How to Use (Plain) Witness Encryption: Registered ABE, Flexible Broadcast, and More

Cody Freitag
Boston University
freitag@bu.edu

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

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

Abstract

Witness encryption is a generalization of public-key encryption where the public key can be any NP statement x and the associated decryption key is any witness w for x. While early constructions of witness encryption relied on multilinear maps and indistinguishability obfuscation ($i\mathcal{O}$), recent works have provided direct constructions of witness encryption that are more efficient than $i\mathcal{O}$ (and also seem unlikely to yield $i\mathcal{O}$). Motivated by this progress, we revisit the possibility of using witness encryption to realize advanced cryptographic primitives previously known only in "obfustopia."

In this work, we give new constructions of *trustless* encryption systems from *plain* witness encryption (in conjunction with the learning-with-errors assumption): (1) flexible broadcast encryption (a broadcast encryption scheme where users choose their *own* secret keys and users can encrypt to an *arbitrary* set of public keys); and (2) registered attribute-based encryption (a system where users choose their own keys and then register their public key together with a set of attributes with a deterministic and transparent key curator). Both primitives were previously only known from $i\mathcal{O}$. We also show how to use our techniques to obtain an *optimal* broadcast encryption scheme in the random oracle model.

Underlying our constructions is a novel technique for using witness encryption based on a new primitive which we call *function-binding hash functions*. Whereas a somewhere statistically binding hash function statistically binds a digest to a few bits of the input, a function-binding hash function statistically binds a digest to the *output* of a function of the inputs. As we demonstrate in this work, function-binding hash functions provide us new ways to leverage the power of plain witness encryption and use it as the foundation of advanced cryptographic primitives. Finally, we show how to build function-binding hash functions for the class of disjunctions of block functions from leveled homomorphic encryption; this in combination with witness encryption yields our main results.

1 Introduction

In the last decade, indistinguishability obfuscation ($i\mathcal{O}$) [BGI+01, GGH+13] has emerged as a central hub for cryptography, and in combination with one-way functions or somewhere statistically binding hash functions [HW15], have yielded solutions to a broad range of cryptographic problems [SW14, BZ14, GGHR14, BPR15, CLP15, KRR17, AL18, CPP20]; we refer to [JLS21, Corollary 1.1] for a more comprehensive list. Witness encryption [GGSW13] is another cryptographic notion that is often considered alongside indistinguishability obfuscation, and has been used successfully to construct a number of cryptographic primitives such as public-key encryption, identity-based encryption, attribute-based encryption [GGSW13], oblivious transfer [BGI+17], laconic arguments [FNV17, BISW18], and null-$i\mathcal{O}$ [GKW17, WZ17]. However, compared to indistinguishability obfuscation, there has been significantly less success in leveraging witness encryption to realize other cryptographic notions, and indeed, many of the notions implied by witness encryption can be achieved directly from standard number-theoretic or lattice-based assumptions.

Witness encryption. Intuitively, witness encryption [GGSW13] is a generalization of public-key encryption where the public key can be any NP statement x and the associated decryption key is any witness w for x. Here, a user can encrypt a message m with respect to the statement x and anyone who knows a witness w for x can decrypt. The security requirement is that if the statement x is false, then the message is computationally hidden.

Although witness encryption appears to be a substantially weaker primitive than $i\mathcal{O}$ (both in terms of its applications and with respect to black-box separations [GMM17]), early constructions nonetheless either relied on $i\mathcal{O}$ [GGH+13] or techniques that sufficed for $i\mathcal{O}$ (e.g., assumptions over multilinear maps [GGSW13, GLW14]). This state of affairs changed recently with several works providing constructions from lattice-based assumptions [CVW18, Tsa22, VWW22] or suggesting new routes from group-based approaches [BIOW20]. Of particular note, the recent results [Tsa22, VWW22] construct witness encryption from the "evasive LWE" assumption, a new knowledge assumption on lattices [Wee22, Tsa22]. These works give a direct construction of witness encryption that is significantly more efficient than existing $i\mathcal{O}$ approaches. Moreover, the current techniques do not seem to extend to give $i\mathcal{O}$.

This work: cryptography from *plain* witness encryption. Motivated by the recent progress in constructing witness encryption, our goal in this work is to revisit the possibility of using witness encryption to realize advanced cryptographic primitives. Much like how the punctured programming framework [SW14] and notions like somewhere statistically binding hash functions [HW15] allowed us to leverage the power of $i\mathcal{O}$, our goal is to develop analogous techniques in the setting of (plain) witness encryption. Previous approaches to use witness encryption for realizing new cryptographic primitives have focused on augmenting witness encryption with additional properties such as "position-hiding" [GLW14, GVW19] or "extractability" [GKP+13, FNV17]. However, positional witness encryption is only known from $i\mathcal{O}$ [GLW14, GVW19] while extractable witness encryption currently relies on a knowledge assumption over multilinear maps or the stronger notion of differing-inputs obfuscation [GKP+13]. Moreover, the recent advancements in construction witness encryption seem insufficient for realizing these stronger notions. As such, we focus on techniques that use *plain* witness encryption.

Function-binding hash functions. In this work, we introduce a new cryptographic primitive called a *function-binding hash function*. Similar to how somewhere statistically binding (SSB) hash functions [HW15] statistically bind to a few bits of an *input* to the hash function, our function-binding hash function statistically binds to the *output* of a *function* of the inputs. In fact, we can view SSB hash functions as a special case of function-binding hash functions for the class of index functions (see Remark 3.2). Similar to the setting of SSB hash function, a function-binding hash function should allow a user to *locally* open one or more bits of the input. The size of the opening should be small compared to the size of the input.

To illustrate the function-binding property, consider a (keyed) hash function H that is statistically binding for the OR function. Suppose that $\mathsf{dig} \leftarrow H(\mathsf{hk}, b_1, \ldots, b_n)$ where hk is the hash key and $b_1 = \cdots = b_n = 0$. If H is (statistically) function-binding for the OR function, then it should not be possible to locally open *any* index i to a value $b'_i = 1$. This is because the output of the OR function on all inputs $(b'_1, \ldots, b'_n)$ is $1 \neq 0$, irrespective of the values of b'_j for $j \neq i$. More generally, a function-binding hash function only supports local openings to values that are consistent with the output of the associated function (i.e., the locally-opened value(s) can be extended into an input to the function that maps to the value associated with the digest).

We provide a formal definition of this notion in Section 3 and a high-level overview in Section 1.1. We then show how to construct a function-binding hash function using leveled homomorphic encryption, which can in turn be based on standard lattice assumptions [BV11, BGV12, Bra12, GSW13]. Our construction here follows a similar structure as the construction of somewhere statistically binding hash functions [HW15] from leveled homomorphic encryption.

Our results. Using function-binding hash functions together with plain witness encryption, we are able to realize new *trustless* encryption notions like flexible broadcast encryption and registered attribute-based encryption (for general policies) [HLWW23] without indistinguishability obfuscation. These examples are intended to illustrate the usefulness of function-binding hash functions in conjunction with witness encryption, and we expect to see further applications in the future. We summarize our main results below:

- **Flexible broadcast encryption.** In a flexible broadcast encryption scheme, users generate their own public/private key-pair $(\mathsf{pk}, \mathsf{sk})$ (much like in vanilla public-key encryption) and then post their public key pk to a public bulletin board. The encryption algorithm takes a collection of user public keys $S = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ along with a message m and outputs a single ciphertext ct that can be decrypted by any user who possesses a secret key corresponding to a public key in the broadcast set S. The size of the ciphertext ct scales *polylogarithmically* with the size of the broadcast set S. Flexible broadcast encryption generalizes both the traditional notion of broadcast encryption [FN93] which assumes a central *trusted* authority issues keys to users as well as the

notion of distributed broadcast encryption [BZ14]. In distributed broadcast encryption, users generate their own public and private keys, but the syntax stipulates that each user is associated with a particular "index" (like in standard broadcast encryption); in turn, one can only encrypt to users associated with different indices. Flexible broadcast encryption eliminates all user coordination and supports encryption to an arbitrary set of user public keys. Previously, distributed broadcast encryption was only known from $i\mathcal{O}$ [BZ14]. In this work, we show how to construct flexible broadcast encryption from plain witness encryption and function-binding hash functions (which implies a distributed broadcast encryption scheme). Previously, it was not even known how to obtain *vanilla* broadcast encryption from witness encryption; instead, previous approaches relied on the stronger notion of positional witness encryption [GLW14, GVW19].

- **Registered attribute-based encryption.** A second application we show is to registered attribute-based encryption [HLWW23]. Much like how flexible broadcast encryption removes the trusted key issuer from broadcast encryption, registered ABE removes the trusted key issuer from traditional ABE [SW05, GPSW06]. In the registered ABE model, instead of a key issuer generating decryption keys for user attributes, users instead generate their own public/secret keys and then *register* their public key along with their attributes with a key curator. The key curator in this case is a deterministic and transparent algorithm that is responsible for aggregating the keys and attributes for the different users into a short master public key (whose size is polylogarithmic in the number of registered users). The aggregated public key functions as a public key for a standard ABE scheme. Since the key curator maintains no long-term secrets and can be publicly audited, registered ABE serves as a new paradigm for enabling the access-control capabilities of ABE without introducing a trusted key-issuing authority into the picture. Previously, the work of [HLWW23] showed how to construct registered ABE for an a priori bounded number of users using pairing-based assumptions in a model with a structured common reference string, as well as a scheme for an unbounded number of users in the common *random* string model from $i\mathcal{O}$. In this work, we show how to obtain a registered ABE scheme with the same properties as the previous construction from $i\mathcal{O}$ using plain witness encryption and function-binding hash functions.

- **Optimal broadcast encryption.** As an additional application, we show how to adapt our flexible broadcast encryption scheme to obtain an *optimal* broadcast encryption scheme (in the traditional centralized model with a trusted key issuer) in the random oracle model from plain witness encryption and function-binding hash functions. An optimal broadcast encryption scheme is one where all of the scheme parameters (public key size, secret key size, and ciphertext size) all scale polylogarithmically with the number of users n, and was previously known from multilinear maps [BWZ14], *positional* witness encryption [GVW19], on combinations of (non-falsifiable or idealized) pairing-based and lattice-based assumptions [AY20, AWY20], or from new (non-falsifiable) lattice assumptions [BV22, Wee22].

By instantiating the underlying witness encryption scheme with the recent constructions [Tsa22, VWW22] based on evasive LWE, we realize for the first time flexible (and distributed) broadcast encryption and registered ABE for general policies from lattice assumptions. Previously, these were only known from $i\mathcal{O}$. As we elaborate in Section 1.1, our approach to the above problem is to start from an approach that relies on obfuscation and SSB hash functions, and then roughly speaking, replace $i\mathcal{O}$ with plain witness encryption and the SSB hash function with a function-binding hash function. In some sense, we are able to substitute the strong notion of $i\mathcal{O}$ with the comparatively weaker notion of witness encryption by using the more expressive notion of function-binding hash functions in place of an SSB hash function. Broadly speaking, we are optimistic that the techniques we develop will allow us to use witness encryption in place of $i\mathcal{O}$ in other settings, and thus, bring us closer to simpler and more practical realizations of many advanced cryptographic primitives.

Why witness encryption? Since witness encryption is often regarded as an "obfustopia" primitive, a natural question one might ask is whether it is interesting to study cryptographic constructions from plain witness encryption. We believe the answer is yes. As noted previously, recent advances have introduced new routes [BJK+18, BIOW20] and constructions [CVW18, Tsa22, VWW22] for building (plain) witness encryption in ways that do not seem sufficient for indistinguishability obfuscation (nor stronger forms of witness encryption). Moreover, these constructions are simpler and far more efficient than all existing constructions of indistinguishability obfuscation. Thus, witness encryption

can be an easier primitive to build, and provide a more efficient path towards realizing advanced cryptographic notions that are currently only known from obfuscation. In more detail, [BJK+18] show how to construct a non-trivial exponentially-efficient witness encryption assuming sub-exponential hardness of the standard learning with errors (LWE) assumption, and [BIOW20] show how to construct witness encryption in the generic (pairing-free) group model under the hypothesis that approximating the minimal distance in linear codes is NP-hard (for super-logarithmic approximation factors). Recent works [Tsa22, VWW22] construct witness encryption under a new "evasive LWE" assumption, a new knowledge assumption that conjectures hardness of LWE even given some trapdoor information. Moreover, at least with respect to black-box separations [GMM17], witness encryption is a demonstrably weaker notion than indistinguishability obfuscation.

1.1 Technical Overview

In this section, we provide a general overview of our notion of function-binding hash functions and how to combine it with witness encryption to realize new cryptographic applications. To demonstrate our approach, we first introduce our notion of flexible broadcast encryption (which generalizes the notion of distributed broadcast encryption from [BZ14]) and describe a simple construction of flexible broadcast encryption from indistinguishability obfuscation and somewhere statistically binding hash functions. We then show how we instantiate an analogous template using witness encryption and function-binding hash functions.

Flexible broadcast encryption. In a traditional broadcast encryption scheme [FN93], each user is associated with an index $i \in [n]$ and a broadcaster can encrypt a message to a set of recipients $S \subseteq [n]$. Any user in the set S can use their private key to decrypt the encrypted broadcast, while users outside the set S should learn no information about the message (even if they combine their keys together). The efficiency requirement is that the size of the ciphertext should be sublinear in the size of S (ideally, polylogarithmic in $|S|$). Note that we assume that the decrypter knows the set S. In traditional broadcast encryption, a central authority is needed to issue the decryption keys to users. The central authority is fully trusted, and if compromised, their long-term secret key can be used to decrypt all ciphertexts. Broadcast encryption is a well-studied primitive and construction are known from a broad range of cryptographic assumptions [FN93, NNL01, HS02, DF02, BGW05, BW06, GW09, BW13, BZ14, BWZ14, CGW15, GVW19, AY20, AWY20, Wee21, BV22, Wee22].

Distributed broadcast encryption [BZ14] provides an elegant approach to remove the trusted key issuer in the setting of broadcast encryption. Here, instead of a central authority issuing keys, users generate their own key. Like broadcast encryption, each user in a distributed broadcast encryption is still associated with an index $i \in [n]$, and the implicit assumption is that indices for different users are unique. The encrypter in this case can encrypt a message by specifying a set $S \subseteq [n]$ of user indices. As before, the size of the broadcast ciphertext should be sublinear, or preferably, polylogarithmic, in the number of users in the broadcast set. Anyone in the broadcast set can decrypt using their own secret key. Analogous to broadcast encryption where we assume the decrypter knows the set of users associated with an encrypted broadcast, we assume the decrypter in the distributed broadcast scheme knows the set of public keys associated with the broadcast ciphertext. Unlike the case with traditional broadcast encryption, distributed broadcast is currently only known from $i\mathcal{O}$ [BZ14].

Flexible broadcast encryption is a further generalization of distributed broadcast encryption where we remove the index-dependence for users. Here, users simply generate their public key and post it to the public bulletin board (e.g., a public-key directory). There is no notion of a user "index" and indeed users can generate their keys independently and without any coordination.[1] We do assume that there is a global set of public parameters for the scheme (that users use when generating keys).

An obfuscation-based approach for flexible broadcast encryption. We start by describing a simple approach to construct a flexible broadcast encryption from indistinguishability obfuscation and somewhere statistically binding (SSB) hash functions [HW15]. The approach we take here is an adaptation of the registered ABE scheme

[1]Note that we can generically obtain a flexible broadcast encryption scheme from a distributed broadcast encryption scheme by having users sample their index randomly. To support a maximum of ℓ users, we would instantiate the scheme with $n = \Omega(\ell^2)$ indices. This is sufficient if the number of users is *a priori* bounded, though it may incur a *quadratic* blowup in the size of some scheme parameters. If the underlying distributed broadcast encryption scheme supports a super-polynomial number of users (i.e., $n = \lambda^{\omega(1)}$), then it directly implies a flexible broadcast encryption for an arbitrary polynomial number of users.

from [HLWW23] (rather than the Boneh-Zhandry approach [BZ14] based on a reduction to key-agreement).

- The public parameters $\mathsf{pp} = (\mathsf{pk}, \mathsf{hk})$ for the scheme contains a public key pk for a standard public-key encryption scheme and a hash key hk for an SSB hash function.
- To join the system, a user samples randomness r and computes an encryption of 1 under pk with randomness r: $c = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$. The user's public key is the ciphertext c and the decryption key is the randomness r.
- To encrypt a message m to an (ordered) set of public keys $(c_1, \ldots, c_n)$, the encrypter now proceeds as follows:
  - The encrypter starts by hashing the public keys using the SSB hash function to obtain a digest $\mathsf{dig} = H(\mathsf{hk}, (c_1, \ldots, c_n))$. Recall that in an SSB hash function, one can give a succinct proof (of size $\mathsf{poly}(\lambda, \log n)$ where λ is a security parameter) that the value at index i is c_i (with respect to $(\mathsf{hk}, \mathsf{dig})$).
  - The encrypter now prepares a program $\mathcal{P}$ that has message m, the public parameters $\mathsf{pp} = (\mathsf{hk}, \mathsf{pk})$, and the digest dig hard-wired inside. The program takes as input an index $i \in [n]$, a public key c, an opening π, and a secret key r and checks the following:
    - **Inclusion in broadcast set:** The opening π is a valid opening for c at index i with respect to $(\mathsf{hk}, \mathsf{dig})$.
    - **Knowledge of secret key:** The pair (c, r) is a valid public/secret key-pair: $c = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$.

    If all of these properties are satisfied, then the program outputs the message m. Otherwise, it outputs $\perp$.
  - The broadcast ciphertext is an obfuscation of program $\mathcal{P}$: $\mathsf{ct} \leftarrow i\mathcal{O}(\mathcal{P})$.
- During decryption, the user know both the ciphertext $\mathsf{ct} = i\mathcal{O}(\mathcal{P})$ and the set of public keys $(c_1, \ldots, c_n)$ associated with ct.[2] Suppose the user's public key is $c_i \in (c_1, \ldots, c_n)$ and let r_i be their secret key. The user starts by computing $\mathsf{dig} = H(\mathsf{hk}, (c_1, \ldots, c_n))$ together with an opening π for (i, c_i) with respect to $(\mathsf{hk}, \mathsf{dig})$. It runs the obfuscated program ct on input (i, c_i, π, r_i).

