactly as in $\mathsf{EXP}_{\mathsf{prg}}^{(0)}$ and $\mathsf{EXP}_{\mathsf{prg}}^{(1)}$, where the challenger's secret u plays the role of x in the reduction. We consider the distribution of the challenge bit b.

- Suppose $b = \mathsf{hc}(u^{\prod_{j \neq i} e_j}, \mathbf{r})$. In this case, the responses to the evaluation queries are distributed according to the specification in Hyb_{i^*-1}. We consider the probability that algorithm $\mathcal{B}$ outputs 1 in this case. First, if algorithm $\mathcal{A}$ does not make an evaluation query on index i^*, then algorithm $\mathcal{B}$ always outputs 0. If algorithm $\mathcal{A}$ does make an evaluation query on index i^*, then algorithm $\mathcal{B}$ perfectly simulates the challenge according to the distribution in Hyb_{i^*-1}, and thus, outputs 1 with probability $\Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}]$. Thus, in this case, algorithm $\mathcal{B}$ outputs 1 with probability
$$\Pr[\mathcal{B} \text{ outputs } 1 \mid i = i^*] = \Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}] \cdot \Pr[\mathsf{E}_{i^*}] = \Pr[\mathsf{Hyb}_{i^*-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_{i^*}].$$
- Suppose $b \xleftarrow{R} \{0,1\}$. In this case, the responses to the evaluation queries are distributed according to the specification in Hyb_{i^*}. By a similar reasoning as in the previous case, we conclude that in this case
$$\Pr[\mathcal{B} \text{ outputs } 1 \mid i = i^*] = \Pr[\mathsf{Hyb}_{i^*}(\mathcal{A}) = 1 \mid \mathsf{E}_{i^*}] \cdot \Pr[\mathsf{E}_{i^*}] = \Pr[\mathsf{Hyb}_{i^*}(\mathcal{A}) = 1 \wedge \mathsf{E}_{i^*}].$$

Finally, algorithm $\mathcal{B}$ samples $i \xleftarrow{R} [\ell]$. Thus

$$\Pr[W_0 = 1] = \frac{1}{\ell} \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i],$$
$$\Pr[W_1 = 1] = \frac{1}{\ell} \sum_{i \in [\ell]} \Pr[\mathsf{Hyb}_{i}(\mathcal{A}) = 1 \wedge \mathsf{E}_i].$$

Since $\mathsf{EXP}^{(0)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{EXP}^{(1)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) \equiv \mathsf{Hyb}_\ell(\mathcal{A})$, we appeal to Eqs. (6.1) and (6.2) and conclude that

$$\begin{aligned}|\Pr[W_0 = 1] - \Pr[W_1 = 1]| &= \frac{1}{\ell}\left|\sum\nolimits_{i\in[\ell]} \Pr[\mathsf{Hyb}_{i-1}(\mathcal{A}) = 1 \wedge \mathsf{E}_i] - \Pr[\mathsf{Hyb}_i(\mathcal{A}) = 1 \wedge \mathsf{E}_i]\right| \\ &= \frac{1}{\ell}\left|\sum\nolimits_{i\in[\ell]} p_i - p_{i-1}\right| \\ &= \frac{1}{\ell}|p_0 - p_\ell| = \frac{1}{\ell}|\Pr[\mathsf{EXP}^{(0)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}_{\mathsf{prg}}(1^\lambda, \mathcal{A}) = 1]| \geq \frac{\varepsilon}{\ell},\end{aligned}$$

which is non-negligible since $\mathcal{A}$ is efficient which means $\ell = \mathsf{poly}(\lambda)$. □

Remark B.9 (Reducing the Public Parameter Size)**.** The public parameters in Construction B.3 scale linearly with the output length (since it includes the description of the exponents $e_1, \ldots, e_\ell$). The work of [ABI+23] describes an approach to replace the description of $(e_1, \ldots, e_\ell)$ with a succinct seed of size $\mathsf{poly}(\lambda, \log \ell)$. The idea is to generate the primes in a pseudorandom manner and replace the tuple $(e_1, \ldots, e_\ell)$ with a short seed ρ that can be expanded into a sequence of primes. Moreover, to facilitate the security proof, there is a procedure that allows "programming" the seed to output a specific prime at a particular index $i \in [\ell]$. We refer to [ABI+23, §3.2.1] for details on this approach.

C The Gentry-Waters Semi-Static Broadcast Encryption Scheme

Gentry and Waters [GW09] previously showed how to construct a semi-statically-secure broadcast encryption scheme with *linear-size* public parameters (and constant-size secret keys and ciphertexts) from the decisional q-bilinear Diffie-Hellman exponent sum assumption. Their work describes the construction as a special case of an adaptively-secure identity-based broadcast encryption scheme in the random oracle model. For completeness, we include a self-contained description of a simplified version of their construction specialized to the setting of vanilla broadcast encryption with *semi-static* security in the *plain* model. Then, using our publicly-sampleable projective PRGs (say from the computational bilinear Diffie-Hellman assumption), we obtain an adaptively-secure broadcast encryption scheme in the plain model with linear-size public keys (and constant-size secret keys and ciphertexts). We start by introducing the decisional q-BDHE sum problem that they rely on for security. For simplicity, we give a game-based formulation.

Assumption C.1 (Decision q-BDHE Sum)**.** Let $\mathsf{PrimeBGroupGen}$ be a prime-order bilinear group generator. For a security parameter λ, a bit $b \in \{0, 1\}$, and an adversary $\mathcal{A}$, we define the q-bilinear Diffie-Hellman exponent sum experiment $q\text{-}\mathsf{BDHES}^{(b)}(1^\lambda, \mathcal{A})$ as follow:

- On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a tuple (S, m), where $S \subset \mathbb{Z}, m \in \mathbb{Z}$.
- The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda)$ and a random exponent $a \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. The challenger computes $[Z_0]_{\mathbb{G}_T} = [a^m]_{\mathbb{G}_T}$ and $[Z_1]_{\mathbb{G}_T} \xleftarrow{\mathrm{R}} \mathbb{G}_T$, and gives $(\mathcal{G}, \{[a^i]_{\mathbb{G}}\}_{i\in S}, [Z_b]_{\mathbb{G}_T})$ to the adversary.
- The adversary outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