Correctness of the scheme follows as long as the obfuscation scheme is functionality-preserving. Security follows via a simple hybrid argument:

- In the normal flexible broadcast encryption security game, the challenge ciphertext is an encryption of m to a set $S = (c_1, \ldots, c_n)$, where $c_i = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$. Since the adversary cannot be in the broadcast set in the security game, each public key c_i is an *honestly-generated* encryption of 1: $c_i = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r_i)$. We appeal to semantic security of the public-key encryption scheme to replace each c_i with an encryption of 0 (i.e., $c_i = \mathsf{PKE.Enc}(\mathsf{pk}, 0; r_i)$).
- Next, we replace the hash key hk with one that is statistically binding at position 1. This is computationally indistinguishable from the normal hash key if the SSB scheme satisfies index hiding. Now $\mathsf{dig} = H(\mathsf{hk}, (c_1, \ldots, c_n))$ and c_1 is now an encryption of 0. We conclude that every input (i, c, π, r) where $i = 1$ causes the program $\mathcal{P}$ to output $\perp$. This is because the only valid opening for index $i = 1$ is $c = c_1$ (since the hash function is *statistically* binding at index 1). However, c_1 is an encryption of 0, so it cannot be the case that $c_1 = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$ by (perfect) correctness of PKE. Then, by $i\mathcal{O}$ security, we can replace $\mathcal{P}$ with an obfuscation of the program $\mathcal{P}_1$ that outputs $\perp$ whenever $i = 1$.
- More generally, we can let $\mathcal{P}_t$ be the program $\mathcal{P}$ that always outputs $\perp$ whenever the input index satisfies $i \leq t$. By the above argument, if we sample hk to be statistically binding at index t, then the obfuscations of programs $\mathcal{P}_{t-1}$ and $\mathcal{P}_t$ are computationally indistinguishable.
- Finally, the program $\mathcal{P}_n$ outputs $\perp$ on all indices $i \leq n$. This program outputs $\perp$ on *all* inputs, so an obfuscation of $\mathcal{P}_n$ is computationally indistinguishable from an obfuscation of the program that outputs $\perp$ on *all* inputs.

[2] Note that set of public keys could just be a set of indices if we include a mapping between indices and user public keys (i.e., a public-key directory) as part of the public parameters of the flexible broadcast encryption scheme. This is the setting considered in [BZ14]. More generally, the set of users could admit a succinct description (e.g., "all computer science students") and the decrypter would look up the public keys associated with the members of the set.

In the final hybrid where the ciphertext is an obfuscated program that always outputs $\perp$, the adversary's view is *independent* of the message m, and semantic security holds.

Using witness encryption instead of $i\mathcal{O}$. Suppose we try to implement the above strategy using witness encryption in place of indistinguishability obfuscation. Observe that the program $\mathcal{P}$ defined above is a program that is checking membership in an NP language and outputting m if the input is a valid witness. Specifically, we can view the hard-wired components $(\mathsf{hk}, \mathsf{pk}, \mathsf{dig})$ as the statement and the input (i, c, π, r) as the witness. The associated NP relation then checks the inclusion-in-broadcast-set and knowledge-of-secret-key properties defined above. Thus, we can derive the same functionality as above by replacing the obfuscation of $\mathcal{P}$ with a witness encryption of m with respect to the statement $(\mathsf{hk}, \mathsf{pk}, \mathsf{dig})$. Correctness now follows exactly as before.

The challenge is in the security proof. Suppose we try a similar strategy as above where we first replace the public keys $(c_1, \ldots, c_n)$ with encryptions of 0. Let $\mathsf{dig} = H(\mathsf{hk}, (c_1, \ldots, c_n))$ as before. To leverage security of the witness encryption scheme and argue that the message is computationally hidden, we need to show that there does not exist a witness (i, c, π, r) for the statement $(\mathsf{hk}, \mathsf{pk}, \mathsf{dig})$. Certainly, any c where $c \in (c_1, \ldots, c_n)$ cannot be part of a valid witness. However, we cannot rule out inputs where $c \notin (c_1, \ldots, c_n)$ because it is possible for there to *exist* some $c = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$ and opening π such that π is a valid opening for (i, c) with respect to $(\mathsf{hk}, \mathsf{dig})$. While finding such an opening is computationally hard (by SSB security), such openings can certainly exist for some *index* $i \in [n]$. The crux of the issue is that SSB hash functions can only bind to a *single* component of the input, whereas to appeal to security of witness encryption, we need to argue that the digest dig rules out invalid openings to *all* indices.

Using $i\mathcal{O}$, the ability to statistically bind to a single index was sufficient to argue security because we are able to "save our progress" within the obfuscated program. Namely, we can iteratively rule out inputs (i, c, π, r) where $i \le t$, and then just focus on ruling out inputs where $i \le t + 1$. In this step, we only have to consider inputs of the form $(t+1, c, \pi, r)$ which we can handle by programming the SSB hash function to statistically bind on index $t+1$. A similar step-by-step approach does not work if we use witness encryption. While we can certainly embed the threshold t into the NP relation (i.e., only accept witnesses (i, c, π, r) where $i > t$), witness encryption does not provide any hiding property for t in this case. Indeed, the adversary can distinguish between a witness encryption ciphertext encrypted with respect to threshold t and one encrypted with respect to threshold $t + 1$. In the case of $i\mathcal{O}$, the thresholds themselves are hidden. In fact, this limitation motivated the formulation of a stronger version of "positional witness encryption" [GLW14, GVW19] which augments witness encryption with an additional "index hiding" property that allows one to secretly embed a threshold within a witness encryption ciphertext. Using positional witness encryption, it is possible to carry out the proof strategy defined above. Unfortunately, positional witness encryption appears to be a much stronger notion than plain witness encryption and the only known instantiations are from $i\mathcal{O}$.

Function-binding hash functions. The key barrier to instantiating the above framework is that SSB hash functions only bind statistically to a *single* input, but to rule out the *existence* of *any* valid tuple (i, c, π, r), we would seemingly need to statistically bind on all inputs simultaneously. The latter is clearly impossible if we require that the digest be compressing. We take a different approach. Instead of statistically binding to part of the *input* to the hash function, we instead statistically bind to the *output* of a function of the inputs. Namely, let $f\colon (\{0,1\}^\ell)^n \to \{0,1\}^t$ be a function. We say that a keyed function H with hash key hk is statistically function-binding for f if for all inputs $x_1, \ldots, x_n \in \{0,1\}^\ell$ and computing $\mathsf{dig} = H(\mathsf{hk}, (x_1, \ldots, x_n))$, there does not exist an opening π for $(j, \hat{x}_j)$ with respect to $(\mathsf{hk}, \mathsf{dig})$ whenever

$$\forall (x'_1, \ldots x'_{j-1}, x'_{j+1}, \ldots, x'_n) : f(x'_1, \ldots, x'_{j-1}, \hat{x}_j, x'_{j+1}, \ldots, x'_n) \neq f(x_1, \ldots, x_n).$$

In other words, the only possible openings at *any* index j are to values $\hat{x}_j$ where there exists an assignment to the remaining variables that are consistent with an evaluation of $f(x_1, \ldots, x_n)$. As in the case of SSB hash functions, we require succinct local openings to any index. We describe two cases of interest:

- **Index functions:** As a special case, an SSB hash function is a function-binding hash function for the index function $\mathsf{ind}_i\colon (\{0,1\}^\ell)^n \to \{0,1\}^\ell$ where $\mathsf{ind}_i(x_1, \ldots, x_n) \coloneqq x_i$.
- **Disjunction of predicates:** Let $g\colon \{0,1\}^\ell \to \{0,1\}$ be a binary-valued predicate and let $f_g\colon (\{0,1\}^\ell)^n \to \{0,1\}$ be the block-wise disjunction function

$$f_g(x_1, \ldots, x_n) \coloneqq \bigvee_{i \in [n]} g(x_i). \tag{1.1}$$

Suppose a hash key hk is statistically binding for the function f_g, and suppose $\mathsf{dig} = H(\mathsf{hk}, (x_1, \ldots, x_n))$ for an input $(x_1, \ldots, x_n)$ where $g(x_i) = 0$ for all $i \in [n]$. Function-binding security now says that there does *not* exist any opening $(j, \hat{x}_j)$ for any $j \in [n]$ and $\hat{x}_j \in \{0,1\}^\ell$ where $g(\hat{x}_j) = 1$.

Like index hiding in the context of SSB hash functions, we additionally require that the hash key hk associated with a predicate f computationally hides the function f. Similarly, we require that given a digest $\mathsf{dig} = H(\mathsf{hk}, (x_1, \ldots, x_n))$, one can efficiently compute a succinct local opening π to any index (j, x_j), where $|\pi| = \mathsf{poly}(\lambda, \ell, \log n)$, where ℓ is the output length of f. We provide the formal definition in Section 3.

Using function-binding hash functions. Consider again our template for constructing flexible broadcast encryption, except we replace the SSB hash function in the construction with a function-binding hash function. Our security reduction now proceeds as follows:

- As before, we start by switching the honest users' public keys $c_1, \ldots, c_n \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 0)$ to encryptions of 0.
- Next, we sample the hash key hk to be statistically function-binding for the function $f_g(c_1, \ldots, c_n)$ where f_g is the disjunction-of-predicates function from Eq. (1.1) and $g(c) = \mathsf{PKE.Dec}(\mathsf{sk}, c)$, and sk is the decryption key associated with pk in the public-key encryption scheme. Observe now that in the challenge ciphertext, $f_g(c_1, \ldots, c_n) = 0$ since each c_i is an encryption of 0.
- Security now follows by security of the witness encryption scheme. We claim that with overwhelming probability over the choice of the hash key hk, the statement $(\mathsf{hk}, \mathsf{pk}, \mathsf{dig})$ is false. Take any candidate witness (i, c, π, r). For this to be a valid witness, it must be the case that $c = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$ and π is a valid opening for (i, c) with respect to $(\mathsf{hk}, \mathsf{dig})$ where $\mathsf{dig} = H(\mathsf{hk}, (c_1, \ldots, c_n))$. Since c is an encryption of 1, $g(c) = 1$. Since f_g is a disjunction, the output of f_g on any tuple of inputs that includes c is necessarily $1 \neq 0 = f_g(c_1, \ldots, c_n)$. Function-binding security now says that there does not exist any valid π for (i, c) with respect to $(\mathsf{hk}, \mathsf{dig})$.

The use of the function-binding hash function eliminates the need for a step-by-step hybrid strategy. As such, we can rely directly on *plain* witness encryption to complete the analysis. This yields the first flexible (and correspondingly, distributed) broadcast encryption scheme that does not rely on $i\mathcal{O}$. Like the Boneh-Zhandry distributed broadcast encryption scheme from $i\mathcal{O}$, the size of the ciphertexts in our scheme scale with $\mathsf{poly}(\lambda, \log n)$, where n is the number of users in the broadcast set. We provide the full construction and analysis in Section 4.

Optimal broadcast encryption. A flexible broadcast encryption scheme immediately implies a traditional broadcast encryption scheme with a central key issuer. Here, the central key issuer would generate the public/private keys for all of the users and publish the public keys for each user as the master public key for the broadcast encryption system. Instantiated with our flexible broadcast encryption scheme, this yields a construction with a linear-size public key, but short ciphertexts and secret keys (scaling polylogarithmically with the number of users). It is straightforward to extend our scheme to an *optimal* broadcast encryption where the size of the public key also scales polylogarithmically with the number of users by working in the random oracle model.

The reason our basic approach has a linear-size public key is because we concatenate n user keys $c_1, \ldots, c_n$ to form the master public key for the broadcast encryption scheme. The key idea behind our optimal broadcast encryption scheme is to instead derive each user's public key from a hash of the user index i: that is, we compute $c_i \leftarrow h(i)$, where h is modeled as a random oracle. On the one hand, the master public key only needs to contain the public parameters for our flexible broadcast encryption system, which is now independent of the number of users in the scheme. On the flip side, we need a way to generate decryption keys for the users.

To recover functionality, we start with a more general view of our scheme. Recall that in our scheme, an honestly-generated public key c_i is an encryption of 1 and the associated secret key is a proof that c_i is an encryption of 1. In this case, the proof is the encryption randomness. In the security analysis, each c_i is replaced with an encryption of 0. In this case, there no longer exists a proof (i.e., encryption randomness) that asserts c_i to be an encryption of 1. We can more abstractly view this as implementing a set system where the honestly-generated public keys c_i are elements of some set S (i.e., encryptions of 1 in our case), and the associated secret key is a (statistically sound) proof of membership that $c_i \in S$. In the security proof, the public keys are sampled from some set T that is disjoint from S (i.e., encryptions of 0 in our case). Finally, we also require that a random element in S should be computationally indistinguishable from a random element in T so we can change the distribution of the honest users' keys in the

security proof. Now, suppose we can construct sets $S, T \subseteq \{0,1\}^k$ where S is dense (i.e., a random element in $\{0,1\}^k$ is contained in S with all but negligible probability), and moreover, given a trapdoor, it is possible to construct a proof of membership for any element $c \in S$. We refer to this notion as a "trapdoor proof system." Trapdoor proof systems can be constructed in a straightforward manner from a non-interactive zero-knowledge proof in conjunction with a public-key encryption scheme with pseudorandom ciphertexts (see Section 5.1). Using a trapdoor proof system, we can now adapt our flexible broadcast encryption scheme to obtain an optimal broadcast encryption scheme in the random oracle model:

- The setup algorithm for the broadcast encryption scheme samples a trapdoor td for our trapdoor proof system. Let $S, T \subseteq \{0,1\}^k$ be the sets associated with the trapdoor proof system. Let $h\colon [n] \to \{0,1\}^k$ be a hash function that is modeled as a random oracle.
- The public key for user i is $c_i \leftarrow h(i)$. With overwhelming probability, $c_i \in S$. The secret key r_i is a proof of membership that $c_i \in S$, which can be sampled using the trapdoor td for the trapdoor proof system.
- Encryption and decryption proceed exactly as in the flexible broadcast encryption scheme.

The security analysis also proceeds analogously to that for our flexible broadcast encryption scheme. Namely, we first move the honest users' public keys from S to T (so no proofs of membership exist), and then appeal to statistical function-binding and security of witness encryption. We refer to Section 5 for the full details.

Registered attribute-based encryption. In Section 6, we show that function-binding hash functions and plain witness encryption can also be used to obtain a registered ABE scheme that supports an arbitrary number of users and supporting general policies. Previously, this was also only known from indistinguishability obfuscation [HLWW23]. Non-obfuscation-based constructions of registered ABE could only support a bounded number of users [HLWW23] or support less-expressive policies such as Boolean formulas [HLWW23] or equality predicates [GHMR18, GHM+19, GV20, CES21, GKMR23]. We sketch the basic idea here (which closely parallels the $i\mathcal{O}$-based approach from [HLWW23]):

- The common reference string for the registered ABE scheme consists of a public key pk for a public-key encryption scheme and the hash key for a function-binding hash function.
- As in our flexible broadcast encryption scheme, to generate a public key, a user samples randomness r and computes their public key as $c = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$. Their secret key is the encryption randomness r.
- Suppose n users register with public keys $(c_1, \ldots, c_n)$ and attributes $(x_1, \ldots, x_n)$. The aggregated public key is a hash of the public keys and associated attributes: $\mathsf{mpk} = \mathsf{dig} = H(\mathsf{hk}, ((c_1, x_1), \ldots, (c_n, x_n)))$. The helper decryption key for user i is an opening π_i for (c_i, x_i) with respect to $(\mathsf{hk}, \mathsf{dig})$.
- An encryption of message m under a policy $\mathcal{P}$ consists of a witness encryption ciphertext for m with respect to the statement $(\mathsf{pk}, \mathsf{hk}, \mathsf{dig}, \mathcal{P})$. A witness (i, c, x, π, r) is valid if the following holds:
  - **Valid opening:** The opening π is a valid opening for (c, x) at index i with respect to $(\mathsf{hk}, \mathsf{dig})$.
  - **Policy satisfiability:** The associated attribute satisfies the policy: $\mathcal{P}(x) = 1$.
  - **Knowledge of secret key:** The pair (c, r) is a valid public/secret key-pair: $c = \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$.

We show that this scheme is secure in a *selective* model where the adversary commits to its challenge policy $\mathcal{P}$ at the beginning of the security game. The security proof follows a similar hybrid structure as the security proof for our flexible broadcast encryption scheme:

- We first replace the honest users' public keys with encryptions of 0.
- Next, we sample the hash key to be statistically function-binding for the function $f_g((c_1, x_1), \ldots, (c_n, x_n))$ where f_g is the disjunction-of-predicates function from Eq. (1.1) and $g(c, x) := \mathsf{PKE.Dec}(\mathsf{sk}, c) \wedge \mathcal{P}(x)$. Since we need to program the challenge policy $\mathcal{P}$ into the hash key, we are only able to argue selective security.

- Security now follows by security of the witness encryption scheme. Specifically, with overwhelming probability over the choice of the hash key, the statement $(\mathsf{hk}, \mathsf{pk}, \mathsf{dig})$ is false. This is because $f_g((c_1, x_1), \ldots, (c_n, x_n)) = 0$ by construction[3] whereas a valid witness (i, c, x, π, r) must satisfy $g(c, x) = 1$.

Constructing function-binding hash functions. To complete the loop, we finally show how to construct function-binding hash functions. In this work, we describe a construction that supports the disjunction-of-blocks family of functions f_g (for arbitrary g) from Eq. (1.1). This is the function family used in our applications to flexible broadcast encryption and registered ABE. Our construction proceeds very similarly to the construction of SSB hash functions from fully homomorphic encryption [HW15].

Specifically, let $(x_1, \ldots, x_n)$ be the input to our function-binding hash function; for ease of exposition, suppose that $n = 2^d$ is a power of two. We combine a Merkle tree [Mer89] with a (leveled) homomorphic encryption scheme. We construct a complete binary tree of depth d, where the leaves of the binary tree are associated with the labels $x_1, \ldots, x_n$. At the base level (the leaves), we homomorphically evaluate the function g on the input. Note that the description of g itself is encrypted as part of the hash key, since we require the hash key to hide the function. The value of each internal node is then obtained by homomorphically evaluated the OR function on its child ciphertexts. Observe that at the end of this process, the root of the tree will be an encryption of $f_g(x_1, \ldots, x_n)$. The function-binding property now follows from (perfect) correctness of the encryption scheme. We provide the full details and proof in Section 3.1.

From selective to adaptive security in the random oracle model. The aforementioned constructions of flexible/optimal broadcast encryption and registered ABE all satisfy a selective notion of security where the adversary has to pre-commit to the set of corrupted users in the security game. We do note that our flexible/optimal broadcast encryption schemes, however, satisfy a stronger notion of "semi-static security" [GW09] where the adversary only needs to commit to a *super-set* of its challenge set (but is not allowed to corrupt any users in its committed set). Previously, Gentry and Waters [GW09] showed that a broadcast encryption scheme with semi-static security implies an adaptively-secure scheme in the random oracle model. Using the techniques of [GW09], we show in Appendix B how to transform the flexible broadcast encryption scheme above into one that satisfies adaptive security in the random oracle model. Similarly, our registered ABE construction described above satisfies a weaker policy-selective security without corruption queries. Namely, in the security game, the adversary has to declare its challenge policy upfront and it is not allowed to corrupt honestly-generated keys (but it can still generate its own keys and register those). We show in Appendix C that in the random oracle model, we can apply a similar approach based on [GW09] to support corruption queries in our registered ABE scheme.[4] We note that the $i\mathcal{O}$-based construction of registered ABE of [HLWW23] achieves full adaptive security. We leave it as an open question to satisfy this stronger notion from plain witness encryption.

Flexible broadcast encryption from registered ABE. As a final contribution, we also show that a registered ABE scheme *generically* implies a flexible broadcast encryption scheme.[5] The idea is simple: the public parameters for the flexible broadcast encryption scheme is the CRS for the registered ABE scheme. Now, to broadcast to a set of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, the encrypter initializes a registered ABE scheme and registers $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ to a dummy attribute x to obtain a master public key mpk. It then encrypts the message m with respect to the master public key and a dummy policy P where $P(x) = 1$. To decrypt, a user would run the registration algorithm itself, derive the associated helper decryption key, and combine with their own secret key to recover the message m. We describe this transformation in Section 7.

By applying the generic transformation to our registered ABE scheme, we obtain another approach for constructing flexible broadcast encryption from witness encryption and function-binding hash functions. We do note though that the flexible broadcast encryption scheme obtained via this generic approach satisfies a *weaker* notion of static security

[3] Specifically, if c_i is the public key of an honest user, then $\mathsf{PKE.Dec}(\mathsf{sk}, c_i) = 0$ whereas if x_i belongs to a corrupted user, then $\mathcal{P}(x_i) = 0$ by the admissibility restriction on the registered ABE adversary. In either case, $g(c_i, x_i) = 0$ for all $i \in [n]$.

[4] The transformed scheme is still policy-selective; however, this can be removed generically via complexity leveraging and assuming subexponential hardness. Note that one cannot use complexity leveraging to handle corruption queries without having the ciphertext size grow with the number of registered users.

[5] Note that the direct approach of encoding the broadcast set as part of the policy in the ABE scheme does *not* yield a flexible broadcast encryption scheme with the required efficiency. Namely, in existing registered ABE schemes (including the one in this work), the size of the ciphertext scales with the size of the policy. Using this to implement broadcast encryption yields a flexible broadcast encryption scheme where the size of the ciphertext scales *linearly* with the number of users.

(as opposed to semi-static security). Thus, the generic transformation does not subsume our direct construction of flexible broadcast encryption which satisfies semi-static security. As noted above, a semi-statically secure scheme can be bootstrapped to an adaptively-secure one in the random oracle model, but a similar transformation is not known if we start from a statically-secure scheme.

The transformation described here critically assumes that the key-generation process is "stateless:" namely, users can independently generate their keys without knowledge of the current state of the key curator. In schemes like the pairing-based construction of [HLWW23], the registration process is not stateless and users need to know the current number of registered users when registering. In this setting, our transformation can still be applied to obtain a *distributed* broadcast encryption scheme from assumptions over pairing groups. We provide additional details in Remark 7.6. Ultimately, in conjunction with [HLWW23], we also obtain the first distributed broadcast encryption scheme from pairing-based assumptions (over a composite-order group). The resulting scheme supports an *a priori bounded* number of users, requires a quadratic-size CRS (inherited from [HLWW23]), and satisfies static security.

1.2 Concurrent Work of Brakerski et al. [BBK+23]

In an independent and concurrent work, Brakerski et al. [BBK+23] constructed succinct non-interactive arguments (SNARGs) for monotone policy batch NP. A key technical ingredient introduced in their work is the notion of a predicate-extractable hash family. Their notion shares some similarity to our notion of function-binding hash functions. In both primitives, the hash key is associated with some function $f \in \mathcal{F}$ from a family $\mathcal{F}$, and the evaluation of the hash function on some input x will in some way "bind" to $f(x)$. Namely, the hash value reflects some global property of the hashed data. Moreover, the function f associated with the hash key should be computationally hidden. However, there are some important distinctions between the notions that we highlight below.

The work of [BBK+23] presents two sets of definitions. The paper first provides a broad definition of a predicate-extracatable hash family for a general function family $\mathcal{F}$ (c.f., [BBK+23, Definition 5.1]). Next, they consider a specialized definition tailored for bit-fixing predicates (i.e., a bit-fixing predicate takes as input a bitstring and compares the values on a fixed set of indices to a particular target value y, and outputs 1 if they match and 0 otherwise) [BBK+23, Definition 5.2]. In addition to restricting the functionality, the predicate-extractable hash for bit-fixing predicates definition introduces additional security properties beyond those inherited from the main definition for general functions. The predicate-extractable hash function for bit-fixing predicates is used to obtain SNARGs for monotone policy batch NP.[6]

Our notion of function-binding hash functions differs from the general notion of predicate-extractable hash functions from [BBK+23, Definition 5.1] in two important ways:

- **Statistical vs. computational binding.** First, our binding property is *statistical* in that even a computationally unbounded adversary cannot construct a local opening for an "incompatible" input (i.e., an input that cannot be extended to one that is consistent with the function output stored in the hash). In contrast, the binding property in [BBK+23, Definition 5.1] is computational. (This also means that the notion of somewhere statistically binding hash functions [HW15] is not captured by their security definition.) This statistical binding property is critical for realizing our applications from plain witness encryption, but not necessarily needed for constructing proof systems. Note that while predicate-extractable hash functions require an explicit extraction algorithm, this property is defined independently of the verification algorithm. Thus, the combination of computational binding and extraction correctness does not imply a statistical binding property on the hash function.

- **Function binding.** Second, the (statistical) binding property in our definition is tightly coupled to the function f associated with the hash key chosen at setup. In contrast, the computational binding property from [BBK+23] is defined independently of the function f associated with the hash key and only requires that it be computationally difficult to open an index to both a 1 and 0. The more specialized definition of predicate-extractable hash for bit-fixing predicates adds an additional property they call "consistency of extraction" which does couple the security of the verify algorithm to the global hash property. This additional consistency property of the hash function is important for realizing their SNARG construction.

[6] There is also a third definition of predicate-extractable hash functions with tags for bit-fixing predicates. For simplicity, our discussion is limited to the first two.

The two notions also have some additional minor differences. Unlike [BBK+23], our definition does not require an algorithm to extract the function value associated with the hashed input. In addition, for certain function classes under our definition, it may be possible to open an input bit to both 0 and 1. As a trivial example, consider a function family $\mathcal{F}$ for which all functions $f \in \mathcal{F}$ ignore a certain bit. Finally, [BBK+23] consider function families as predicates with one-bit output, whereas we consider multi-bit outputs. For certain security definitions it might be possible to go from supporting one-bit output to supporting ℓ-bit outputs by concatenating ℓ hash values. This is conceptually similar to the observation that one can obtain the indistinguishability notion of functional encryption [BSW11] from earlier notions of predicate encryption [BW07, KSW08].

2 Preliminaries

In this section, we introduce the notation and definitions of standard cryptographic primitives that we use in this paper. We let $\mathbb{N} = \{1, 2, 3, \ldots\}$ denote the set of natural numbers, and for any $n \in \mathbb{N}$, we write $[n]$ to denote the set $[n] = \{1, \ldots, n\}$. For integers $a, b \in \mathbb{Z}$ with $a \leq b$, we write $[a, b]$ to denote the set $\{a, a+1, \ldots, b\}$. For a set Σ, referred to as the *alphabet*, we denote Σ^* the set of strings consisting of 0 or more elements from Σ. We let Σ^n denote the set of n-character strings from Σ and $\Sigma^{\leq n}$ the set of string of length at most n. For a string $s \in \Sigma^*$, we let $|s|$ denote the length of s. Unless specified otherwise, we assume that a string s is defined over the binary alphabet $\{0, 1\}$.

We write $\lambda \in \mathbb{N}$ to denote the security parameter. We say that a function $p \colon \mathbb{N} \to \mathbb{N}$ is in the set $\mathsf{poly}(\lambda)$ and is *polynomially-bounded* if there exists a constant c and an index $i \in \mathbb{N}$ such that $p(\lambda) \leq \lambda^c$ for all $\lambda \geq i$. We say that a function $\mathsf{negl} \colon \mathbb{N} \to \mathbb{R}$ is negligible if for every constant $c > 0$ there exists $i \in \mathbb{N}$ such that $\mathsf{negl}(\lambda) \leq \lambda^{-c}$ for all $\lambda \geq i$.

We use PPT to denote the acronym *probabilistic polynomial time*. We say that an algorithm is *stateful* if when invoked multiple times in succession, it implicitly takes its entire view so far, including previous inputs and random coins used, as input. We say that a function $f \colon \mathbb{N} \to \mathbb{N}$ is efficiently computable if there exists a PPT algorithm that computes f on all inputs.

For a distribution X, we write $x \leftarrow X$ to denote the process of sampling a value x from the distribution X. For a set $\mathcal{X}$, we use $x \xleftarrow{R} \mathcal{X}$ to denote the process of sampling a value x from the uniform distribution over $\mathcal{X}$. We use $x = A(\cdot)$ to denote the output of a deterministic algorithm and $x \leftarrow A(\cdot)$ to denote the output of a randomized algorithm. We write $x \leftarrow A(y; r)$ to denote the output of running the randomized algorithm A on input y with *explicit* randomness r. We write $x \coloneqq y$ to denote the assignment of value y to x. For a distribution D, we define $\mathsf{Supp}(D)$ to denote the support of the distribution D.

Boolean circuits. A Boolean circuit consist of input and output wires (each labeled with a bit) and gates (either AND, OR, or NOT gates unless specified otherwise) that are arranged according to a directed acyclic graph. We define the size of a circuit to be the number of input/output wires and gates in the circuit. We assume a canonical description for all Boolean circuits of a given size. We define the circuit depth to be the depth of the graph representing the circuit.

Message spaces. For a function $m \colon \mathbb{N} \to \mathbb{N}$, we say that a sequence $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ is a *message space with message-length m* if for every $\mathsf{msg} \in \mathcal{M}_\lambda$, $|\mathsf{msg}| = m(\lambda)$. We reserve the special character $\perp$ to denote an invalid message.

Languages and witness relations. We consider sequences of languages $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda \in \mathbb{N}}$ that are parameterized by a security parameter λ. We say that a language has instance length $m = m(\lambda)$ if for all security parameters $\lambda \in \mathbb{N}$ and statements $x \in \mathcal{L}_\lambda$, we have that $|x| = m(\lambda)$. We say a language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda \in \mathbb{N}}$ is polynomially-bounded if there exists a polynomial p such that for all $\lambda \in \mathbb{N}$, $\max\{|x| : x \in \mathcal{L}_\lambda\} \leq p(\lambda)$. A witness relation for an NP language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda \in \mathbb{N}}$ with instances of length m is a deterministic, binary relation $\mathbf{R}_\mathcal{L}$ that can be computed in time $\mathsf{poly}(\lambda, m(\lambda))$ and characterizes $\mathcal{L}$ by $\mathcal{L}_\lambda = \{x : \exists w, (1^\lambda, x, w) \in \mathbf{R}_\mathcal{L}\}$. We say that a string $w \in \{0, 1\}^*$ is a *witness* for an instance $x \in \mathcal{L}_\lambda$ if $(1^\lambda, x, w) \in \mathbf{R}_\mathcal{L}$. We let $\mathbf{R}_\mathcal{L}(1^\lambda, x)$ denote the set of all witnesses for x: namely, $\mathbf{R}_\mathcal{L}(1^\lambda, x) = \{w : \mathbf{R}_\mathcal{L}(1^\lambda, x, w) = 1\}$. We say that $\mathcal{L}$ is efficiently samplable with ℓ uniform bits if there exists an efficient algorithm Sample such that

$$\{x \leftarrow \mathsf{Sample}(1^\lambda; r) : r \xleftarrow{R} \{0,1\}^\ell\} \equiv \{x \xleftarrow{R} \mathcal{L}_\lambda\}.$$

We say that $\mathcal{L}$ is efficiently-recognizable if there exists an efficient algorithm Decide that takes as input the security parameter λ and an instance $x \in \{0, 1\}^m$ and outputs whether $x \in \mathcal{L}_\lambda$.

2.1 Witness Encryption

Witness encryption [GGSW13] allows users to encrypt a message with respect to an instance x of an NP language $\mathcal{L}$. Anyone who has a corresponding witness w for x should be able to decrypt, whereas if x is false, the ciphertext should computationally hide the underlying message. We formalize this as follows.

Definition 2.1 (Witness Encryption)**.** Let $m \in \mathsf{poly}(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space of length m. A witness encryption scheme WE for an NP language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda\in\mathbb{N}}$ with instance length $n = n(\lambda)$ and witness relation $\mathbf{R}_\mathcal{L}$ consists of polynomial-time algorithms $(\mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Enc}(1^\lambda, \mathsf{msg}, x) \to \mathsf{ct}$: A probabilistic algorithm that on input a security parameter λ, a message $\mathsf{msg} \in \mathcal{M}_\lambda$, and an instance x for the language $\mathcal{L}$, outputs a ciphertext ct. We implicitly assume that ct includes 1^λ and x.
- $\mathsf{Dec}(\mathsf{ct}, w) \to \mathsf{msg}$: A deterministic algorithm that on input a ciphertext ct and a witness w, outputs a message $\mathsf{msg} \in \mathcal{M}_\lambda \cup \{\bot\}$.

We require that $(\mathsf{Enc}, \mathsf{Dec})$ satisfy the following properties:

- **(Perfect) correctness:** For all $\lambda \in \mathbb{N}$, messages $\mathsf{msg} \in \mathcal{M}_\lambda$, and tuples $(1^\lambda, x, w) \in \mathbf{R}_\mathcal{L}$, it holds that

$$\Pr\left[\mathsf{ct} \leftarrow \mathsf{Enc}(1^\lambda, \mathsf{msg}, x) : \mathsf{Dec}(\mathsf{ct}, w) = \mathsf{msg}\right] = 1.$$

- **Message indistinguishability:** For all stateful PPT algorithms A, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l}(x, \mathsf{msg}_0, \mathsf{msg}_1) \leftarrow A(1^\lambda) \\ b \xleftarrow{R} \{0,1\} \\ \mathsf{ct}^\star \leftarrow \mathsf{Enc}(1^\lambda, \mathsf{msg}_b, x) \\ b' \leftarrow A(\mathsf{ct}^\star)\end{array} : x \notin \mathcal{L}_\lambda \wedge b' = b\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

2.2 Public-Key Encryption

We recall the standard definition of semantically-secure public-key encryption.

Definition 2.2 (Public-Key Encryption)**.** Let $m \in \mathsf{poly}(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space of length m. A (semantically-secure) public-key encryption scheme PKE for message space $\mathcal{M}$ consists of polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: A probabilistic algorithm that on input a security parameter λ, outputs a public key pk and a secret key sk. We implicitly assume that pk includes 1^λ and sk includes pk.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg}) \to \mathsf{ct}$: A probabilistic algorithm that on input a public key pk and a message $\mathsf{msg} \in \mathcal{M}_\lambda$, outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{msg}$: A deterministic algorithm that on input a secret key sk and a ciphertext ct, outputs a message $\mathsf{msg} \in \mathcal{M}_\lambda \cup \{\bot\}$.

We require that $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfy the following properties:

- **(Perfect) correctness:** For all $\lambda \in \mathbb{N}$, messages $\mathsf{msg} \in \mathcal{M}_\lambda$, it holds that

$$\Pr\left[\begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})\end{array} : \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = \mathsf{msg}\right] = 1.$$

- **Semantic security:** For all stateful PPT algorithms A, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{ll} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) & \\ (\mathsf{msg}_0,\mathsf{msg}_1) \leftarrow A(\mathsf{pk}) & \\ b \xleftarrow{R} \{0,1\} & : \quad b' = b \\ \mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pk},\mathsf{msg}_b) & \\ b' \leftarrow A(\mathsf{ct}^\star) & \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

We say that the encryption scheme is a bit encryption scheme if $\mathcal{M}_\lambda = \{0,1\}$ for all $\lambda \in \mathbb{N}$.

Remark 2.3 (Randomness Complexity)**.** We assume for simplicity that the algorithms KeyGen, Enc in Definition 2.2 use at most λ bits of randomness. This is without loss of generality since we can always expand the randomness using a pseudorandom generator (PRG).