We say the adversary $\mathcal{A}$ is admissible if it outputs a tuple (S, m) where for every $s_0, s_1 \in S \cup \{0\}$ it holds that $s_0 + s_1 \neq m$. We say the decision q-BDHE sum assumption holds with respect to $\mathsf{PrimeBGroupGen}$ if for every efficient and admissible adversary $\mathcal{A}$, there exist a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left|\Pr[q\text{-}\mathsf{BDHES}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[q\text{-}\mathsf{BDHES}^{(1)}(1^\lambda, \mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Construction C.2 ([GW09, §4, adapted])**.** Let $\mathsf{PrimeBGroupGen}$ be a prime-order bilinear group generator. The broadcast encryption scheme $\Pi_{\mathsf{BE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is constructed as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the number of parties N, the setup algorithm samples a prime-order pairing group $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeBGroupGen}(1^\lambda)$ and random exponents $a, u, v, \beta, \gamma, r_1, \ldots, r_N \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. If $a \in [N]$, then the setup algorithm outputs $\mathsf{mpk} = \mathsf{msk} = \perp$. Otherwise, it outputs

$$\begin{aligned}\mathsf{mpk} &= (\{[ua^j]_{\mathbb{G}}\}_{j\in[0,N-2]}, \{[va^k]_{\mathbb{G}}, [\beta va^k]_{\mathbb{G}}\}_{k\in[0,N]}, [\gamma\beta v]_{\mathbb{G}}, [\gamma\beta va]_{\mathbb{G}}), \\ \mathsf{msk} &= (a, u, v, \beta, \gamma, r_1, \ldots, r_n).\end{aligned}$$

- $\mathsf{KeyGen}(\mathsf{msk}, i)$: On input the master secret key msk and an index $i \in [N]$, if $\mathsf{msk} = \bot$, then output $\bot$. Otherwise, parse $\mathsf{msk} = (a, u, v, \beta, \gamma, r_1, \dots, r_n)$ and compute $[s_i]_{\mathbb{G}} = [\beta u \cdot \frac{\gamma - r_i}{a-i}]_{\mathbb{G}}$. Output $\mathsf{sk}_i = (i, r_i, [s_i]_{\mathbb{G}})$. Note that by construction of Setup, if $\mathsf{msk} \neq \bot$, then $a \notin [N]$ so $(a-i)^{-1}$ is well-defined.
- $\mathsf{Enc}(\mathsf{mpk}, S, \mu)$: On input the master public key mpk a set $S \subseteq [N]$, and a message $\mu \in \{0,1\}$, the encryption algorithm simply outputs $\mathsf{ct} = \mu$ if $\mathsf{mpk} = \bot$. Otherwise, it parses

$$\mathsf{mpk} = (\{[ua^j]_{\mathbb{G}}\}_{j \in [0, N-2]}, \{[va^k]_{\mathbb{G}}, [\beta va^k]_{\mathbb{G}}\}_{k \in [0,N]}, [\gamma\beta v]_{\mathbb{G}}, [\gamma\beta va]_{\mathbb{G}}),$$

samples a random $t \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and computes the degree-N polynomial P defined by

$$P(x) = \left(\prod_{k \in S}(x-k)\right) \cdot \left(\prod_{k \in [N] \setminus S}(x-k-N)\right). \tag{C.1}$$

Let $p_0, \dots, p_N$ be the coefficients of $P(x)$. Namely, write $P(x) = \sum_{k=0}^{N} p_k x^k$. The encryption algorithm then computes

$$\begin{aligned}
[C_1]_{\mathbb{G}} &= \sum_{k=0}^{N}(p_k t \cdot [va^k]_{\mathbb{G}}) = [tvP(a)]_{\mathbb{G}}, \\
[C_2]_{\mathbb{G}} &= t \cdot [\gamma\beta v]_{\mathbb{G}} = [t\gamma\beta v]_{\mathbb{G}}, \\
[C_3]_{\mathbb{G}} &= t \cdot [\beta v]_{\mathbb{G}} = [t\beta v]_{\mathbb{G}}, \\
[C_4]_{\mathbb{G}_T} &= t \cdot [ua^{N-2}]_{\mathbb{G}} \cdot [\beta va]_{\mathbb{G}} = [t\beta uva^{N-1}]_{\mathbb{G}_T}, \\
[C_5]_{\mathbb{G}_T} &= t \cdot [ua^{N-2}]_{\mathbb{G}} \cdot [\gamma\beta va]_{\mathbb{G}} + [\mu]_{\mathbb{G}_T} = [t\gamma\beta uva^{N-1} + \mu]_{\mathbb{G}_T},
\end{aligned}$$

and outputs $\mathsf{ct} = ([C_1]_{\mathbb{G}}, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}}, [C_4]_{\mathbb{G}_T}, [C_5]_{\mathbb{G}_T})$

- $\mathsf{Dec}(\mathsf{mpk}, S, \mathsf{sk}, \mathsf{ct})$: On input the master public key mpk a set $S \subseteq [N]$, a secret key $\mathsf{sk} = (i, r_i, [s_i]_{\mathbb{G}})$ and a ciphertext ct, the decryption algorithm outputs ct if $\mathsf{mpk} = \bot$. Otherwise, it parses

$$\mathsf{mpk} = (\{[ua^j]_{\mathbb{G}}\}_{j \in [0, N-2]}, \{[va^k]_{\mathbb{G}}, [\beta va^k]_{\mathbb{G}}\}_{k \in [0,N]}, [\gamma\beta v]_{\mathbb{G}}, [\gamma\beta va]_{\mathbb{G}})$$

and $\mathsf{ct} = ([C_1]_{\mathbb{G}}, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}}, [C_4]_{\mathbb{G}_T}, [C_5]_{\mathbb{G}_T})$. Then, it outputs 0 if $i \notin S$. Otherwise, the decryption algorithm computes the degree-$(N-1)$ polynomial $P_i'(x) = P(x)/(x-i)$, where $P(x)$ is the polynomial from Eq. (C.1). In particular,

$$P_i'(x) = \left(\prod_{k \in S \setminus \{i\}}(x-k)\right) \cdot \left(\prod_{k \in [N] \setminus S}(x-k-N)\right). \tag{C.2}$$