2.3 Leveled Homomorphic Encryption

A (public-key) leveled homomorphic encryption scheme [Gen09] enables homomorphic evaluation on encrypted messages. Specifically, given any ciphertext ct encrypting a message x and a circuit C, the evaluation algorithm outputs a ciphertext for the message $C(x)$. A leveled homomorphic encryption scheme can only support circuits C with a priori bounded depth (specified during key generation). We formalize this as follows:

Definition 2.4 (Leveled Homomorphic Encryption)**.** Let $m \in \mathsf{poly}(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space of length m. A (public-key) leveled homomorphic encryption scheme LHE for a message space $\mathcal{M}$ consists of polynomial-time algorithms $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, where $(\mathsf{Enc}, \mathsf{Dec})$ have syntax corresponding to a standard public-key encryption scheme, and $(\mathsf{KeyGen}, \mathsf{Eval})$ have the following syntax:

- $\mathsf{KeyGen}(1^\lambda, 1^L) \to (\mathsf{pk}, \mathsf{sk})$: A probabilistic algorithm that on input a security parameter λ and a depth bound L, outputs a public key pk and a secret key sk. We implicitly assume that pk includes 1^λ and 1^L and sk includes pk.
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) \to \mathsf{ct}'$: A deterministic algorithm that on input a public key pk, a Boolean circuit C, and a ciphertext ct, outputs another ciphertext ct′.

We require that $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is a correct public-key encryption scheme for all depth bounds $L \in \mathbb{N}$ given to KeyGen and satisfies semantic security for all polynomially-bounded depth bounds $L(\lambda)$ given to KeyGen. We additionally require the following properties:

- **Correct evaluation:** For any $\lambda \in \mathbb{N}$, $L \in \mathbb{N}$, $x \in \mathcal{M}_\lambda$, and Boolean circuits C of depth at most L, it holds that

$$\Pr\left[\begin{array}{ll} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^L) & \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x) & : \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}') = C(x) \\ \mathsf{ct}' = \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) & \end{array}\right] = 1.$$

- **Compactness:** There exists a polynomial p such that for any $\lambda \in \mathbb{N}$, $L \in \mathbb{N}$, public/private key-pair $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Supp}\left(\mathsf{KeyGen}(1^\lambda, 1^L)\right)$, message $x \in \mathcal{M}_\lambda$, $\mathsf{ct} \in \mathsf{Supp}\left(\mathsf{Enc}(\mathsf{pk}, x)\right)$, and Boolean circuit C of depth at most L, it holds that

$$|\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct})| \leq p(\lambda, L, |C(x)|).$$

We emphasize that the definition above stipulates perfect correctness, meaning that any honestly generated ciphertext (output from either encryption or the result of evaluation) corresponding to a message m will correctly decrypt to m. We use this property to guarantee statistical function binding in our construction of Section 3.

Polylogarithmic depth decryption. A long sequence of works (c.f., [BV11, BGV12, Bra12, GSW13]) have shown how to construct leveled homomorphic encryption schemes from the plain learning with errors (LWE) assumption. Moreover, in these constructions, the decryption circuit can be computed by a circuit of polylogarithmic depth, which we formally define as follows:

Definition 2.5 (Polylogarithmic Depth Decryption)**.** We say that a leveled homomorphic encryption scheme $\mathsf{LHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ supports *polylogarithmic depth decryption* if there exists a polynomial p such that for any $\lambda \in \mathbb{N}, L \in \mathbb{N}$, public/private key-pair $(\mathsf{pk}, \mathsf{sk}) \in \mathrm{Supp}\left(\mathsf{KeyGen}(1^\lambda, 1^L)\right)$, the depth of the decryption circuit computing $\mathsf{Dec}(\mathsf{sk}, \cdot)$ is at most $p(\log \lambda, \log L)$.

Theorem 2.6 (Leveled Homomorphic Encryption, [BV11, BGV12, Bra12, GSW13])**.** *Assuming the plain learning with errors (LWE) problem, there exists a leveled homomorphic encryption scheme* LHE *with polylogarithmic depth decryption.*

3 Function-Binding Hash Functions

In this section, we introduce the notion of a *function-binding* hash function. In the following, we consider the setting where the hash function satisfies *statistical function binding* and *computational function hiding*. This is the notion we rely on for our applications. However, we note that we can easily consider alternative notions where where function binding is computational or function hiding is statistical.

Definition 3.1 (Function-Binding Hash Function)**.** Let $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ be the block size, $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$ be the output size. Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be a class of functions where each $\mathcal{F}_\lambda$ is a collection of functions $f\colon (\{0,1\}^{\ell_{\mathsf{blk}}(\lambda)})^* \to \{0,1\}^{\ell_{\mathsf{out}}(\lambda)}$ implementable by a circuit of size at most $s(\lambda) \cdot \mathsf{poly}(k)$, where k is the number of input blocks to f. A *function-binding hash function* for $\mathcal{F}$ is a tuple of polynomial-time algorithms $(\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{ProveOpen}, \mathsf{VerOpen})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, n) \to \mathsf{hk}$: A probabilistic algorithm that on input a security parameter λ and a bound on the number of input blocks n (in *binary*), outputs a hash key hk. We implicitly assume that hk includes 1^λ and n.
- $\mathsf{SetupBinding}(1^\lambda, n, f) \to \mathsf{hk}$: A probabilistic algorithm that on input a security parameter λ, a bound on the number of input blocks n, and a function $f \in \mathcal{F}_\lambda$, outputs a hash key hk. We implicitly assume that hk includes 1^λ and n.
- $\mathsf{Hash}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k)) \to \mathsf{dig}$: A deterministic algorithm that on input a hash key hk and hash inputs $(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) \in (\{0,1\}^{\ell_{\mathsf{blk}}(\lambda)})^k$ for some $k \le n$, outputs a digest dig.
- $\mathsf{ProveOpen}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k), S) \to \pi$: A deterministic algorithm that on input a hash key hk, a sequence of hash inputs $(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) \in (\{0,1\}^{\ell_{\mathsf{blk}}(\lambda)})^k$ for any $k \le n$, and a subset of indices $S \subseteq [k]$, outputs a proof π.
- $\mathsf{VerOpen}(\mathsf{hk}, \mathsf{dig}, S, \{(i, \mathsf{hinp}_i)\}_{i \in S}, \pi) \to \{0,1\}$: A deterministic algorithm that on input a hash key hk, a digest dig, a subset $S \subseteq [k]$ for any $k \le n$, a set of hash inputs $\{(i, \mathsf{hinp}_i)\}_{i \in S}$ where each $\mathsf{hinp}_i \in \{0,1\}^{\ell_{\mathsf{blk}}(\lambda)}$, and a proof π, outputs a bit $b \in \{0,1\}$.

We require that $(\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{ProveOpen}, \mathsf{VerOpen})$ satisfy the following properties:

- **Efficiency:** There exist polynomials p_1, p_2, p_3 such that for all parameters $\lambda \in \mathbb{N}$, $k \in \mathbb{N}$, $n \in \mathbb{N}$ with $k \le n$, any function $f \in \mathcal{F}_\lambda$, any hash key $\mathsf{hk} \in \mathrm{Supp}\left(\mathsf{Setup}(1^\lambda, n)\right)$, any tuple of inputs $(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) \in (\{0,1\}^{\ell_{\mathsf{blk}}(\lambda)})^k$, any digest $\mathsf{dig} \in \mathrm{Supp}\left(\mathsf{Hash}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k))\right)$, any subset $S \subseteq [k]$, and any proof $\pi \in \mathrm{Supp}\left(\mathsf{ProveOpen}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k), S)\right)$, the following hold:
  - $\mathsf{Setup}(1^\lambda, n)$ and $\mathsf{SetupBinding}(1^\lambda, n, f)$ run in time $p_1(\lambda, s(\lambda), \log n)$;
  - $|\mathsf{dig}| \le p_2(\lambda, \ell_{\mathsf{out}}(\lambda), \log k)$; and
  - $|\pi| \le |S| \cdot p_3(\lambda, \ell_{\mathsf{out}}(\lambda), \log k)$.
- **Perfect completeness:** For all parameters $\lambda \in \mathbb{N}$, $k \in \mathbb{N}$, $n \in \mathbb{N}$ with $k \le n$, inputs $(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) \in (\{0,1\}^{\ell_{\mathsf{blk}}(\lambda)})^k$, and subsets $S \subseteq [k]$, it holds that

$$\Pr\left[\begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, n) \\ \mathsf{dig} = \mathsf{Hash}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k)) \\ \pi = \mathsf{ProveOpen}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k), S) \\ b = \mathsf{VerOpen}(\mathsf{hk}, \mathsf{dig}, S, \{(i, \mathsf{hinp}_i)\}_{i \in S}, \pi) \end{array} : b = 1\right] = 1.$$

- **Statistical function binding:** For all stateful unbounded adversaries A and efficiently-computable functions $n \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Expt}^{\mathsf{FB}}_{A,n(\lambda)}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

  where $\mathsf{Expt}^{\mathsf{FB}}_{A,n}(\lambda)$ is defined via the following security game between the adversary A and a challenger on common input 1^λ:

  1. On input the security parameter λ, A outputs a function $f \in \mathcal{F}_\lambda$.
  2. The challenger samples a hash key $\mathsf{hk} \leftarrow \mathsf{SetupBinding}(1^\lambda, n, f)$ for the function f and gives hk to A.
  3. Algorithm A outputs a sequences of hash inputs $(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k)$ where each $\mathsf{hinp}_i \in \{0,1\}^{\ell_{\mathsf{blk}}(\lambda)}$ and $k \leq n$. Additionally, A outputs a set $S \subseteq [k]$ with associated values $\{(i, \mathsf{hinp}^\star_i)\}_{i \in S}$ and a proof π.
  4. The experiment outputs 0 if there exists an extension of remaining inputs $\{(j, \mathsf{hinp}^\star_j)\}_{j \in [k] \setminus S}$ such that $f(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = f(\mathsf{hinp}^\star_1, \ldots, \mathsf{hinp}^\star_k)$. Otherwise, let $\mathsf{dig} = \mathsf{Hash}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k))$. The output of the experiment is $\mathsf{VerOpen}(\mathsf{hk}, \mathsf{dig}, S, \{(i, \mathsf{hinp}^\star_i)\}_{i \in S}, \pi)$.

- **Computational function hiding:** For all stateful PPT algorithms A and efficiently-computable functions $n \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} f \leftarrow A(1^\lambda) \\ \mathsf{hk}_0 \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda)) \\ \mathsf{hk}_1 \leftarrow \mathsf{SetupBinding}(1^\lambda, n(\lambda), f) \\ b \xleftarrow{R} \{0,1\} \end{array} : f \in \mathcal{F}_\lambda \wedge A(\mathsf{hk}_b) = b\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

Remark 3.2 (Somewhere Statistically Binding Hash Functions)**.** Definition 3.1 captures the notion of a somewhere statistically binding hashing [HW15] as a special case where the function class $\mathcal{F}$ is the class of *index functions* (i.e., functions f_i where $f_i(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_n) = \mathsf{hinp}_i$).

3.1 Function-Binding Hash Functions for Disjunctions of Block Functions

We construct a function-binding hash for the function class consisting of disjunctions of block functions:

Definition 3.3 (Disjunction of Block Functions)**.** Let $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ be an input length parameter. We say a function $f_g \colon (\{0,1\}^{\ell_{\mathsf{blk}}})^* \to \{0,1\}$ is a disjunction of block function if we can express it as

$$f_g(x_1, \ldots, x_k) = \bigvee_{i \in [k]} g(x_i), \tag{3.1}$$

and where the function $g \colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ is a predicate on ℓ_{blk}-bit inputs. We say that $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ is the class of disjunctions for block functions with input length $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, size $s = s(\lambda)$, and depth $d = d(\lambda)$ if $\mathcal{F}_\lambda$ contains all functions $f_g \colon (\{0,1\}^{\ell_{\mathsf{blk}}})^* \to \{0,1\}$ where the function g can be computed by a Boolean circuit of size s and depth d.

Our construction closely follows the construction of somewhere statistically binding hash functions of [HW15], which uses a Merkle tree [Mer89] where the underlying two-to-one hash function is built from a leveled homomorphic encryption scheme. We first introduce some preliminary notation and background on Merkle trees and then describe our full construction.

Merkle tree preliminaries. We recall some preliminary definitions regarding Merkle trees [Mer89] that we will use in our construction. A Merkle tree MT with depth α is a complete binary tree with 2^α leaf nodes at level 0 in the tree. Each leaf node is associated with an arbitrary fixed-length value. The values of the intermediate nodes in level $i \in [0, \alpha - 1]$ is defined by applying a two-to-one hash function to the two values associated with its child nodes. As such, there are $2^{\alpha - j}$ nodes at level j in the tree MT, and a single node at level α, which we refer to as the *root* node. We view the Merkle tree MT as a directed acyclic graph with edges running from each node to its parent. We index

each node in the tree by a pair $(j, i) \in [0, \alpha] \times [1, 2^{\alpha - j}]$ corresponding to its level j and its index i at level j, ordered from left to right.

Merkle tree notation. We also introduce some helpful notation for referring to different sets of nodes in the tree. Our treatment follows the generalization of authentication paths in Merkle trees used by [EFKP20]. Let MT be a depth α Merkle tree. For a leaf at index $i \in [2^\alpha]$, we write $\mathsf{path}(i)$ to denote the set of nodes along the directed path from leaf i to the root. For a set of leaves $S \subseteq [2^\alpha]$, we define the subtree corresponding to S, denoted $\mathsf{ST}(S)$, to be the union of all paths associated with the leaf nodes $i \in S$: $\mathsf{ST}(S) = \bigcup_{i \in S} \mathsf{path}(i)$. This is sometimes referred to as the Steiner tree for S (c.f., [NNL01]). We define the sibling of a node (j, i), denoted $\mathsf{sib}((j, i))$, to be the adjacent node in the tree who shares a parent with the node; this is either node $(j, i-1)$ if i is even or node $(j, i+1)$ is i is odd.

Definition 3.4 (Dangling Nodes)**.** Let MT be a depth α Merkle tree, and let $S \subseteq [2^\alpha]$. We define the set $\mathsf{dangling}(S)$ to be the set of nodes "dangling" off of the subtree $\mathsf{ST}(S)$. These are the siblings of nodes in $\mathsf{ST}(S)$ which are not contained in $\mathsf{ST}(S)$. Formally,

$$\mathsf{dangling}(S) = \{\mathsf{sib}(\mathsf{node}) : \mathsf{node} \in \mathsf{ST}(S) \wedge \mathsf{sib}(\mathsf{node}) \notin \mathsf{ST}(S)\}.$$

Normally, one can prove membership of a leaf node i in a Merkle tree (with respect to a Merkle root) by providing the values associated with $\mathsf{dangling}(\{i\})$; this is referred to as the authenticating path of i. To check the proof of membership, the verifier uses the value of node i together with the values of the nodes in $\mathsf{dangling}(\{i\})$ to compute the root of the Merkle tree and compare with the given value. More generally, $\mathsf{dangling}(S)$ defines an authentication path for a set of leaves. Namely, given the values of the nodes in S together with those in $\mathsf{dangling}(S)$, one can again compute the value of the root. Assuming the underlying two-to-one hash function is collision-resistant, the value at the root is computationally unique, so the values in $\mathsf{dangling}(S)$ can be used to authenticate the values of S with respect to the root.

Construction 3.5 (Function-Binding Hash for Disjunctions)**.** Let $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ be an input length parameter. Fix a size parameter $s_g = s_g(\lambda)$ and depth parameter $d_g = d_g(\lambda)$. We define the function class $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ to consist of all disjunction-of-block-functions $f_g \colon (\{0,1\}^{\ell_{\mathsf{blk}}})^* \to \{0,1\}$ where

$$f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = \bigvee_{i \in [k]} g(\mathsf{hinp}_i), \tag{3.2}$$

and $g \colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ is a function that can be computed by a Boolean circuit C_g of size s_g and depth d_g. In particular, the function f_g can be computed by a circuit of size at most $s_g(\lambda) \cdot (2k+1) \in s_g(\lambda) \cdot \mathsf{poly}(k)$. Our construction relies on the following building blocks:

- Let $U(\cdot, \cdot)$ be a universal circuit that takes as input a circuit C_g of size s_g and depth d_g as well as an input $x \in \{0,1\}^{\ell_{\mathsf{blk}}}$ and outputs $C_g(x)$. Such a universal circuit exists with size $s \in \mathsf{poly}(s_g)$ and depth $d_0 \in O(d_g)$ [CH85].
- Let $\mathsf{LHE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a leveled homomorphic encryption scheme with polylogarithmic depth decryption (Definition 2.5). Let $d_0(\lambda) \in O(d_g(\lambda))$ be the depth of the universal circuit U defined above, and for $j = 1, 2, \ldots, \lceil \log_2 n \rceil$, we recursively define $d_j(\lambda)$ to be the depth of the decryption circuit $\mathsf{LHE.Dec}$ instantiated with security parameter λ and depth parameter $d_{j-1}(\lambda)$. Note that since LHE supports polylogarithmic depth decryption, there exists a polynomial p such that $d_j(\lambda) \le p(\lambda, \log j)$ for all $j \in [0, \lceil \log_2 n \rceil]$.

We show how to construct a function-binding hash for the function family $\mathcal{F}$. We construct the function-binding hash function $\mathsf{FBH} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{ProveOpen}, \mathsf{VerOpen})$ for the class of functions $\mathcal{F}$ as follows:

- $\mathsf{Setup}(1^\lambda, n)$: On input the security parameter λ and a bound on the number of inputs $n \in \mathbb{N}$, let $\alpha = \lceil \log_2 n \rceil$. For $j \in [0, \alpha]$, sample encryption keys $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{d_j(\lambda)+1})$, where $d_j(\lambda)$ is defined above. Let $c_j \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}_j, \mathsf{sk}_{j-1})$ for $j \in [\alpha]$. Let $c_g \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}_0, g_\perp)$ where $g_\perp \colon \{0,1\}^{\ell_{\mathsf{blk}}(\lambda)} \to \{0,1\}$ is a dummy circuit of size $s(\lambda)$ that outputs 0 on all inputs. Output the hash key $\mathsf{hk} = (\mathsf{pk}_0, \ldots, \mathsf{pk}_\alpha, c_1, \ldots, c_\alpha, c_g)$.
- $\mathsf{SetupBinding}(1^\lambda, n, f_g)$: On input the security parameter λ, a bound on the number of inputs n, and a target function $f_g \in \mathcal{F}$, the binding setup function samples $(\mathsf{pk}_0, \ldots, \mathsf{pk}_\alpha, c_1, \ldots, c_\alpha)$ exactly as in $\mathsf{Setup}(1^\lambda, n)$. It then samples $c_g \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}_0, g)$ and outputs the hash key $\mathsf{hk} = (\mathsf{pk}_0, \ldots, \mathsf{pk}_\alpha, c_1, \ldots, c_\alpha, c_g)$.

- $\mathsf{Hash}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k))$: On input the hash key $\mathsf{hk} = (\mathsf{pk}_0, \ldots, \mathsf{pk}_\alpha, c_1, \ldots, c_\alpha, c_g)$, and a tuple of inputs $(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) \in (\{0,1\}^{\ell_{\mathsf{blk}}})^k$, the hash algorithm sets $\alpha' = \lceil \log_2 k \rceil$ and $k' = 2^{\alpha'}$. Then, for each $i \in [k']$, it constructs ciphertexts as follows:

  - If $i \leq k$, it homomorphically computes $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_0, U(\cdot, \mathsf{hinp}_i), c_g)$ where $U(\cdot, \mathsf{hinp}_i)$ is the universal circuit defined above.
  - If $i > k$, it deterministically sets $\mathsf{ct}_i \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}_0, 0; 0^\lambda)$ using fixed randomness 0^λ.

  The hash algorithm now constructs a Merkle tree MT of depth α':

  - It associates leaf i with the value ct_i.
  - It uses the following two-to-one hash function h_j to compute the values of the nodes at level $j \in [0, \alpha']$ in the tree (recall that level 0 corresponds to the leaves):

$$h_j(\mathsf{node}_1, \mathsf{node}_2) := \mathsf{LHE.Eval}\big(\mathsf{pk}_j, f_{\mathsf{node}_1, \mathsf{node}_2, j}, c_j\big), \tag{3.3}$$

    where

$$f_{\mathsf{node}_1, \mathsf{node}_2, j}(\mathsf{sk}) := \begin{cases} 1 & \mathsf{LHE.Dec}(\mathsf{sk}, \mathsf{node}_1) = 1 \ \vee \ \mathsf{LHE.Dec}(\mathsf{sk}, \mathsf{node}_2) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

    By design, $f_{\mathsf{node}_1, \mathsf{node}_2, j}$ can be computed by a circuit of depth $d_j(\lambda) + 1$.

  Output $\mathsf{dig} = (\mathsf{root}, k)$ where root is the value associated with the root of the Merkle tree MT.

- $\mathsf{ProveOpen}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k), S)$: On input the hash key hk and a tuple of inputs $(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) \in (\{0,1\}^{\ell_{\mathsf{blk}}})^k$, the prove algorithm starts by computing the Merkle tree MT according to the specification of $\mathsf{Hash}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k))$. Let dvals to be a map consisting of the values associated with the nodes in $\mathsf{dangling}(S)$ in MT. Output $\pi = \mathsf{dvals}$.

- $\mathsf{VerOpen}(\mathsf{hk}, \mathsf{dig}, S, \{(i, \mathsf{hinp}_i)\}_{i \in S}, \pi)$: On input the hash key $\mathsf{hk} = (\mathsf{pk}_0, \ldots, \mathsf{pk}_\alpha, c_1, \ldots, c_\alpha, c_g)$, a digest $\mathsf{dig} = (\mathsf{root}, k)$, a set S, values $\{(i, \mathsf{hinp}_i)\}_{i \in S}$, and a proof $\pi = \mathsf{dvals}$, the verification algorithm first checks that $S \subseteq [k]$ and moreover that dvals corresponds to $\mathsf{dangling}(S)$ for a Merkle tree with $k' = 2^{\lceil \log_2(k) \rceil}$ leaves. If either check fails, it outputs 0. Otherwise, it computes $\mathsf{ct}_i = \mathsf{LHE.Eval}(\mathsf{pk}_0, U(\cdot, \mathsf{hinp}_i), c_g)$ for each $i \in S$. Then, using the values of $\{\mathsf{ct}_i\}_{i \in S}$ together with the values in dvals for the set $\mathsf{dangling}(S)$, the verification algorithm computes the root root′ of the Merkle tree using the two-to-one hash function from the specification of $\mathsf{Hash}(\mathsf{hk}, \cdot)$ (Eq. (3.3)). If root = root′, it outputs 1. Otherwise, it outputs 0.

Correctness and security analysis. We now show that FBH from Construction 3.5 is a secure function binding hash for $\mathcal{F}$. First, we note that completeness of Construction 3.5 holds by construction since VerOpen and ProveOpen perform identical operations to compute the dig. As such, VerOpen will always accept on honestly generated proofs. We proceed to prove efficiency, computational function hiding, and statistical function binding in Theorems 3.6 to 3.8 below:

Theorem 3.6 (Efficiency). *Let $p = p(\cdot, \cdot)$ be a fixed polynomial. Assuming that* LHE *satisfies efficiency and compactness and $d_j(\lambda) \leq p(\lambda, \log j)$ for all $\lambda \in \mathbb{N}$ and $j \in [0, \lceil \log_2 n \rceil]$, then Construction 3.5 satisfies efficiency.*

Proof. We show each of the efficiency properties individually:

- **Setup efficiency:** First, we analyze the efficiency of Setup and SetupBinding. Note that Setup runs $\mathsf{LHE.Setup}(1^\lambda, 1^{d_j(\lambda)+1})$ for each $j \in [\lceil \log_2 n \rceil]$. As $d_j(\lambda) \leq p(\lambda, \log j)$, this takes time $\mathsf{poly}(\lambda, \log n)$. Computing ciphertexts $c_1, \ldots, c_\alpha$ then also takes time $\mathsf{poly}(\lambda, \log n)$ by the efficiency of LHE. Computing c_g takes time $\mathsf{poly}(\lambda, s_g(\lambda), \log n)$ as $g_\perp$ is of size $\mathsf{poly}(\lambda, s_g(\lambda))$. So, Setup (and SetupBinding by the same argument) runs in total time $\mathsf{poly}(\lambda, s_g(\lambda), \log n)$, as required.

- **Digest size:** Note that root is an encryption of a bit under LHE for public key $\mathsf{pk}_{\alpha'}$ where $\alpha' = \lceil \log_2 k \rceil$. As $d_{\alpha'}(\lambda) \leq p(\lambda, \log \alpha')$, it follows by the efficiency and compactness of LHE that $|\mathsf{root}| \in \mathsf{poly}(\lambda, \log k)$. Also, $|k| \leq \alpha'$, so together $|\mathsf{dig}| \in \mathsf{poly}(\lambda, \log k)$.
- **Proof size:** The proof consists of at most $|S| \cdot \alpha'$ ciphertexts corresponding to the values of nodes in the Merkle tree. As argued for the digest size, each of the ciphertexts are of size $\mathsf{poly}(\lambda, \log k)$, so in total the size of the proof $\pi \in |S| \cdot \mathsf{poly}(\lambda, \log k)$, as required. □

Theorem 3.7 (Computational Function Hiding)**.** *Assuming* LHE *is semantically secure, then Construction 3.5 satisfies computational function hiding.*

Proof. Suppose function hiding does not hold. Namely, there exists a stateful PPT algorithm A, a function $n \in \mathsf{poly}(\lambda)$, and a polynomial q such that for infinitely many $\lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} f \leftarrow A(1^\lambda) \\ \mathsf{hk}_0 \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda)) \\ \mathsf{hk}_1 \leftarrow \mathsf{SetupBinding}(1^\lambda, n(\lambda), f) \\ b \xleftarrow{R} \{0,1\} \end{array} : f \in \mathcal{F}_\lambda \wedge A(\mathsf{hk}_b) = b\right] \geq 1/2 + 1/q(\lambda).$$

We construct a stateful algorithm B that breaks the semantic security of LHE with depth parameter $d_0(\lambda) + 1$. We define B as follows.

1. Algorithm B first takes as input a uniformly sampled public key $\hat{\mathsf{pk}}$ for the LHE scheme with depth parameter $d_0(\lambda) + 1$. It computes $f_g \leftarrow A(1^\lambda)$. Algorithm B then outputs $(\mathsf{msg}_0, \mathsf{msg}_1)$ where msg_1 is the description of the circuit g and msg_0 is the description of the dummy circuit $g_\perp$ (from Setup).
2. Algorithm B receives as input a challenge ciphertext $\mathsf{ct}^\star$ corresponding to msg_b for a random $b \xleftarrow{R} \{0,1\}$. Algorithm B computes $\mathsf{Setup}(1^\lambda, n(\lambda))$ to compute the hash key hk, except it sets $c_g = \mathsf{ct}^\star$. Finally, B outputs $b' \leftarrow A(\mathsf{hk})$.

By construction, if algorithm A is efficient, then so is B. It remains to show that B succeeds with the correct probability. Note that whenever the challenge bit $b = 1$, hk is distributed identically to $\mathsf{SetupBinding}(1^\lambda, n(\lambda), f_g)$, and whenever the challenge bit $b = 0$, hk is distributed identically to $\mathsf{Setup}(1^\lambda, n(\lambda))$. Furthermore, whenever $f_g \in \mathcal{F}_\lambda$, it holds that $|g| = |g_\perp|$ by construction. So, the probability that B wins is given by

$$\Pr[b' = b] = \Pr[b' = b \wedge f_g \notin \mathcal{F}_\lambda] + \Pr[b' = b \wedge f_g \in \mathcal{F}_\lambda] \geq 1/2 + 1/q(\lambda).$$ □

Theorem 3.8 (Statistical Function Binding)**.** *Assuming* LHE *satisfies perfect correctness, then Construction 3.5 satisfies function binding.*

Proof. Consider any stateful unbounded algorithm A and function $n \in \mathsf{poly}(\lambda)$. We prove a stronger statement and show that *perfect* function binding holds (i.e., function binding holds for the negligible function $\mathsf{negl}(\lambda) = 0$ for all $\lambda \in \mathbb{N}$). Let $f_g \in \mathcal{F}_\lambda$ be any initial function output by A, and let $\mathsf{hk} \in \mathrm{Supp}\left(\mathsf{SetupBinding}(1^\lambda, n(\lambda), f_g)\right)$. Let $(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k)$ for $k \leq n(\lambda)$ be any set of hash inputs (that may depend on hk) and $(\mathsf{root}, k) = \mathsf{dig} = \mathsf{Hash}(\mathsf{hk}, (\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k))$. For any set $S \subseteq [k]$, set of inputs $\{(i, \mathsf{hinp}_i^\star)\}_{i \in S}$, and proof $\pi = \mathsf{dvals}$, let $b = \mathsf{VerOpen}(\mathsf{hk}, \mathsf{dig}, S, \{(i, \mathsf{hinp}_i^\star)\}_{i \in S}, \pi)$. We show that

$$\left(\nexists \{(j, \mathsf{hinp}_j^\star)\}_{j \in [k] \setminus S} : f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = f_g(\mathsf{hinp}_1^\star, \ldots, \mathsf{hinp}_k^\star)\right) \Rightarrow b = 0, \tag{3.4}$$

which suffices to prove the claim. Specifically, we show the following invariant holds for all of the nodes at level $j \in [0, \alpha']$ in the tree:

- If $g(\mathsf{hinp}_i) = 0$ for all $i \in [k]$, then all of the nodes at level j in the tree are encryptions of 0 under pk_j.
- Otherwise, if there exists some index $i \in [k]$ such that $g(\mathsf{hinp}_i) = 1$, then there exists some node at level j that is an encryption of 1 under pk_j.

For the root node (at level α'), there is just a single node, so it is either an encryption of 1 if there exists $i \in [k]$ such that $g(\mathsf{hinp}_i) = 1$ (i.e., $f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = 1$) and is an encryption of 0 otherwise (i.e., $f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = 0$). We now show the invariant inductively:

- **Base case:** For the base case $j = 0$, each leaf node $i \in [k]$ has value $\mathsf{LHE.Eval}(\mathsf{pk}_0, U(\cdot, \mathsf{hinp}_i), c_g)$. By correctness of LHE, this is an encryption of $g(\mathsf{hinp}_i)$. For indices $i \geq k$, each leaf node has value $\mathsf{LHE.Eval}(\mathsf{pk}_0, U(\cdot, \mathsf{hinp}_i), c_\perp)$, which is an encryption of 0 by definition of $c_\perp$ and correctness of LHE. If $g(\mathsf{hinp}_i) = 0$ for all $i \in [k]$, then every leaf node is an encryption of 0 under pk_0. Otherwise, if $g(\mathsf{hinp}_i) = 1$ for some $i \in [k]$, then the i^{th} leaf node is an encryption of 1 under pk_0. The invariant holds.
- **Inductive step:** This follows by definition of h_j in Eq. (3.3). Specifically, suppose there exists a node that is an encryption of 1 under pk_j at level j. Then, this will result in an encryption of 1 under pk_{j+1} in the next level $j + 1$ (specifically, the *parent* node of any node that encrypts 1 in level $j + 1$ will also be an encryption of 1 by construction of Eq. (3.3)). Alternatively, if all of the nodes at level j are encryptions of 0 under pk_j, then the nodes at level $j + 1$ will be encryptions of 0 under pk_{j+1}.

To complete the argument, we now consider the following cases and show that the implication in Eq. (3.4) holds in each case:

- **Case 1a:** Suppose $f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = 1$ and $S = [k]$. In this case, the inputs $\mathsf{hinp}_1^\star, \ldots, \mathsf{hinp}_k^\star$ are all fixed since $S = [k]$. To show Eq. (3.4) holds, it suffices to show that if $f_g(\mathsf{hinp}_1^\star, \ldots, \mathsf{hinp}_k^\star) = 0$, then $b = 0$. Let root′ be the root of the Merkle tree computed on inputs $(\mathsf{hinp}_1^\star, \ldots, \mathsf{hinp}_k^\star)$. By our invariant above, root′ is an encryption of 1 under $\mathsf{pk}_{\alpha'}$. At the same time, root is an encryption of $0 = f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k)$ under $\mathsf{pk}_{\alpha'}$. Perfect correctness of LHE now implies that root $\neq$ root′, in which case VerOpen outputs $b = 0$.
- **Case 1b:** Suppose $f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = 1$ and $S \neq [k]$. Since $S \neq [k]$, we can take any $\mathsf{hinp}_j^\star$ for $j \notin S$ such that $g(\mathsf{hinp}_j^\star) = 1$. Note that if such an $\mathsf{hinp}_j^\star$ does not exist, then it cannot be the case that $f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = 1$. Thus, there always exists a set of inputs $\{(j, \mathsf{hinp}_j^\star)\}_{j \in S \setminus [k]}$ such that $f_g(\mathsf{hinp}_1^\star, \ldots, \mathsf{hinp}_k^\star) = 1$. Eq. (3.4) now holds trivially as the hypothesis is always false in this case.
- **Case 2:** Suppose $f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k) = 0$. Suppose there exists $i \in S$ where $g(\mathsf{hinp}_i^\star) = 1$ for some $i \in S$; otherwise there always exists values $\mathsf{hinp}_j^\star$ for $j \in [k] \setminus S$ that make $f_g(\mathsf{hinp}_1^\star, \ldots, \mathsf{hinp}_k^\star) = 0$. Let root′ be the root of the Merkle tree computed on an input that contains $\mathsf{hinp}_i^\star$. By our invariant above, this means that root′ is an encryption of 1 under $\mathsf{pk}_{\alpha'}$. However, root is an encryption of $0 = f_g(\mathsf{hinp}_1, \ldots, \mathsf{hinp}_k)$ under $\mathsf{pk}_{\alpha'}$. It again follows that root $\neq$ root′ by perfect correctness of LHE, so VerOpen outputs $b = 0$, as required. □

4 Flexible Broadcast Encryption

In this section, we introduce the notion of flexible broadcast encryption. As described in Section 1.1, a flexible broadcast encryption allows anyone to encrypt a message to an arbitrary set of public keys with a ciphertext whose size scales sublinearly with the size of the broadcast set. Unlike traditional broadcast encryption [FN93], there is no central authority that is responsible for issuing decryption keys to users. Instead, users generate their own public and secret keys, and the encryption algorithm takes in the list of public keys for the users in a broadcast set and outputs a single *short* ciphertext that can be decrypted by every member in the broadcast set. Note that this notion of broadcast encryption implies standard broadcast encryption with short ciphertexts and a long public key (by having a central broadcast authority generate the keys for each user and publishing all of them as part of the master public key). Flexible broadcast encryption also generalizes the notion of distributed broadcast encryption introduced by Boneh and Zhandry [BZ14]. The main difference is that in distributed broadcast encryption, users generate keys with respect to a specific index and one can broadcast to at most one key for each index. Flexible broadcast encryption imposes no requirement on the public keys to which one may broadcast. We now give the formal definition:

Definition 4.1 (Flexible Broadcast Encryption)**.** Let $m \in \mathsf{poly}(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ be a message space for messages of length m. A *flexible broadcast encryption scheme* FBE with message space $\mathcal{M}$ consists of polynomial-time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, n) \to \mathsf{pp}$: A probabilistic algorithm that on input a security parameter λ and a bound on the size of the broadcast set n (in *binary*), outputs public parameters pp. We implicitly assume that pp contains 1^λ and n.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: A probabilistic algorithm that on input public parameters pp, outputs a public key pk and a secret key sk.
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)) \to \mathsf{ct}$: A probabilistic algorithm that on input public parameters pp, a message $\mathsf{msg} \in \mathcal{M}_\lambda$, and an ordered sequence of k public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$ for some $k \le n$, outputs a ciphertext ct.[7]
- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)) \to \mathsf{msg}$: A deterministic algorithm that on input public parameters pp, a ciphertext ct, an index and secret key pair (j, sk_j), and public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$ where $k \le n$, outputs a message $\mathsf{msg} \in \mathcal{M}_\lambda \cup \{\bot\}$.