Let $p_0', \dots, p_{N-1}'$ be the coefficients of P_i'. Namely, $P_i'(x) = \sum_{k=0}^{N-1} p_k' x^k$. Note that $p_{N-1}' = 1$. The algorithm then computes

$$\begin{aligned}
[C_6]_{\mathbb{G}_T} &= [s_i]_{\mathbb{G}} \cdot [C_1]_{\mathbb{G}} - \left(\sum_{j=0}^{N-2} p_j' [ua^j]_{\mathbb{G}}\right) \cdot ([C_2]_{\mathbb{G}} - r_i [C_3]_{\mathbb{G}}), \\
[\mu']_{\mathbb{G}_T} &= [C_5]_{\mathbb{G}_T} - [C_6]_{\mathbb{G}_T} - r_i \cdot [C_4]_{\mathbb{G}_T}.
\end{aligned}$$

The algorithm outputs 1 if $[\mu']_{\mathbb{G}_T} = [1]_{\mathbb{G}_T}$ and 0 otherwise.

Theorem C.3 (Correctness [GW09, adapted]). *Construction C.2 is correct.*

Proof. Take any $\lambda \in \mathbb{N}$ and any polynomial function $N = \mathrm{poly}(\lambda)$. Let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$. First, if $\mathsf{mpk} = \bot$, then correctness holds trivially (in this case, the encryption algorithm simply outputs the message in the clear and the decryption algorithm simply outputs the ciphertext). Suppose that $\mathsf{mpk} \neq \bot$. Then, we can write

$$\begin{aligned}
\mathsf{mpk} &= (\{[ua^j]_{\mathbb{G}}\}_{j \in [0, N-2]}, \{[va^k]_{\mathbb{G}}, [\beta va^k]_{\mathbb{G}}\}_{k \in [0,N]}, [\gamma\beta v]_{\mathbb{G}}, [\gamma\beta va]_{\mathbb{G}}), \\
\mathsf{msk} &= (a, u, v, \beta, \gamma, r_1, \dots, r_n).
\end{aligned}$$

where $a, u, v, \beta, \gamma, r_1, \dots, r_N \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $a \notin [N]$. Take any set $S \subseteq [N]$, any index $i \in S$, and any message $\mu \in \{0,1\}$. Suppose sk_i is in the support of $\mathsf{KeyGen}(\mathsf{msk}, i)$. Then, we can write $\mathsf{sk}_i = (i, r_i, [s_i]_{\mathbb{G}})$ where $s_i = \beta u \cdot \frac{\gamma - r_i}{a-i}$. Take any ciphertext $\mathsf{ct} = ([C_1]_{\mathbb{G}}, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}}, [C_4]_{\mathbb{G}_T}, [C_5]_{\mathbb{G}_T})$ in the support of $\mathsf{Enc}(\mathsf{mpk}, S, \mu)$. Let $t \in \mathbb{Z}_p$ be the encryption randomness. Consider the behavior of $\mathsf{Dec}(\mathsf{mpk}, S, \mathsf{sk}_i, \mathsf{ct})$. By definition, we have

$$[s_i]_{\mathbb{G}} \cdot [C_1]_{\mathbb{G}} = \left[\beta u \cdot \frac{\gamma - r_i}{a - i}\right]_{\mathbb{G}} \cdot [tvP(a)]_{\mathbb{G}} = \left[t\beta(\gamma - r_i)uv\frac{P(a)}{a-i}\right]_{\mathbb{G}_T} = [t\beta(\gamma - r_i)uvP_i'(a)]_{\mathbb{G}_T},$$

where $P_i'(x) = P(x)/(x-i)$ is the polynomial from Eq. (C.2). Since $p_{N-1}' = 1$, we can write

$$P_i'(x) = x^{N-1} + \sum_{k=0}^{N-2} p_k' x^k.$$

This means

$$\begin{aligned}\sum_{j=0}^{N-2} p_j' [ua^j]_{\mathbb{G}} \cdot ([C_2]_{\mathbb{G}} - r_i[C_3]_{\mathbb{G}}) &= [u(P_i'(a) - a^{N-1})]_{\mathbb{G}} \cdot [t(\gamma - r_i)\beta v]_{\mathbb{G}} \\ &= [t\beta(\gamma - r_i)uvP_i'(a)]_{\mathbb{G}_T} - [t\beta(\gamma - r_i)uva^{N-1}]_{\mathbb{G}_T}.\end{aligned}$$

Therefore,

$$\begin{aligned}[C_6]_{\mathbb{G}_T} &= [s_i]_{\mathbb{G}} \cdot [C_1]_{\mathbb{G}} - \left(\sum_{j=0}^{N-2} p_j' [ua^j]_{\mathbb{G}}\right) \cdot ([C_2]_{\mathbb{G}} - r_i[C_3]_{\mathbb{G}}) \\ &= [t\beta(\gamma - r_i)uva^{N-1}]_{\mathbb{G}_T} \\ &= [t\beta\gamma uva^{N-1}]_{\mathbb{G}_T} - r_i[t\beta uva^{N-1}]_{\mathbb{G}_T} \\ &= [C_5]_{\mathbb{G}_T} - r_i[C_4]_{\mathbb{G}_T} - [\mu]_{\mathbb{G}_T}.\end{aligned}$$

Thus $[\mu']_{\mathbb{G}_T} = [C_5]_{\mathbb{G}_T} - [C_6]_{\mathbb{G}_T} - r_i \cdot [C_4]_{\mathbb{G}_T} = [\mu]_{\mathbb{G}_T}$ and correctness holds. □

Theorem C.4 (Semi-Static Security [GW09, adapted]). *Assuming the decision q-BDHE sum problem (Assumption C.1) is hard with respect to* PrimeBGroupGen, *Construction C.2 is semi-statically secure.*

Proof. In the following, for any polynomial $P(x)$, we write $P|_i$ to denote the coefficient of x^i in P. The proof uses the following lemma from [GW09, §B], which we state below:

Lemma C.5 ([GW09, Lemma B.1]). *Let $P_1, P_2 \in \mathbb{Z}_p[x]$ be polynomials of degree d_1, d_2, respectively. Suppose moreover that P_1 and P_2 do not share any common factors. Let $d_3 = d_1 + d_2 - 1$ and $i \in [d_1, d_3]$. Then, there exists an efficiently-computable polynomial Q of degree d_3 such that*

- $P_1Q|_i = 1$;
- $\forall j \in [d_1, d_3] \setminus \{i\}, P_1Q|_j = 0$; *and*
- $\forall j \in [d_2, d_3], P_2Q|_j = 0$.