We require that $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfy the following properties:

- **Succinct ciphertexts:** There exists a polynomial p such that for all $\lambda \in \mathbb{N}$, $k \in \mathbb{N}$, $n \in \mathbb{N}$ with $k \le n$, public parameters $\mathsf{pp} \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda, n)\right)$, public/secret keys $(\mathsf{pk}_i, \mathsf{sk}_i) \in \mathsf{Supp}\left(\mathsf{KeyGen}(\mathsf{pp})\right)$ for $i \in [k]$, messages $\mathsf{msg} \in \mathcal{M}_\lambda$, and ciphertexts $\mathsf{ct} \in \mathsf{Supp}\left(\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))\right)$, it holds that $|\mathsf{ct}| \le p(\lambda, m(\lambda), \log n)$.
- **Correctness:** There exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, $n \in \mathsf{poly}(\lambda)$, $k \le n(\lambda)$, $j \in [k]$, and $\mathsf{msg} \in \mathcal{M}_\lambda$, it holds that

$$\Pr\left[\begin{array}{ll} & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda)) \\ & \forall i \in [k], (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ & \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)) \\ : & \mathsf{msg}' = \mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)) \end{array} \; : \mathsf{msg}' = \mathsf{msg}\right] \ge 1 - \mathsf{negl}(\lambda).$$

  We say that the scheme satisfies perfect correctness if the above probability is equal to 1.
- **Adaptive security:** For all stateful PPT adversaries A and all efficiently-computable functions $n \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Expt}^{\mathsf{FBE}}_{A,n(\lambda)}(\lambda) = 1\right] \le 1/2 + \mathsf{negl}(\lambda),$$

  where $\mathsf{Expt}^{\mathsf{FBE}}_{A,n}(\lambda)$ is defined via the following security game between the adversary A and a challenger on common input 1^λ:
  - **Setup phase:** The challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n)$ and sends pp to A. The challenger also initializes a counter $\mathsf{ctr} := 0$, a dictionary D, and a set of (corrupted) indices $C = \emptyset$.
  - **Pre-challenge query phase:** The adversary A can now issue the following queries:
    - **Key-generation query:** In a key-generation query, the challenger increments $\mathsf{ctr} := \mathsf{ctr} + 1$, samples a key-pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and replies to A with $(\mathsf{ctr}, \mathsf{pk})$. The challenger adds the mapping $\mathsf{ctr} \mapsto (\mathsf{pk}, \mathsf{sk})$ to the dictionary D.
    - **Corruption query:** In a corruption query, the adversary specifies a counter value $c \in [\mathsf{ctr}]$. In response, the challenger looks up $(\mathsf{pk}, \mathsf{sk}) := \mathsf{D}[c]$, replies to A with sk, and adds ctr to C.
  - **Challenge phase:** Algorithm A computes two messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}_\lambda$ and an ordered list $S^\star = (i_1, \ldots, i_{k^\star}) \subseteq [\mathsf{ctr}]$ that it sends to the challenger. If $S^\star \cap C \neq \emptyset$, the experiment outputs 0. Otherwise, the challenger samples a bit $b \xleftarrow{R} \{0,1\}$ and computes an encryption of msg_b under the keys corresponding to $S^\star$: $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}_b, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_{k^\star}}))$. The challenger sends $\mathsf{ct}^\star$ to A.
  - **Output phase:** Algorithm A outputs a bit $b' \in \{0,1\}$. The experiment outputs 1 if $b' = b$ and 0 otherwise.

[7] Here, we assume an ordered list of public keys for simplicity. However, we could have alternatively encrypted to an (unordered) set of public keys by first ordering the public keys in lexicographic order.

Semi-static security. In this work, we also consider a weaker notion of semi-static security introduced by Gentry and Waters [GW09] in the context of broadcast encryption. In the setting of flexible broadcast encryption, semi-static security corresponds to the setting where the adversary is allowed to make any number of honest key-generation queries, and then declares its challenge set to be some subset of these keys. The adversary cannot make any corruption queries in this model. In the context of broadcast encryption, Gentry and Waters showed that semi-static security implies adaptive security in the random oracle model. In Appendix B, we show that the same technique also applies in the case of flexible broadcast encryption.

Definition 4.2 (Semi-Static Security [GW09, adapted])**.** We say that a flexible broadcast encryption scheme (Setup, KeyGen, Enc, Dec) satisfies semi-static security if instead of adaptive security, it satisfies the following:

- **Semi-static security:** For all stateful PPT adversaries A and all efficiently-computable functions $n \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Expt}^{\mathsf{FBE,SS}}_{A,n(\lambda)}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

  where $\mathsf{Expt}^{\mathsf{FBE,SS}}_{A,n}(\lambda)$ is identical to the adaptive security game $\mathsf{Expt}^{\mathsf{FBE}}_{A,n(\lambda)}(\lambda)$, except the adversary A is not allowed to make any corruption queries.

4.1 Constructing Flexible Broadcast Encryption

We now show how to construct a flexible broadcast encryption scheme by combining a witness encryption scheme with a function-binding hash function and a vanilla public-key encryption scheme. Note that public-key encryption is implied by combining witness encryption and one-way functions [GGSW13]. Thus, the additional assumption of public-key encryption is technically unnecessary (since function-binding hash functions imply one-way functions). Our construction satisfies semi-static security. However, we show in Appendix B how to use this to construct an adaptively secure scheme via a transformation following the approach of [GW09].

Construction 4.3 (Flexible Broadcast Encryption)**.** Let $s = s(\lambda)$, $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, and $d = d(\lambda)$ be polynomials, and $m = m(\lambda)$ be any function. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space for messages of length m. Our construction of flexible broadcast encryption relies on the following primitives:

- Let $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a semantically-secure public-key bit encryption scheme where for $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Supp}\left(\mathsf{PKE.KeyGen}(1^\lambda)\right)$, ciphertexts have length at most $\ell_{\mathsf{blk}}(\lambda)$ and decryption can be computed by a circuit of size $s(\lambda)$ and depth $d(\lambda)$.
- Let $\mathsf{FBH} = (\mathsf{FBH.Setup}, \mathsf{FBH.SetupBinding}, \mathsf{FBH.Hash}, \mathsf{FBH.ProveOpen}, \mathsf{FBH.VerOpen})$ be a function-binding hash for the class $\mathcal{F}$ of disjunctions of block functions for input length ℓ_{blk}, size s, and depth d (Definition 3.3).
- Let $\mathsf{WE} = (\mathsf{WE.Enc}, \mathsf{WE.Dec})$ be a witness encryption scheme for the language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda\in\mathbb{N}}$ defined by the relation $\mathbf{R}_{\mathcal{L}}$ as follows. Instances of the language $\mathcal{L}_\lambda$ are of the form $(\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig})$ and the relation $\mathbf{R}_{\mathcal{L}}$ is given by

$$\begin{aligned}&\mathbf{R}_{\mathcal{L}}\left(1^\lambda, (\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}), (i, \mathsf{hinp}_i, r, \pi)\right) = 1\\ &\Leftrightarrow \mathsf{hinp}_i = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; r) \ \wedge\ \mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{i\}, \{(i, \mathsf{hinp}_i)\}, \pi) = 1.\end{aligned}$$

We construct a flexible broadcast encryption scheme $\mathsf{FBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ as follows:

- $\mathsf{Setup}(1^\lambda, n)$: On input the security parameter λ and a bound on the broadcast set size n, the setup algorithm samples $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, n)$, and outputs the public parameters $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$. Note that we implicitly assume that pp contain 1^λ and n.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$, the key-generation algorithm samples $r \xleftarrow{R} \{0,1\}^\lambda$ and computes $\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; r)$. Output the public key $\mathsf{pk} = \mathsf{ct}$ and the secret key $\mathsf{sk} = r$.

- $\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$: On input the public parameters $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$, the message $\mathsf{msg} \in \mathcal{M}_\lambda$, and public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_k$, the encryption algorithm computes the digest $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$ and outputs the ciphertext $\mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathsf{msg}, (\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}))$.
- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$: On input the public parameters $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$, the ciphertext ct, an index $j \in [n]$, the associated secret key sk_j, and the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_k$, the decryption algorithm computes the digest $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$, the opening $\pi = \mathsf{FBH.ProveOpen}(\mathsf{hk}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k), \{j\})$, and finally, outputs the message $\mathsf{msg} = \mathsf{WE.Dec}(1^\lambda, \mathsf{ct}, (j, \mathsf{pk}_j, \mathsf{sk}_j, \pi))$.

Correctness and security analysis. We now show that FBE from Construction 4.3 is a flexible broadcast encryption scheme.

Theorem 4.4 (Correctness and Succinctness)**.** *Assuming* PKE *and* WE *are correct, and* FBH *is complete, then Construction 4.3 has succinct ciphertexts and is correct.*

Proof. We consider each property separately:

- **Succinct ciphertexts:** This follows by efficiency of WE and FBH. Specifically, checking the relation $\mathbf{R}_{\mathcal{L}}$ requires time at most $\mathsf{poly}(\lambda, s(\lambda), \log n)$ by the efficiency of FBH, which is $\mathsf{poly}(\lambda, \log n)$ for polynomially-bounded s, ℓ_{blk}. Correspondingly, efficiency of WE implies that $|\mathsf{ct}| \in \mathsf{poly}(\lambda, m(\lambda), \log n)$, as required.
- **Perfect correctness:** We show perfect correctness of the scheme. Take any security parameter $\lambda \in \mathbb{N}$, bound on the number of participants $n \in \mathsf{poly}(\lambda)$, $k \le n(\lambda)$, index $j \in [k]$, and message $\mathsf{msg} \in \mathcal{M}_\lambda$. Then, we have the following:
  - Let $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk}) \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda))$, and for each $i \in [k]$, let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. By construction, this means $\mathsf{pk}_i = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; r_i)$ and $\mathsf{sk}_i = r_i$.
  - Let $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$. Then, $\mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathsf{msg}, (\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}))$ where the digest is $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$.
  - We now argue that $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)) = \mathsf{msg}$. The decryption algorithm computes the digest dig as above and the proof $\pi = \mathsf{FBH.ProveOpen}(\mathsf{hk}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k), \{j\})$. By completeness of FBH, we conclude $\mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{j\}, \{(j, \mathsf{pk}_j)\}, \pi) = 1$. By construction $\mathsf{pk}_j = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; \mathsf{sk}_{\mathsf{PKE}})$. This means that
  $$\mathbf{R}_{\mathcal{L}}\big(1^\lambda, (\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}), (j, \mathsf{pk}_j, \mathsf{sk}_j, \pi)\big) = 1,$$
  and correctness now follows by correctness of WE. □

Theorem 4.5 (Semi-Static Security)**.** *Assuming* PKE *is semantically secure and satisfies perfect correctness,* FBH *satisfies computational function hiding and statistical function binding, and* WE *satisfies message indistinguishability, then Construction 4.3 satisfies semi-static security.*

Proof. Suppose by way of contradiction that FBE does not satisfy semi-static security. Namely, there exists a stateful PPT algorithm A, an efficiently-computable function $n \in \mathsf{poly}(\lambda)$, a polynomial q, and an infinite set $\Lambda \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$, it holds that
$$\Pr\left[\mathsf{Expt}^{\mathsf{FBE,SS}}_{A,n(\lambda)}(\lambda) = 1\right] \ge 1/2 + 1/q(\lambda).$$
For fixed parameters as above, we define the following sequence of hybrid experiments for each $\lambda \in \mathbb{N}$.

- $H_{b,0}(\lambda)$: This hybrid is the experiment $\mathsf{Expt}^{\mathsf{FBE,SS}}_{A,n(\lambda)}(\lambda)$ where the challenge bit is fixed to $b \in \{0, 1\}$.
- $H_{b,1}(\lambda)$: This hybrid experiment is the same as $H_{b,0}$ except the challenger computes pk to be an encryption of 0 under $\mathsf{pk}_{\mathsf{PKE}}$ for each honest key-generation query. Specifically, the challenger uses $(\mathsf{pk}, \mathsf{sk}) = (\mathsf{ct}, r)$ where $r \leftarrow \{0,1\}^\lambda$ and $\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 0; r)$.
- $H_{b,2}(\lambda)$: In this hybrid, the challenger instead computes $\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^\lambda, n(\lambda), f_g)$, where f_g is the function from Eq. (3.1) and g is the function computing $\mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}, \cdot)$.

- $H_{b,3}(\lambda)$: In this hybrid, instead of encrypting msg_b in the semi-static security game, the challenger encrypts the lexicographically-first message $\mathsf{msg}^\star$ in $\mathcal{M}_\lambda$: $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}^\star, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_k^\star}))$.

Since the challenge bit is sampled uniformly (i.e., $b \xleftarrow{R} \{0,1\}$), the success probability of A on any security parameter λ is equal to

$$\Pr[A \text{ wins}] = \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right).$$

For each $b \in \{0,1\}$ and $i \in \{1,2,3\}$, we define

$$\delta_{b,i}(\lambda) = \Pr[H_{b,i-1}(\lambda) = 1] - \Pr[H_{b,i}(\lambda) = 1].$$

It follows by our assumption that for all $\lambda \in \Lambda$,

$$\begin{aligned}\frac{1}{2} + \frac{1}{q(\lambda)} &\leq \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right) \\ &= \frac{1}{2} \cdot \big(\Pr[H_{0,3}(\lambda) = 1] + \delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) + \delta_{0,3}(\lambda) \\ &\qquad + \Pr[H_{1,3}(\lambda) = 1] + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda) + \delta_{1,3}(\lambda)\big).\end{aligned}$$

As $H_{b,3}(\lambda)$ is independent of b for each $b \in \{0,1\}$, it holds that $\Pr[H_{b,3}(\lambda) = 1] \leq 1/2$. It follows that for all $\lambda \in \Lambda$,

$$\frac{1}{q(\lambda)} \leq \frac{1}{2} \cdot \left(\delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) + \delta_{0,3}(\lambda) + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda) + \delta_{1,3}(\lambda)\right).$$

As Λ is an infinitely large subset of $\mathbb{N}$, it must be the case that for some $b \in \{0,1\}$ and $i \in \{1,2,3\}$, $\delta_{b,i}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. In the following three claims, we show that this must contradict one of our assumptions from the statement of Theorem 4.5. We prove the following claims for $b = 0$ without loss of generality.

Claim 4.6. *If $\delta_{0,1}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then* PKE *does not satisfy semantic security.*

Proof. Let $Q(\lambda)$ be a polynomial upper bounding the number of honest key-generation queries that A makes, which must exist as A is PPT. We construct an adversary B that uses A to break the semantic security of PKE. We define B as follows:

1. Algorithm B first takes as input a public key $\widehat{\mathsf{pk}}$ for the PKE scheme and outputs $\mathsf{msg}_0 = 0$ and $\mathsf{msg}_1 = 1$ as its challenge message.
2. Algorithm B then receives a challenge ciphertext $\widehat{\mathsf{ct}}$, corresponding to the message $\hat{b}$ which is either 0 or 1. It computes the guess $\hat{b}'$ for $\hat{b}$ as follows:
   (a) Algorithm B computes $Q(\lambda)$ and samples a uniformly random index $i^\star \leftarrow [n(\lambda)]$.
   (b) Algorithm B then samples $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, n(\lambda))$, sets $\mathsf{pp} = (\widehat{\mathsf{pk}}, \mathsf{hk})$, and sends pp to A.
   (c) For each counter value $\mathsf{ctr} \in [Q(\lambda)]$, algorithm B computes the response pk to A's honest key-generation for ctr as follows:
      - If $\mathsf{ctr} = i^\star$, B sets $\mathsf{pk} = \widehat{\mathsf{ct}}$.
      - If $\mathsf{ctr} > i^\star$, B samples $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$ and sets $\mathsf{pk} = \mathsf{PKE.Enc}(\widehat{\mathsf{pk}}, 1; \mathsf{sk})$.
      - If $\mathsf{ctr} < i^\star$, B samples $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$ and sets $\mathsf{pk} = \mathsf{PKE.Enc}(\widehat{\mathsf{pk}}, 0; \mathsf{sk})$.
   (d) Algorithm B then receives the challenge messages $\mathsf{msg}_0, \mathsf{msg}_1$ and challenge set $S^\star$ from A, computes $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}_0, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_{k^\star}}))$ where $S^\star = \{i_1, \ldots, i_{k^\star}\}$, and sends $\mathsf{ct}^\star$ to A.
   (e) The adversary A outputs a bit $b' \in \{0,1\}$, which B outputs.

If A is PPT, then so is algorithm B by construction. It remains to analyze the success probability of B. To do so, we define a sequence of sub-hybrids. For each $i \in [0, Q(\lambda)]$, we define $H_{0,0,i}(\lambda)$ as follows:

- $H_{0,0,i}(\lambda)$: This experiment is the same as $H_{0,0}(\lambda)$ except that we sample the public keys for $j \le k$ where $j \le i$ to be encryptions of 0. Namely, we set $\mathsf{pk}_j = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 0; r)$ for a random $r \leftarrow \{0,1\}^\lambda$ as in $H_{0,1}(\lambda)$.

Note that $H_{0,0,0}(\lambda)$ is identical to $H_{0,0}(\lambda)$ and $H_{0,0,Q(\lambda)}$ is identical to $H_{0,1}(\lambda)$, so in particular, it holds that

$$\begin{aligned}\delta_{0,1}(\lambda) &= \Pr[H_{0,0}(\lambda) = 1] - \Pr[H_{0,1}(\lambda) = 1] \\ &= \sum_{i=1}^{Q(\lambda)} \left(\Pr[H_{0,0,i-1}(\lambda) = 1] - \Pr[H_{0,0,i}(\lambda) = 1]\right).\end{aligned}$$

We proceed to analyze the success probability of B conditioned on $i^\star = i$. Note that when $\hat{b} = 1$, the view of A corresponds to the hybrid $H_{0,0,i}$ since $\mathsf{pk}_{i^\star}$ is an encryption of 0. If $\hat{b} = 0$, this corresponds to the hybrid $H_{0,0,i-1}$. Since B outputs $\hat{b}' = b'$, it follows if $\hat{b} = 1$, then $\hat{b}' = \hat{b}$ whenever $b' = 1$ so $H_{0,0,i}(\lambda) = 0$ (i.e. A loses since b is fixed to 0). Similarly, if $\hat{b} = 0$, then $\hat{b}' = \hat{b}$ whenever $b' = 0$ so $H_{0,0,i-1}(\lambda) = 1$ (i.e. A wins). Thus, the success probability of B for any $i \in [Q(\lambda)]$ conditioned on $i^\star = i$ is given by the following:

$$\begin{aligned}\Pr[\hat{b}' = \hat{b} \mid i^\star = i] &= \frac{1}{2} \cdot \left(1 - \Pr[H_{0,0,i}(\lambda) = 1]\right) + \frac{1}{2} \cdot \Pr[H_{0,0,i-1}(\lambda) = 1] \\ &= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[H_{0,0,i-1}(\lambda) = 1] - \Pr[H_{0,0,i}(\lambda) = 1]\right).\end{aligned}$$

It follows that the success probability of B, where B chooses a random $i^\star$, is equal to

$$\begin{aligned}\Pr[\hat{b}' = \hat{b}] &= \sum_{i=1}^{Q(\lambda)} \Pr[i^\star = i] \cdot \Pr[\hat{b}' = \hat{b} \mid i^\star = i] \\ &= \sum_{i=1}^{Q(\lambda)} \frac{1}{Q(\lambda)} \cdot \left(\frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[H_{0,0,i-1}(\lambda) = 1] - \Pr[H_{0,0,i}(\lambda) = 1]\right)\right) \\ &= \frac{1}{2} + \frac{1}{2 \cdot Q(\lambda)} \cdot \sum_{i=1}^{Q(\lambda)} \left(\Pr[H_{0,0,i-1}(\lambda) = 1] - \Pr[H_{0,0,i}(\lambda) = 1]\right) \\ &= \frac{1}{2} + \frac{1}{2 \cdot Q(\lambda)} \cdot \delta_{0,1}(\lambda).\end{aligned}$$

As $Q \in \mathsf{poly}(\lambda)$ and $\delta_{0,1}(\lambda) \ge 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$ by assumption, it follows that B succeeds with inverse polynomial probability for infinitely many $\lambda \in \mathbb{N}$. This violates semantic security of PKE, as required. □

Claim 4.7. *If $\delta_{0,2}(\lambda) \ge 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then* FBH *does not satisfy computational function hiding.*

Proof. We construct a stateful PPT adversary B that uses the adversary A to break the function hiding security of FBH. We define B as follows:

1. On input the security parameter λ, algorithm B samples $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and outputs the function f_g from Eq. (3.1) where g is the function computing $\mathsf{Dec}(\mathsf{sk}_{\mathsf{PKE}}, \cdot)$. Note that $f_g \in \mathcal{F}$ since g can be computed by a size $s(\lambda)$ circuit by assumption.

2. Algorithm B receives $\mathsf{hk}_{\hat{b}}$ as input for some $\hat{b} \in \{0,1\}$ and computes $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$. It then proceeds as in the experiment $H_{0,1}(\lambda)$, setting $\mathsf{hk} = \mathsf{hk}_{\hat{b}}$, until A outputs a bit b', which B outputs.

Note that B only runs A and otherwise performs polynomial-time computation, so it remains to analyze the success probability of B at distinguishing $\mathsf{hk}_0 \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda))$ versus $\mathsf{hk}_1 \leftarrow \mathsf{SetupBinding}(1^\lambda, n(\lambda), f_g)$. When hk is sampled from Setup (the case where $\hat{b} = 0$), algorithm A's view is distributed identically to hybrid $H_{0,1}(\lambda)$, so $\hat{b} = b'$ whenever A outputs $b' = 0$, which corresponds to the event that A wins since b is fixed to 0. Therefore, B wins in this case with probability $\Pr[H_{0,1}(\lambda) = 1]$. Similarly, when hk is sampled from SetupBinding (the case where $\hat{b} = 1$),

algorithm A's view is identically distributed to hybrid $H_{0,2}(\lambda)$, so $\hat{b} = b'$ whenever A loses, which happens with probability $(1 - \Pr[H_{0,2}(\lambda) = 1])$. It follows that

$$\begin{aligned}
&\Pr\left[\begin{array}{l} f_g \leftarrow B(1^\lambda) \\ \mathsf{hk}_0 \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda)) \\ \mathsf{hk}_1 \leftarrow \mathsf{SetupBinding}(1^\lambda, n(\lambda), f_g) \\ b \xleftarrow{R} \{0,1\} \end{array} : f_g \in \mathcal{F}_\lambda \wedge B(\mathsf{hk}_b) = b\right] \\
&\quad \geq \frac{1}{2} \cdot \Pr[H_{0,1}(\lambda) = 1] + \frac{1}{2} \cdot (1 - \Pr[H_{0,2}(\lambda) = 1]) \\
&\quad = 1/2 + \delta_{0,2}(\lambda)/2.
\end{aligned}$$

As $\delta_{0,2}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, this violates computational function hiding of FBH, as required. □

Claim 4.8. *If $\delta_{0,3}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$ and* PKE *satisfies (perfect) correctness, then either* WE *does not satisfy message indistinguishability or* FBH *does not satisfy statistical function binding.*

Proof. We construct a PPT adversary B that uses adversary A to either break the message indistinguishability security of WE, or the statistical function binding of FBH. We define B as follows:

1. Algorithm B takes 1^λ as input and uses A to compute $(x, \mathsf{msg}_0, \mathsf{msg}_1)$ to send to the WE challenger as follows:
   (a) Algorithm B samples $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^\lambda, n(\lambda), f_g)$. It gives $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$ to A.
   (b) Whenever A makes an honest key-generation query for counter value $\mathsf{ctr} \in \mathbb{N}$, algorithm B samples $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$, sets $\mathsf{pk} = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 0; \mathsf{sk})$, and returns pk.
   (c) In the challenge phase, algorithm A then outputs two messages msg'_0, msg'_1 along with an ordered list $S^\star = (i_1, \ldots, i_{k^\star})$. Algorithm B computes the digest $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_{k^\star}}))$ as in Enc.
   (d) Algorithm B sets $x = (\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig})$, $\mathsf{msg}_0 = \mathsf{msg}'_0$, and msg_1 is the lexicographically first message $\mathsf{msg}^\star$ in $\mathcal{M}_\lambda$.
2. The WE challenger samples a random bit $\hat{b} \xleftarrow{R} \{0,1\}$, and algorithm B receives a challenge ciphertext $\mathsf{ct}^\star$ corresponding to $\mathsf{WE.Enc}(1^\lambda, \mathsf{msg}_{\hat{b}}, x)$. Algorithm B uses $\mathsf{ct}^\star$ as the challenge ciphertext for A. Let $b' \in \{0,1\}$ be the final output of A in the experiment, which B outputs.

Note that B runs A and otherwise performs polynomial-time operations, so B is PPT. It remains to analyze the success probability of B for guessing $\hat{b}' = b'$ such that $\hat{b}' = \hat{b}$. For any $\lambda \in \mathbb{N}$, let $\mathsf{Wit}(\lambda)$ be the event that there exists a valid witness $(j, \mathsf{hinp}_j, r, \pi)$ for the instance $x = (\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig})$ such that

$$\mathsf{hinp}_j = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; r) \wedge \mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{j\}, \{(j, \mathsf{hinp}_j)\}, \pi) = 1. \tag{4.1}$$

Note that the view of A is identical to hybrid $H_{0,2}(\lambda)$ if $\hat{b} = 0$ and is identical to hybrid $H_{0,3}(\lambda)$ if $\hat{b} = 1$ since $\mathsf{msg}_1 = \mathsf{msg}^\star$. Furthermore, when $\hat{b} = 0$, $\hat{b}' = \hat{b}$ whenever $b' = 0$, so $H_{0,2}(\lambda) = 1$ since $b = 0$. When $\hat{b} = 1$, $\hat{b}' = \hat{b}$ whenever $b' = 1$, so $H_{0,3}(\lambda) = 0$ since $b = 0$. Additionally, if $\mathsf{Wit}(\lambda)$ holds, then B always loses the message indistinguishability game since it implies that $x \in \mathcal{L}_\lambda$. Thus, the success probability of B is at least

$$\begin{aligned}
\Pr[B \text{ wins}] &\geq \Pr[\hat{b}' = \hat{b}] - \Pr[\mathsf{Wit}(\lambda)] \\
&\geq \frac{1}{2} \cdot \Pr[H_{0,2}(\lambda) = 1] + \frac{1}{2} \cdot (1 - \Pr[H_{0,3}(\lambda) = 1]) - \Pr[\mathsf{Wit}(\lambda)] \\
&= \frac{1}{2} + \frac{1}{2} \cdot \delta_{0,3}(\lambda) - \Pr[\mathsf{Wit}(\lambda)].
\end{aligned}$$

We now consider two cases depending on the value of $\Pr[\mathsf{Wit}(\lambda)]$:

- If $\Pr[\mathsf{Wit}(\lambda)] \leq 1/(12q(\lambda))$ for infinitely many $\lambda \in \Lambda$, then this violates message indistinguishability of WE since $\delta_{0,3}(\lambda) \geq 1/(3q(\lambda))$.
- Suppose $\Pr[\mathsf{Wit}(\lambda)] > 1/(12q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. This means Eq. (4.1) occurs with probability at least $1/(12q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. In the above construction, $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_{k^\star}}))$ where
$$g(\mathsf{pk}_{i_j}) = \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{pk}_{i_j}) = 0,$$
by perfect correctness of PKE. Thus,
$$f_g(\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_{k^\star}}) = \bigvee_{j \in [k^\star]} g(\mathsf{pk}_{j_i}) = 0.$$
Next, if $\mathsf{hinp}_j = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; r)$ for some randomness $r \in \{0,1\}^\lambda$, then $g(\mathsf{hinp}_j) = 1$, and the output of f_g on *any* input that contains hinp_j is necessarily $1 \neq f_g(\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_{k^\star}})$. As such, if such an input hinp_j exists such that Eq. (4.1) holds with probability at least $1/(12q(\lambda))$, then a computationally unbounded adversary would break statistical function binding of FBH with the same advantage.

Thus, algorithm B either breaks message indistinguishability of the witness encryption scheme or statistical function binding of the function-binding hash. □

The proof now follows from Claims 4.6, 4.7, and 4.8. □

Remark 4.9 (Transparent Setup)**.** We note that the public parameters in our construction consist of a hash key for the function-binding hash and a public key for a public-key encryption scheme. If we instantiate the function-binding hash function using Construction 3.5, then the hash key consists of a sequence of public keys and ciphertexts for a leveled homomorphic encryption scheme. Using a suitable encryption scheme (e.g., [GSW13]), these are all pseudorandom. In other words, the public parameters in Construction 4.3 is pseudorandom and can be instantiated with a uniform random string. This yields a flexible broadcast encryption scheme with a *transparent* setup.

5 Optimal Broadcast Encryption in the Random Oracle Model

As noted in Section 4, a flexible broadcast encryption scheme immediately implies a traditional broadcast encryption scheme with a central trusted authority. Namely, to construct a scheme for n users, the central authority would sample n different public/secret keys $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n)$. The master public key for the broadcast encryption scheme is the concatenation of all n public keys while the secret key for user i is sk_i. While this yields a scheme with short secret keys and ciphertexts, the master public key is very long (scales linearly with the number of users). In this section, we show a simple adaptation of our approach yields an *optimal* broadcast encryption scheme in the *random oracle mode*. In an optimal broadcast encryption scheme [BWZ14], we require that all of the scheme parameters (i.e., the master public key, the user decryption keys, and the ciphertext) to be short (e.g., polylogarithmic with the number of users). We start by defining broadcast encryption:

Definition 5.1 (Broadcast Encryption)**.** Let $m \in \mathsf{poly}(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ be a message space for messages of length m. An *broadcast encryption* scheme with message space $\mathcal{M}$ consists of polynomial-time algorithms (Setup, KeyGen, Enc, Dec) with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{pp}, \mathsf{msk})$: A probabilistic algorithm that on input a security parameter λ and a bound on the number of users n, outputs public parameters pp and a master secret key msk. We implicitly assume that pp and msk contain 1^λ and 1^n.
- $\mathsf{KeyGen}(\mathsf{msk}, i) \to \mathsf{sk}$: A probabilistic algorithm that on input a master secret key msk and an index $i \in [n]$, outputs a secret key sk.
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, S) \to \mathsf{ct}$: A probabilistic algorithm that on input public parameters pp, a message $\mathsf{msg} \in \mathcal{M}_\lambda$, and a set $S \subseteq [n]$, outputs a ciphertext ct.

- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), S) \to \mathsf{msg}$: A deterministic algorithm that on input public parameters pp, a ciphertext ct, an index and secret key pair (j, sk_j), and a set S, outputs a message $\mathsf{msg} \in \mathcal{M}_\lambda \cup \{\perp\}$.

We require that $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfy the following properties:

- **Succinct ciphertexts:** There exists a polynomial p such that for all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $S \subseteq [n]$, $(\mathsf{pp}, \mathsf{msk}) \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda, n)\right)$, $\mathsf{sk}_i \in \mathsf{Supp}\left(\mathsf{KeyGen}(\mathsf{msk}, i)\right)$ for all $i \in S$, $\mathsf{msg} \in \mathcal{M}_\lambda$, and $\mathsf{ct} \in \mathsf{Supp}\left(\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, S)\right)$, it holds that
$$|\mathsf{ct}| \leq p(\lambda, m(\lambda), \log n).$$
- **Correctness:** There exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, $n \in \mathsf{poly}(\lambda)$, $S \subseteq [n(\lambda)]$, $j \in S$, and $\mathsf{msg} \in \mathcal{M}_\lambda$, it holds that
$$\Pr\left[\begin{array}{ll} (\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda)) & \\ \mathsf{sk}_j \leftarrow \mathsf{KeyGen}(\mathsf{msk}, j) & \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, S) & : \mathsf{msg}' = \mathsf{msg} \\ \mathsf{msg}' = \mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), S) & \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$
We say that the scheme satisfies perfect correctness if the above probability is equal to 1.
- **Adaptive security:** For all stateful PPT adversaries A and efficiently-computable functions $n \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,
$$\Pr\left[\mathsf{Expt}^{\mathsf{BE}}_{A,n(\lambda)}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$
where $\mathsf{Expt}^{\mathsf{BE}}_{A,n}(\lambda)$ is defined via the following security game between the adversary A and a challenger on common input 1^λ:
  - **Setup phase:** The challenger samples $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n)$ and sends pp to A. The challenger also initializes a set of corrupted indices $C := \emptyset$.
  - **Pre-challenge query phase:** The adversary A can now issue key-corruption queries for indices $i \in [n]$. On each query, the challenger responds to A with $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$ and adds i to C. Without loss of generality, we assume the adversary queries each index i at most once.
  - **Challenge phase:** Algorithm A outputs two messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}_\lambda$ and a set $S^\star \subseteq [n]$. If $S^\star \cap C \neq \emptyset$, then the experiments halts with output 0. Otherwise, the challenger samples a bit $b \xleftarrow{R} \{0,1\}$ and gives the challenge ciphertext $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}_b, S^\star)$ to A.
  - **Output phase:** Algorithm A outputs a bit $b' \in \{0,1\}$. The output of the experiment is 1 if $b' = b$ and 0 otherwise.

Semi-static security. Similar to the case with flexible broadcast encryption (c.f., Section 4 and Definition 4.2), we also consider the weaker notion of semi-static security from [GW09]. A broadcast encryption scheme satisfies semi-static security if the adversary has to pre-commit to a set $S \subseteq [n]$ at the beginning of the security game. The adversary can then make adaptive key-corruption queries for indices $i \notin S$; for the challenge ciphertext, the adversary is allowed to choose any subset $S^\star \subseteq S$. This is a *stronger* security model than selective security where the adversary is required to commit to its challenge set $S^\star$ at the beginning of the security game. In the semi-static model, the adversary only needs to commit to a *super-set* of its challenge set. Gentry and Waters previously showed that in the random oracle model, a broadcast encryption scheme satisfying semi-static security implies one that satisfies adaptive security. The transformation only incurs a constant factor overhead in the scheme parameters. We now recall the formal definition:

Definition 5.2 (Semi-Static Security [GW09])**.** We say that a broadcast encryption scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies semi-static security if for all stateful PPT adversaries A and efficiently-computable functions $n \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,
$$\Pr\left[\mathsf{Expt}^{\mathsf{BE},\mathsf{SS}}_{A,n(\lambda)}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

where $\mathsf{Expt}^{\mathsf{BE},\mathsf{SS}}_{A,n}(\lambda)$ is defined via the following security game between the adversary A and a challenger on common input 1^λ:

- **Commit phase:** Algorithm A begins by committing to a set of indices $S \subseteq [n]$.
- **Setup phase:** The challenger samples $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n)$ and sends pp to A.
- **Pre-challenge query phase:** The adversary can now make adaptive key queries for indices $i \in [n] \setminus S$. On each query $i \in [n] \setminus S$, the challenger responds with the key $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}(\mathsf{msk}, i)$. Without loss of generality, we assume the adversary queries each index i at most once.
- **Challenge phase:** Algorithm A now outputs two messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}_\lambda$ and a set $S^\star \subseteq S$. The challenger samples a bit $b \xleftarrow{R} \{0,1\}$ replies to A with the ciphertext $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}_b, S^\star)$.
- **Output phase:** Algorithm A outputs a bit $b' \in \{0,1\}$. The experiment outputs 1 if $b' = b$ and outputs 0 otherwise.