Proof of Theorem C.4. Suppose there exists an efficient adversary $\mathcal{A}$ that breaks semi-static security (Definition 3.4) of Construction C.2 with non-negligible advantage ε:

$$\left|\Pr[\mathsf{EXP}_{\mathsf{SSBE}}^{(0)}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}_{\mathsf{SSBE}}^{(1)}(1^\lambda, \mathcal{A}) = 1]\right| = \varepsilon(\lambda).$$

We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the decision q-BDHE sum problem (Assumption C.1) as follows:

- On input the security parameter 1^λ, algorithm $\mathcal{B}$ runs $\mathcal{A}(1^\lambda)$ to receive the number of users 1^N and the set S^*. Algorithm $\mathcal{B}$ also samples a random bit $\mu \xleftarrow{\text{R}} \{0,1\}$, which serves as the challenge bit of the simulated semi-static experiment.
- Algorithm $\mathcal{B}$ sets $m = 12N - 1$ and
$$S_0 = [0, N-2] \cup [3N, 5N-1] \cup [6N, 7N-1] \cup [9N, 11N] \cup [12N, 14N+1].$$
Algorithm $\mathcal{B}$ submits (S_0, m) to the q-BDHES challenger to receive the challenge $(\mathcal{G}, \{[a^i]_{\mathbb{G}}\}_{i \in S_0}, [Z]_{\mathbb{G}_T})$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e)$.
- Algorithm $\mathcal{B}$ samples $u_0, v_0, \beta_0 \xleftarrow{\text{R}} \mathbb{Z}_p$ and (implicitly) sets $u = u_0$, $v = v_0 a^{9N}$, $\beta = \beta_0 a^{3N}$.
- Algorithm $\mathcal{B}$ samples $z \xleftarrow{\text{R}} \mathbb{Z}_p$ and $r_i \xleftarrow{\text{R}} \mathbb{Z}_p$ for all $i \notin S^*$. It samples a random polynomial $\Gamma(x) = \sum_{j=0}^{2N} \gamma_j x^j$ of degree $2N$ such that $\Gamma(i) = r_i$ for all $i \notin S^*$ and $\Gamma(i) = z$ for all $i \in S^* \cup [N+1, 2N]$. It implicitly programs $\gamma = \Gamma(a)$. Since Γ has degree $2N$, and algorithm $\mathcal{B}$ only constrains the value of Γ on the points in $[2N]$, the value of $\gamma = \Gamma(a)$ is uniform over $\mathbb{Z}_p$ and independent of z and r_i for $i \notin S^*$ so long as $a \notin [2N]$.
- Algorithm $\mathcal{B}$ now computes the master public key as follows:
  - for all $j \in [0, N-2]$, $[ua^j]_{\mathbb{G}} = u_0 [a^j]_{\mathbb{G}}$;
  - for all $k \in [0, N]$, $[va^k]_{\mathbb{G}} = v_0 [a^{9N+k}]_{\mathbb{G}}$, $[\beta v a^k]_{\mathbb{G}} = \beta_0 v_0 [a^{12N+k}]_{\mathbb{G}}$;
  - $[\gamma \beta v]_{\mathbb{G}} = \beta_0 v_0 [\Gamma(a) a^{12N}]_{\mathbb{G}} = \beta_0 v_0 \sum_{j=0}^{2N} \gamma_j [a^{12N+j}]_{\mathbb{G}}$; and
  - $[\gamma \beta v a]_{\mathbb{G}} = \beta_0 v_0 \sum_{j=0}^{2N} \gamma_j [a^{12N+j+1}]_{\mathbb{G}}$.

  Algorithm $\mathcal{B}$ sends $\mathsf{mpk} = (\{[ua^j]_{\mathbb{G}}\}_{j \in [0,N-2]}, \{[va^k]_{\mathbb{G}}, [\beta v a^k]_{\mathbb{G}}\}_{k \in [0,N]}, [\gamma\beta v]_{\mathbb{G}}, [\gamma \beta v a]_{\mathbb{G}})$ to $\mathcal{A}$.
- Whenever $\mathcal{A}$ makes a key-generation query on an index $i \in [N] \setminus S^*$, algorithm $\mathcal{B}$ computes the polynomial $\Gamma_i'(x) = \frac{\Gamma(x) - r_i}{x - i}$. Since $i \notin S^*$, $\Gamma(i) = r_i$, so i is a root of the polynomial $\Gamma(x) - r_i$. This means $(x - i)$ is a factor of $\Gamma(x) - r_i$, so we can express Γ_i' as a polynomial of degree $2N - 1$. Algorithm $\mathcal{B}$ writes $\Gamma_i'(x) = \sum_{j=0}^{2N-1} \gamma_j' x^j$ and simulates the secret key by computing
$$[s_i]_{\mathbb{G}} = \left[\beta u \cdot \frac{\gamma - r_i}{a - i}\right]_{\mathbb{G}} = \left[\beta u \cdot \frac{\Gamma(a) - r_i}{a - i}\right]_{\mathbb{G}} = [\beta u \cdot \Gamma_i'(a)]_{\mathbb{G}} = \beta_0 u_0 \sum_{j=0}^{2N-1} \gamma_j' [a^{3N+j}]_{\mathbb{G}}$$
Algorithm $\mathcal{B}$ gives $\mathsf{sk}_i = (i, r_i, [s_i]_{\mathbb{G}})$ to $\mathcal{A}$.
- After algorithm $\mathcal{A}$ finishes making its key-generation queries, it submits a set $S \subseteq S^*$ to $\mathcal{B}$. Algorithm $\mathcal{B}$ computes the degree-N polynomial
$$P(x) = \left(\prod_{k \in S} (x - k)\right) \cdot \left(\prod_{k \in [N] \setminus S} (x - k - N)\right).$$
As usual, let $p_0, \ldots, p_N$ be the coefficients of $P(x)$. Namely, write $P(x) = \sum_{k=0}^{N} p_k x^k$. Invoking Lemma C.5 with $(P_1, P_2, i) = (\Gamma, P, 2N)$, algorithm $\mathcal{B}$ computes the polynomial $T_0 = \sum_{k=0}^{3N-1} t_k x^k$ of degree $3N - 1$ such that
$$\begin{aligned} \Gamma T_0|_{2N} &= 1 \\ \Gamma T_0|_j &= 0 \quad \forall j \in [2N+1, 3N-1] \\ P T_0|_j &= 0 \qquad \forall j \in [N, 3N-1]. \end{aligned} \tag{C.3}$$
It then samples $\tau \xleftarrow{\text{R}} \mathbb{Z}_p$ and implicitly programs $t = \tau + z \cdot a^{-3N} T_0(a)$.

- Algorithm $\mathcal{B}$ simulates the challenge ciphertext components as follows:

$$
\begin{aligned}
[C_1]_{\mathbb{G}} &= [tvP(a)]_{\mathbb{G}} = v_0\tau[a^{9N}P(a)]_{\mathbb{G}} + v_0 z[a^{6N}T_0(a)P(a)]_{\mathbb{G}} \\
[C_2]_{\mathbb{G}} &= [t\gamma\beta v]_{\mathbb{G}} = v_0\beta_0\tau[a^{12N}\Gamma(a)]_{\mathbb{G}} + zv_0\beta_0[a^{9N}T_0(a)\Gamma(a)]_{\mathbb{G}} \\
[C_3]_{\mathbb{G}} &= [t\beta v]_{\mathbb{G}} = v_0\beta_0\tau[a^{12N}]_{\mathbb{G}} + zv_0\beta_0[a^{9N}T_0(a)]_{\mathbb{G}} \\
[C_4]_{\mathbb{G}_T} &= u_0v_0\beta_0\tau[a^{13N-1}]_{\mathbb{G}_T} + zu_0v_0\beta_0[a^{10N-1}(T_0(a) - z^{-1}a^{2N})]_{\mathbb{G}_T} + u_0v_0\beta_0[Z]_{\mathbb{G}_T} \\
[C_5]_{\mathbb{G}_T} &= u_0v_0\beta_0\tau[a^{13N-1}\Gamma(a)]_{\mathbb{G}_T} + zu_0v_0\beta_0[a^{10N-1}(T_0(a)\Gamma(a) - a^{2N})]_{\mathbb{G}_T} + zu_0v_0\beta_0[Z]_{\mathbb{G}_T} + [\mu]_{\mathbb{G}_T}.
\end{aligned}
$$

  Algorithm $\mathcal{B}$ gives $\mathsf{ct} = ([C_1]_{\mathbb{G}}, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}}, [C_4]_{\mathbb{G}_T}, [C_5]_{\mathbb{G}_T})$ to $\mathcal{A}$. Below, we will show how $\mathcal{B}$ can compute each of these terms from the corresponding terms in the challenge.

- At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit μ'. Algorithm $\mathcal{B}$ outputs 0 if $\mu = \mu'$ and outputs 1 otherwise.

Efficiency analysis of $\mathcal{B}$. First, we show that $\mathcal{B}$ is efficient. In particular, we show that algorithm $\mathcal{B}$ is able to efficiently compute each term shown in the above reduction using the group elements from the q-BDHES challenge $(\mathcal{G}, \{[a^i]_{\mathbb{G}}\}_{i\in S_0}, [Z]_{\mathbb{G}_T})$. Recall that

$$S_0 = [0, N-2] \cup [3N, 5N-1] \cup [6N, 7N-1] \cup [9N, 11N] \cup [12N, 14N+1].$$

In the following analysis, we assume that $a \notin [2N]$. Since the challenger samples $a \xleftarrow{\text{R}} \mathbb{Z}_p$ and $N = \mathsf{poly}(\lambda)$, this holds with overwhelming probability over the choice of a. We consider each component individually:

- The master public key mpk is efficiently computable since $[0, N-2]$ (for $[ua^j]_{\mathbb{G}}$), $[9N, 10N]$ (for $[va^k]_{\mathbb{G}}$), $[12N, 13N]$ (for $[\beta va^k]_{\mathbb{G}}$), and $[12N, 14N+1]$ (for $[\gamma\beta v]_{\mathbb{G}}$ and $[\gamma\beta va]_{\mathbb{G}}$) are all subsets of S_0.
- Next, the key-generation queries only require $[a^i]$ for $i \in [3N, 5N-1] \subset S_0$.
- To argue that the challenge ciphertext ct is efficiently computable, we first affirm that the hypothesis of Lemma C.5 is satisfied:
  - By construction, the roots of the polynomial P are at the points x where either $x \in S$ or $x = N + k$ for some $k \in [N] \setminus S$.
  - Since $S \subseteq S^*$, we conclude that all of the roots of P are contained in the set $S^* \cup [N+1, 2N]$. By construction, the polynomial Γ has value z on every $x \in S^* \cup [N+1, 2N]$. Since the reduction algorithm samples $z \xleftarrow{\text{R}} \mathbb{Z}_p$, with overwhelming probability $z \neq 0$, which means Γ does not have roots at $x \in S^* \cup [N+1, 2N]$.

  Then, the following two properties hold:

  - With overwhelming probability over the choice of z, Γ and P do not share any common factors. Lemma C.5 now guarantees the existence of the polynomial T_0 of degree $3N-1$ that satisfies the properties in Eq. (C.3).
  - Second, the polynomial $(z^{-1}\Gamma(x) - 1)$ has roots on all $x \in S^* \cup [N+1, 2N]$. Since all of the roots of P are contained in this interval, this means P divides the polynomial $(z^{-1}\Gamma - 1)$. Equivalently, there exists a polynomial $Q(x)$ of degree at most N where $P(x) \cdot Q(x) = z^{-1} \cdot \Gamma(x) - 1$.

  From Eq. (C.3), we have $PT_0|_j = 0$ for all $j \in [N, 3N-1]$. Since Q has degree at most N, this means

$$\forall j \in [2N, 3N-1] : PQT_0|_j = 0 \implies (z^{-1}\Gamma - 1)T_0|_j = z^{-1}\Gamma T_0|_j - T_0|_j = 0. \qquad \text{(C.4)}$$

  Again by Eq. (C.3), we know that $\Gamma T_0|_{2N} = 1$, and that for all $j \in [2N+1, 3N-1]$, $\Gamma T_0|_j = 0$. Combining with Eq. (C.4), we have

$$
\begin{aligned}
T_0|_{2N} &= z^{-1}\Gamma T_0|_{2N} = z^{-1} \\
T_0|_j &= z^{-1}\Gamma T_0|_j = 0 \qquad \forall j \in [2N+1, 3N-1].
\end{aligned}
$$

Finally, since T_0 is a polynomial with degree at most $3N-1$, this means that T_0 has degree exactly $2N$ and leading coefficient z^{-1}. Moreover, we note that the reduction algorithm knows the coefficients of the polynomials P, Γ, T_0 as well as the exponents $u_0, v_0, \beta_0, z, \tau \in \mathbb{Z}_p$. We now consider each of the components in the challenge ciphertext:

- To compute $[C_1]_{\mathbb{G}}$, the reduction needs to compute $[a^{9N}P(a)]_{\mathbb{G}}$ and $[a^{6N}T_0(a)P(a)]_{\mathbb{G}}$:
  * Since P has degree at most N, the reduction can compute $[a^{9N}P(a)]_{\mathbb{G}}$ using $[a^i]_{\mathbb{G}}$ for $i \in [9N, 10N] \subset S_0$.
  * From Eq. (C.3), $PT_0|_j = 0$ for all $j \in [N, 3N-1]$. Moreover, PT_0 has degree at most $3N$. Thus, the reduction can compute $[a^{6N}T_0(a)P(a)]_{\mathbb{G}}$ using $[a^i]_{\mathbb{G}}$ for $i \in [6N, 7N-1] \cup \{9N\} \subseteq S_0$.
- To compute $[C_2]_{\mathbb{G}}$, the reduction needs to compute $[a^{12N}\Gamma(a)]_{\mathbb{G}}$ and $[a^{9N}T_0(a)\Gamma(a)]_{\mathbb{G}}$:
  * Since Γ has degree at most $2N$, the reduction can compute $[a^{12N}\Gamma(a)]_{\mathbb{G}}$ using $[a^i]_{\mathbb{G}}$ for $i \in [12N, 14N] \subset S_0$.
  * By Eq. (C.3), $\Gamma T_0|_j = 0$ for all $j \in [2N+1, 3N-1]$. Moreover ΓT_0 has degree at most $4N$. This means the reduction can compute $[a^{9N}T_0(a)\Gamma(a)]_{\mathbb{G}}$ using $[a^i]_{\mathbb{G}}$ for $i \in [9N, 11N] \cup [12N, 13N] \subseteq S_0$.
- To compute $[C_3]_{\mathbb{G}}$, the reduction needs to first compute $[a^{12N}]_{\mathbb{G}}$, which is given out since $12N \in S_0$. Then it needs to compute $[a^{9N}T_0(a)]_{\mathbb{G}}$. Since T_0 has degree $2N$, the reduction can do so using $[a^i]_{\mathbb{G}}$ for $i \in [9N, 11N] \subseteq S_0$.
- To compute $[C_4]_{\mathbb{G}_T}$, the reduction needs to compute $[a^{13N-1}]_{\mathbb{G}_T}$ and $[a^{10N-1}(T_0(a) - z^{-1}a^{2N})]_{\mathbb{G}_T}$:
  * First, it computes $[a^{13N-1}]_{\mathbb{G}_T} = [a^{13N-1}]_{\mathbb{G}} \cdot [1]_{\mathbb{G}}$. Note that $13N-1 \in S_0$.
  * Consider $[a^{10N-1}(T_0(a) - z^{-1}a^{2N})]_{\mathbb{G}_T}$. From above, we argued that T_0 has degree $2N$ with leading coefficient z^{-1}. This means $T_0(a) - z^{-1}a^{2N}$ is a polynomial of degree $2N-1$. Thus, the reduction needs to be able to compute $[a^i]_{\mathbb{G}_T}$ for $i \in [10N-1, 12N-2]$. For each $i \in [10N-1, 12N-2]$, the reduction can compute
  $$[a^i]_{\mathbb{G}_T} = [a^{7N-1}]_{\mathbb{G}} \cdot [a^{i-7N+1}]_{\mathbb{G}}.$$
  Now $7N-1 \in S_0$ and moreover, when $i \in [10N-1, 12N-2]$, we have $i - 7N + 1 \in [3N, 5N-1] \subset S_0$.
- To compute $[C_5]_{\mathbb{G}_T}$, the reduction needs to compute $[a^{13N-1}\Gamma(a)]_{\mathbb{G}_T}$ and $[a^{10N-1}(T_0(a)\Gamma(a) - a^{2N})]_{\mathbb{G}_T}$:
  * Since Γ has degree at most $2N$, to compute $[a^{13N-1}\Gamma(a)]_{\mathbb{G}_T}$, it suffices to be able to compute $[a^i]_{\mathbb{G}_T}$ for $i \in [13N-1, 15N-1]$. For each $i \in [13N-1, 15N-1]$, the reduction can compute
  $$[a^i]_{\mathbb{G}_T} = [a^{4N-1}]_{\mathbb{G}} \cdot [a^{i-4N+1}]_{\mathbb{G}},$$
  which is feasible since $4N-1 \in S_0$ and likewise, $i - 4N + 1 \in [9N, 11N] \subset S_0$.
  * By Eq. (C.3), $\Gamma T_0|_j = 0$ for all $j \in [2N+1, 3N-1]$ and $\Gamma T_0|_{2N} = 1$. This means $(T_0(a)\Gamma(a) - a^{2N})|_j = 0$ for all $j \in [2N, 3N-1]$. In addition, the degree of $T_0\Gamma$ is at most $4N$. Thus, to compute $[a^{10N-1}(T_0(a)\Gamma(a) - a^{2N})]_{\mathbb{G}_T}$, it suffices that the reduction can compute $[a^i]_{\mathbb{G}_T}$ for all $i \in [10N-1, 12N-2] \cup [13N-1, 14N-1]$. First, for $i \in [10N-1, 12N-2]$, the reduction can compute
  $$[a^i]_{\mathbb{G}_T} = [a^{7N-1}]_{\mathbb{G}} \cdot [a^{i-7N+1}]_{\mathbb{G}},$$
  which is feasible since $7N-1 \in S_0$ and $i - 7N + 1 \in [3N, 5N-1] \subset S_0$. For $i \in [13N-1, 14N-1]$, the reduction can compute
  $$[a^i]_{\mathbb{G}_T} = [a^{10N-1}]_{\mathbb{G}} \cdot [a^{i-10N+1}]_{\mathbb{G}},$$
  which is feasible since $10N-1 \in S_0$ and $i - 10N + 1 \in [3N, 4N] \subset S_0$.

We conclude that algorithm $\mathcal{B}$ can efficiently construct the listed group elements from the elements given out in the challenge.

Advantage analysis of $\mathcal{B}$. It suffices now to compute the advantage of $\mathcal{B}$. First, by inspection, there does not exist $s, s' \in S_0 \cup \{0\}$ such that $s + s' = 12N - 1$, so algorithm $\mathcal{B}$ is admissible. As in the correctness analysis, we assume that $a \notin [2N]$. Since the challenger samples $a \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $N = \mathsf{poly}(\lambda)$ while $p = 2^{\Omega(\lambda)}$, this property holds with overwhelming probability. We now consider two possibilities depending on the distribution of the challenge $[Z]_{\mathbb{G}_T}$:

- Suppose $[Z]_{\mathbb{G}_T} = [a^m]_{\mathbb{G}_T} = [a^{12N-1}]_{\mathbb{G}_T}$. In this case, we claim that with overwhelming probability, algorithm $\mathcal{B}$ simulates the semi-static security experiment q-$\mathsf{BDHES}^{(\mu)}$, where μ is the random (message) bit sampled by $\mathcal{B}$ at the start of its execution. We analyze the distribution of the master public key mpk, the responses sk_i to the key-generation queries, and the challenge ciphertext ct:
  - **Computation of** mpk**.** First, the q-BDHES challenger samples $a \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Since the reduction algorithm samples $u_0, v_0, \beta_0 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, as long as $a \neq 0$, the distributions of $u = u_0, v = v_0 a^{9N}, \beta = \beta_0 a^{3N}$ are independent and uniformly random. Furthermore, since Γ is a random $2N$-degree polynomial with $2N$ fixed points, the evaluation outcome $\gamma = \Gamma(a)$ at any point $a \notin [2N]$ is uniformly random. Thus as long as $a \notin [2N]$, the master public key
  $$\mathsf{mpk} = (\{[ua^j]_{\mathbb{G}}\}_{j \in [0,N-2]}, \{[va^k]_{\mathbb{G}}, [\beta va^k]_{\mathbb{G}}\}_{k \in [0,N]}, [\gamma\beta v]_{\mathbb{G}}, [\gamma\beta va]_{\mathbb{G}}).$$
  constructed by $\mathcal{B}$ is distributed exactly according to the specification of the semi-static experiment. As argued above, $a \notin [2N]$ with overwhelming probability over the choice of a.
  - **Computation of** sk_i**.** For each index $i \in [N] \setminus S^*$, algorithm $\mathcal{B}$ sampled $r_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, which is distributed exactly according to the specification of the semi-static experiment. Since algorithm $\mathcal{B}$ constructs $[s_i]_{\mathbb{G}}$ to satisfy $[s_i]_{\mathbb{G}} = [\beta u \cdot \frac{\gamma - r_i}{a-i}]_{\mathbb{G}}$, we conclude that the secret keys sk_i are also distributed as specified in the semi-static experiment.
  - **Computation of** ct. First, the reduction algorithm implicitly defines $t = \tau + z \cdot a^{-3N} T_0(a)$ where $\tau \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Since τ is uniform, the same holds for t. Now, if $[Z]_{\mathbb{G}_T} = [a^{12N-1}]_{\mathbb{G}_T}$, the challenge ciphertext $\mathcal{B}$ computes can be written as follows:
  $$\begin{aligned}
  [C_1]_{\mathbb{G}} &= [tvP(a)]_{\mathbb{G}} \\
  [C_2]_{\mathbb{G}} &= [t\gamma\beta v]_{\mathbb{G}} \\
  [C_3]_{\mathbb{G}} &= [t\beta v]_{\mathbb{G}} \\
  [C_4]_{\mathbb{G}_T} &= u_0 v_0 \beta_0 \tau [a^{13N-1}]_{\mathbb{G}_T} + z u_0 v_0 \beta_0 [a^{10N-1}(T_0(a) - z^{-1} a^{2N})]_{\mathbb{G}_T} + u_0 v_0 \beta_0 [Z]_{\mathbb{G}_T} \\
  &= [uv\beta\tau a^{N-1}]_{\mathbb{G}_T} + [uv\beta(za^{-2N-1} T_0(a) - a^{-1})]_{\mathbb{G}_T} + [uv\beta a^{-1}]_{\mathbb{G}_T} \\
  &= [uv\beta a^{N-1}(\tau + za^{-3N} T_0(a))]_{\mathbb{G}_T} \\
  &= [uv\beta a^{N-1} t]_{\mathbb{G}_T} \\
  [C_5]_{\mathbb{G}_T} &= u_0 v_0 \beta_0 \tau [a^{13N-1}\Gamma(a)]_{\mathbb{G}_T} + z u_0 v_0 \beta_0 [a^{10N-1}(T_0(a)\Gamma(a) - a^{2N})]_{\mathbb{G}_T} + z u_0 v_0 \beta_0 [Z]_{\mathbb{G}_T} + [\mu]_{\mathbb{G}_T} \\
  &= [uv\beta\tau\gamma a^{N-1}]_{\mathbb{G}_T} + [uv\beta z(a^{-2N-1} T_0(a)\gamma - a^{-1})]_{\mathbb{G}_T} + [uv\beta z a^{-1}]_{\mathbb{G}_T} + [\mu]_{\mathbb{G}_T} \\
  &= [uv\beta\gamma a^{N-1}(\tau + za^{-3N} T_0(a)) + \mu]_{\mathbb{G}_T} \\
  &= [uv\beta\gamma a^{N-1} t + \mu]_{\mathbb{G}_T},
  \end{aligned}$$
  which is distributed exactly as in the real semi-static experiment.

  Thus, with overwhelming probability over the choice of a, we have
  $$\begin{aligned}
  \Pr[q\text{-}\mathsf{BDHES}^{(0)}(1^\lambda, \mathcal{B}) = 1] &= \frac{1}{2} \Pr[\mathsf{EXP}^{(0)}_{\mathsf{SSBE}}(1^\lambda, \mathcal{A}) = 0] + \frac{1}{2} \Pr[\mathsf{EXP}^{(1)}_{\mathsf{SSBE}}(1^\lambda, \mathcal{A}) = 1] + \nu(\lambda) \\
  &= \frac{1}{2} - \frac{1}{2}\big(\Pr[\mathsf{EXP}^{(0)}_{\mathsf{SSBE}}(1^\lambda, \mathcal{A}) = 1] - \Pr[\mathsf{EXP}^{(1)}_{\mathsf{SSBE}}(1^\lambda, \mathcal{A}) = 1]\big) + \nu(\lambda),
  \end{aligned}$$
  where $|\nu(\lambda)| \leq \mathsf{negl}(\lambda)$. We conclude that
  $$\left| \Pr[q\text{-}\mathsf{BDHES}^{(0)}(1^\lambda, \mathcal{B}) = 1] - \frac{1}{2} \right| \geq \frac{\varepsilon(\lambda)}{2} - \mathsf{negl}(\lambda). \tag{C.5}$$

- Suppose $[Z]_{\mathbb{G}_T} \xleftarrow{\mathrm{R}} \mathbb{G}_T$. Then, we can equivalently write $[Z]_{\mathbb{G}_T} = [a^{12N-1}1 + \delta]_{\mathbb{G}_T}$ where $\delta \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. By the same analysis as in the previous case, conditioned on $a \notin [2N]$, the components constructed by algorithm $\mathcal{B}$ are distributed as follows:

$$\begin{aligned} \mathsf{mpk} &= \left(\{[ua^j]_{\mathbb{G}}\}_{j\in[0,N-2]}, \{[va^k]_{\mathbb{G}}, [\beta va^k]_{\mathbb{G}}\}_{k\in[0,N]}, [\gamma\beta v]_{\mathbb{G}}, [\gamma\beta va]_{\mathbb{G}}\right), \\ \mathsf{sk}_i &= \left(i, r_i, \left[\beta u \cdot \frac{\gamma - r_i}{a - i}\right]_{\mathbb{G}}\right), \end{aligned}$$

and the challenge ciphertext components $\mathsf{ct} = ([C_1]_{\mathbb{G}}, [C_2]_{\mathbb{G}}, [C_3]_{\mathbb{G}}, [C_4]_{\mathbb{G}_T}, [C_5]_{\mathbb{G}_T})$ are distributed as follows:

$$\begin{aligned} [C_1]_{\mathbb{G}} &= [tvP(a)]_{\mathbb{G}} \\ [C_2]_{\mathbb{G}} &= [t\gamma\beta v]_{\mathbb{G}} \\ [C_3]_{\mathbb{G}} &= [t\beta v]_{\mathbb{G}}, \\ [C_4]_{\mathbb{G}_T} &= [uv\beta a^{N-1}t]_{\mathbb{G}_T} + [u_0v_0\beta_0\delta]_{\mathbb{G}_T} \\ [C_5]_{\mathbb{G}_T} &= [uv\beta\gamma a^{N-1}t]_{\mathbb{G}_T} + [zu_0v_0\beta_0\delta + \mu]_{\mathbb{G}_T}. \end{aligned}$$

Now as long as $a \notin [2N]$, the values of $u, v, \beta, \{r_i\}_{i\notin S^*}, \gamma, t$ are independent of z. Specifically, when $a \notin [2N]$, the value of $\gamma = \Gamma(a)$ is uniform and independent over $\mathbb{Z}_p$ (because Γ is a polynomial of degree $2N$ that is constrained on only $2N$ points). Similarly, the value of t is blinded by $\tau \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, which is sampled independently of all of the other parameters. Thus, as long as $u_0, v_0, \beta_0 \neq 0$ (which holds with overwhelming probability), the distribution of $zu_0v_0\beta_0\delta$ remains uniform over $\mathbb{Z}_p$ even given all of the other components (including $u_0v_0\beta_0\delta$). In particular, this means that the distribution of $[C_5]_{\mathbb{G}_T}$ is statistically close to uniform over $\mathbb{G}_T$ (independent of the message μ). Correspondingly, this means

$$\left|\Pr[q\text{-}\mathsf{BDHES}^{(1)}(1^\lambda, \mathcal{B}) = 1] - \frac{1}{2}\right| = \mathsf{negl}(\lambda). \tag{C.6}$$

Combining Eqs. (C.5) and (C.6), we have

$$\begin{aligned} &\left|\Pr[q\text{-}\mathsf{BDHES}^{(0)}(1^\lambda, \mathcal{B}) = 1] - \Pr[q\text{-}\mathsf{BDHES}^{(1)}(1^\lambda, \mathcal{B}) = 1]\right| \\ &\quad\geq \left|\Pr[q\text{-}\mathsf{BDHES}^{(0)}(1^\lambda, \mathcal{B}) = 1] - 1/2\right| - \left|\Pr[q\text{-}\mathsf{BDHES}^{(1)}(1^\lambda, \mathcal{B}) = 1] - 1/2\right| \\ &\quad\geq \frac{\varepsilon(\lambda)}{2} - \mathsf{negl}(\lambda), \end{aligned}$$

which is non-negligible. Thus $\mathcal{B}$ breaks the q-BDHES assumption, and the theorem holds. □