Optimal broadcast encryption. In an optimal broadcast encryption scheme, we additionally require that all of the scheme parameters (i.e., the public parameters pp and the decryption keys sk) be short. We give the formal definition below:

Definition 5.3 (Optimal Broadcast Encryption)**.** A broadcast encryption scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies optimal succinctness if it satisfies the following property (in addition to the properties in Definition 5.1):

- **Succinct public key and secret keys:** There exists a polynomial p such that for all $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $i \in [n]$, $(\mathsf{pp}, \mathsf{msk}) \in \mathsf{Supp}\left(\mathsf{Setup}(1^\lambda, n)\right)$ and $\mathsf{sk}_i \in \mathsf{Supp}\left(\mathsf{KeyGen}(\mathsf{msk}, i)\right)$, it holds that

$$|\mathsf{pp}| \leq p(\lambda, \log n) \quad \text{and} \quad |\mathsf{sk}_i| \leq p(\lambda, \log n).$$

5.1 Trapdoor Proof Generator

As outlined in Section 1.1, the main building block we use to construct our optimal broadcast encryption scheme is a trapdoor proof generator. A trapdoor proof generator is defined over a family of sets $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$. There is a public verification algorithm that verifies proofs associated with elements $x \in \mathcal{X}$. The trapdoor proof generator consists of two setup algorithms: a "normal" setup and an "alternative" setup:

- **Normal mode:** The normal setup algorithm outputs a set of public parameters pp (used for verifying proofs) along with a trapdoor td (used for generating proofs). Specifically, using the trapdoor, one can generate proofs π for random instances $x \xleftarrow{R} \mathcal{X}_\lambda$.
- **Alternative mode:** The alternative setup algorithm outputs a set of public parameters $\mathsf{pp}^\star$ that are computationally indistinguishable from that in the normal mode. However, $\mathsf{pp}^\star$ effectively partitions $\mathcal{X}$ into two disjoint sets: (1) a dense subset $\mathcal{X}_T \subset \mathcal{X}$; and (2) a sparse pseudorandom subset $\mathcal{X}_F \subset \mathcal{X}$. There are two sampling algorithms: (1) a $\mathsf{SampleTrue}$ algorithm that jointly samples an instance $x \in \mathcal{X}_T$ together with an accepting proof π for x (whose distributions are computationally indistinguishable from sampling $x \xleftarrow{R} \mathcal{X}$ and using a normal-mode trapdoor to sample the proof); and (2) a $\mathsf{SampleFalse}$ algorithm that samples an instance $x \in \mathcal{X}_F$ for which there does *not* exist any proof π that verifies with respect to $\mathsf{pp}^\star$. Finally, in the alternative mode, there is a trapdoor that allows one to efficiently decide membership in $\mathcal{X}_T$ and $\mathcal{X}_F$.

We formalize this notion below:

Definition 5.4 (Trapdoor Proof Generator)**.** Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ be a sequence of efficiently samplable and recognizable sets. A *trapdoor proof generator (TPG) for* $\mathcal{X}$ consists of a tuple of polynomial-time algorithms $(\mathsf{Setup}, \mathsf{CreateProof}, \mathsf{Verify}, \mathsf{SetupAlt}, \mathsf{SampleTrue}, \mathsf{SampleFalse}, \mathsf{TDDecide})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})$: A probabilistic algorithm that on input a security parameter λ, outputs public parameters pp and a trapdoor td. We implicitly assume that pp includes 1^λ and td includes pp.
- $\mathsf{CreateProof}(\mathsf{td}, x) \to \pi$: A deterministic algorithm that on input a trapdoor td and an instance $x \in \mathcal{X}_\lambda$, outputs a proof π.
- $\mathsf{Verify}(\mathsf{pp}, x, \pi) \to b$: A deterministic algorithm that on input the public parameters pp, an instance x, and a proof π, outputs a bit $b \in \{0, 1\}$.
- $\mathsf{SetupAlt}(1^\lambda) \to (\mathsf{pp}, \mathsf{td})$: A probabilistic algorithm that on input a security parameter λ, outputs public parameters pp and a trapdoor td. We assume for simplicity that pp includes 1^λ and td includes pp.
- $\mathsf{SampleTrue}(\mathsf{pp}) \to (x, \pi)$: A probabilistic algorithm that on input the public parameters pp, outputs an instance $x \in \mathcal{X}_\lambda$ and a proof π.
- $\mathsf{SampleFalse}(\mathsf{pp}) \to x$: A probabilistic algorithm that on input the public parameters pp, outputs a instance $x \in \mathcal{X}_\lambda$.
- $\mathsf{TDDecide}(\mathsf{td}, x) \to b$: A deterministic algorithm that on input a trapdoor td and an instance $x \in \mathcal{X}_\lambda$, outputs a bit b.

We require that (Setup, CreateProof, Verify, SetupAlt, SampleTrue, SampleFalse, TDDecide) satisfy the following properties:

- **Correctness:** There exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{ll} (\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) & \\ x \xleftarrow{R} \mathcal{X}_\lambda & : \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 \\ \pi \leftarrow \mathsf{CreateProof}(\mathsf{td}, x) & \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

- **Mode indistinguishability:** For all stateful PPT algorithms A, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\mathsf{Expt}_A^{\mathsf{MI}}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

  where $\mathsf{Expt}_A^{\mathsf{MI}}(\lambda)$ is defined via the following security game between the adversary A and a challenger on common input 1^λ:

  - **Setup phase:** The challenger samples a bit $b \xleftarrow{R} \{0, 1\}$. If $b = 0$, the challenger samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, and if $b = 1$, the challenger samples $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda)$. The challenger sends pp to A.
  - **Query phase:** Algorithm A makes adaptive queries for either true or false instances. The challenger responds to the queries as follows:
    - **True-instance query:** If $b = 0$, the challenger samples $x \xleftarrow{R} \mathcal{X}_\lambda$ and $\pi \leftarrow \mathsf{CreateProof}(\mathsf{td}, x)$ and replies with (x, π). If $b = 1$, the challenge responds with $(x, \pi) \leftarrow \mathsf{SampleTrue}(\mathsf{pp})$.
    - **False-instance query:** The challenger replies with $x \xleftarrow{R} \mathcal{X}_\lambda$ if $b = 0$ and $x \leftarrow \mathsf{SampleFalse}(\mathsf{pp})$ if $b = 1$.
  - **Output phase:** Algorithm A computes a guess $b' \in \{0, 1\}$ for b, and the experiment outputs 1 if $b = b'$ and 0 otherwise.

- **Trapdoor decidability:** The following hold regarding the algorithm TDDecide:
  - **Accepting true instances:** For all (possibly unbounded) algorithms A, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{ll} (\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda) & \\ (x, \pi) \leftarrow A(\mathsf{pp}) & : \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 \wedge \mathsf{TDDecide}(\mathsf{td}, x) \neq 1 \end{array}\right] \leq \mathsf{negl}(\lambda).$$

  - **Rejecting false instances:** There exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{ll} (\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda) & \\ x \leftarrow \mathsf{SampleFalse}(\mathsf{pp}) & : \mathsf{TDDecide}(\mathsf{td}, x) = 0 \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

5.2 Trapdoor Proof Generator Building Blocks

In Section 5.3, we show how to construct a trapdoor proof generator from a public-key encryption scheme with pseudorandom ciphertexts together with a computational non-interactive zero-knowledge (NIZK) proof system for NP. We start by recalling these notions below:

Definition 5.5 (PKE with Pseudorandom Ciphertexts)**.** Let $\ell = \ell(\lambda)$ be a length parameter. We say that a public-key encryption scheme (KeyGen, Enc, Dec) has *pseudorandom ciphertexts of length* ℓ if for all stateful PPT algorithms A, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{ll} (\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) & \\ \mathsf{msg} \leftarrow A(\mathsf{pk}) & \\ \mathsf{ct}_0^\star \leftarrow \mathsf{Enc}(\mathsf{pk},\mathsf{msg}) & : \quad b' = b \\ \mathsf{ct}_1^\star \xleftarrow{R} \{0,1\}^{\ell(\lambda)} & \\ b \xleftarrow{R} \{0,1\} & \\ b' \leftarrow A(\mathsf{ct}_b^\star) & \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda).$$

Definition 5.6 (Non-Interactive Zero Knowledge)**.** A computational non-interactive zero knowledge (NIZK) proof system for an NP language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda\in\mathbb{N}}$ with witness relation $\mathbf{R}_\mathcal{L}$ consists of polynomial-time algorithms (Setup, Prove, Verify) with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$: A probabilistic algorithm that on input a security parameter λ, outputs a common reference string crs. We implicitly assume that crs includes 1^λ.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: A deterministic algorithm that on input a common reference string crs, an instance x, and a witness w, outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: A deterministic algorithm that on input a common reference string crs, an instance x, and a proof π, outputs a bit $b \in \{0,1\}$.

We require that (Setup, Prove, Verify) satisfy the following properties:

- **Completeness:** There exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, tuple $(1^\lambda, x, w) \in \mathbf{R}_\mathcal{L}$, it holds that
$$\Pr\left[\begin{array}{ll} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) & \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) & : \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$
- **Statistical soundness:** For all (possibly unbounded) algorithms A, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that
$$\Pr\left[\begin{array}{ll} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) & \\ (x,\pi) \leftarrow A(\mathsf{crs}) & : \quad x \notin \mathcal{L}_\lambda \wedge \mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda).$$
- **(Multi-theorem) computational zero knowledge:** There exist PPT algorithms (SimSetup, SimProve) such that for all stateful PPT adversaries A, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, it holds that
$$\Pr\left[\begin{array}{ll} b \xleftarrow{R} \{0,1\} & \\ \mathsf{crs}_0 \leftarrow \mathsf{Setup}(1^\lambda) & \\ (\mathsf{crs}_1, \sigma) \leftarrow \mathsf{SimSetup}(1^\lambda) & : \quad b' = b \\ b' \leftarrow A^{O_b(\cdot,\cdot)}(\mathsf{crs}_b) & \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda),$$
where $O_0(\cdot,\cdot)$ and $O_1(\cdot,\cdot)$ are defined as follows:
  - $O_0(x,w)$ outputs $\mathsf{Prove}(\mathsf{crs}_0, x, w)$ if $(1^\lambda, x, w) \in \mathbf{R}_\mathcal{L}$ and $\perp$ otherwise.
  - $O_1(x,w)$ outputs $\mathsf{SimProve}(\mathsf{crs}_1, x, \sigma)$ if $(1^\lambda, x, w) \in \mathbf{R}_\mathcal{L}$ and $\perp$ otherwise.

Instantiations. Computational NIZK proofs are known from standard pairing-based assumptions [GOS06, CHK03] and lattice-based assumptions [PS19]. Public-key encryption with pseudorandom ciphertexts can be constructed from standard assumptions over groups (with or without a pairing) [Gam84] and from lattice-based assumptions [Reg05].

5.3 Constructing a Trapdoor Proof Generator

We now show how to construct a trapdoor proof generator from a public-key encryption scheme with pseudorandom ciphertexts in conjunction with a computational NIZK proof system for NP. We remark that this is just one way to construct such a primitive. However, we believe that this primitive may be of more general interest, and there are likely other constructions from potentially simpler building blocks. We leave it as an interesting open question to explore such directions.

Construction 5.7 (Trapdoor Proof Generator)**.** Let $\ell = \ell(\lambda)$ be a polynomial and $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ be a sequence of sets where $\mathcal{X}_\lambda = \{0,1\}^{\ell(\lambda)}$. Our construction relies on the following building blocks:

- Let PKE = (PKE.KeyGen, PKE.Enc, PKE.Dec) be a semantically-secure public-key bit encryption scheme with pseudorandom ciphertexts of length ℓ. Without loss of generality, we assume that the encryption randomness to PKE is λ-bits long.[8]
- Let NIZK = (NIZK.Setup, NIZK.Prove, NIZK.Verify, NIZK.SimSetup, NIZK.SimProve) be a computational non-interactive zero knowledge proof for the language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda\in\mathbb{N}}$ defined by the relation $\mathbf{R}_\mathcal{L}$ as follows. Instances of the language $\mathcal{L}_\lambda$ are of the form $(\mathsf{pk}, \mathsf{ct})$ and the relation $\mathbf{R}_\mathcal{L}$ is given by

$$\mathbf{R}_\mathcal{L}\left(1^\lambda, (\mathsf{pk}, \mathsf{ct}), r\right) = 1 \Leftrightarrow \mathsf{PKE.Enc}(\mathsf{pk}, 1; r) = \mathsf{ct}.$$

We construct a trapdoor proof generator TPG = (Setup, CreateProof, Verify, SetupAlt, SampleTrue, SampleFalse, TDDecide) for $\mathcal{X}$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, the setup algorithm samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, $(\mathsf{crs}_{\mathsf{NIZK}}, \sigma) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda)$, and output the public parameters $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}})$ and a trapdoor $\mathsf{td} = \sigma$. Note that we implicitly assume that pp contains 1^λ and n, and td contains pp.
- $\mathsf{CreateProof}(\mathsf{td}, x)$: On input a trapdoor $\mathsf{td} = \sigma$ with associated public parameters $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}})$ and an instance $x \in \mathcal{X}_\lambda$, the algorithm generates the associated proof $\pi_{\mathsf{NIZK}} = \mathsf{NIZK.SimProve}(\mathsf{crs}_{\mathsf{NIZK}}, (\mathsf{pk}, x), \sigma)$.
- $\mathsf{Verify}(\mathsf{pp}, x, \pi)$: On input the public parameters $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}})$, an instance $x \in \mathcal{X}_\lambda$, and a proof π, the verification algorithm outputs $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, (\mathsf{pk}, x), \pi)$.
- $\mathsf{SetupAlt}(1^\lambda)$: On input the security parameter λ, the alternative setup algorithm starts by sampling $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, $\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$, and outputs the public parameters $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}})$ and trapdoor $\mathsf{td} = \mathsf{sk}$. Recall that we implicitly assume that pp contains 1^λ and n and td contains pp.
- $\mathsf{SampleTrue}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}})$, the true-instance sampling algorithm samples $r \xleftarrow{R} \{0,1\}^\lambda$, computes $\mathsf{ct} = \mathsf{PKEEnc}(\mathsf{pk}, 1; r)$, and outputs the instance $x = \mathsf{ct}$ and proof $\pi = \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK}}, (\mathsf{pk}, \mathsf{ct}), r)$.
- $\mathsf{SampleFalse}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}})$, the false-instance sampling algorithm computes $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 0)$, and outputs the instance $x = \mathsf{ct}$.
- $\mathsf{TDDecide}(\mathsf{td}, x)$: On input the trapdoor $\mathsf{td} = \mathsf{sk}$ and an instance $x \in \mathcal{X}_\lambda$, the trapdoor decision algorithm outputs 1 if $\mathsf{PKE.Dec}(\mathsf{sk}, x) = 1$ and 0 otherwise.

Correctness and security analysis. We now show that TPG from Construction 5.7 satisfies each of the correctness and security properties in Definition 5.4.

Theorem 5.8 (Correctness)**.** *Assuming* PKE *has pseudorandom ciphertexts and* NIZK *satisfies completeness and computational zero knowledge, then Construction 5.7 satisfies correctness.*

[8] If PKE.Enc requires more than λ-bits of randomness, we can first stretch the randomness using a pseudorandom generator.

Proof. We consider the following sequence of hybrid experiments, defined as a function of the security parameter $\lambda \in \mathbb{N}$.

- $H_0(\lambda)$: This experiment corresponds to the correctness criteria. Namely, $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, $x \xleftarrow{R} \mathcal{X}_\lambda$, $\pi = \mathsf{CreateProof}(\mathsf{td}, x)$, and then the experiment output the results of $\mathsf{Verify}(\mathsf{pp}, x, \pi)$.
- $H_1(\lambda)$: This hybrid is the same as H_0, except the challenger instead computes $x \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$, where pk is the public key specified in the public parameters pp, and $r \xleftarrow{R} \{0,1\}^\lambda$ is freshly-sampled randomness.
- $H_2(\lambda)$: In this hybrid, we now compute $\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and $\pi_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK}}, (\mathsf{pk}, x), r)$.

It remains to show that there exists a negligible function negl such that $\Pr[H_0(\lambda) = 1] \geq 1 - \mathsf{negl}(\lambda)$. We now argue that there exist negligible functions μ, μ' such that $\Pr[H_2(\lambda) = 1] - \Pr[H_0(\lambda) = 1] \leq \mu(\lambda)$ and moreover, that $\Pr[H_2(\lambda) = 1] \geq 1 - \mu'(\lambda)$. This implies that $\Pr[H_0(\lambda) = 1] \geq 1 - \mathsf{negl}(\lambda)$ for some negligible function negl, as required for correctness.

Claim 5.9. *Assuming* PKE *has pseudorandom ciphertexts, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, $\Pr[H_1(\lambda) = 1] - \Pr[H_0(\lambda) = 1] \leq \mu(\lambda)$.*

Proof. Suppose by way of contradiction that there exists a polynomial q such that $\Pr[H_1(\lambda) = 1] - \Pr[H_0(\lambda) = 1] > 1/q(\lambda)$. We construct a PPT adversary B that breaks that pseudorandom ciphertext property of PKE:

1. On input a public key pk, B sends the message $\mathsf{msg} = 1$.
2. The challenger samples a bit $b \xleftarrow{R} \{0,1\}$ and sends the challenge ciphertext $\mathsf{ct}_b^\star$, which is either equal to $\mathsf{ct}_0^\star \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 1)$ or $\mathsf{ct}_1^\star \xleftarrow{R} \{0,1\}^{\ell(\lambda)}$, where ℓ is the length of ciphertexts for PKE.
3. Algorithm B samples $(\mathsf{crs}_{\mathsf{NIZK}}, \sigma) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda)$ and $\pi \leftarrow \mathsf{NIZK.SimProve}(\mathsf{crs}_{\mathsf{NIZK}}, (\mathsf{pk}, \mathsf{ct}_b^\star), \sigma)$.
4. Algorithm B then computes $b' = \mathsf{Verify}(\mathsf{pp}, (\mathsf{pk}, \mathsf{ct}_b^\star))$ and outputs $1 - b'$ as its guess for b.

Algorithm B runs in polynomial time by construction. To analyze the success probability of B, note that if the challenger chooses the bit $b = 0$, then the output of B corresponds to $1 - H_1(\lambda)$, so B wins if $H_1(\lambda)$ outputs 1. If $b = 1$, then the output of B corresponds to $1 - H_0(\lambda)$, so B wins if $H_0(\lambda)$ outputs 0. Therefore, B succeeds with probability

$$\frac{1}{2} \cdot \Pr[H_1(\lambda) = 1] + \frac{1}{2} \cdot \Pr[H_0(\lambda) = 0] = \frac{1}{2} + \frac{1}{2} \cdot (\Pr[H_0(\lambda) = 1] - \Pr[H_1(\lambda) = 1]) .$$

As $\Pr[H_1(\lambda) = 1] - \Pr[H_0(\lambda) = 1] > 1/q(\lambda)$ by assumption, this violates the pseudorandom ciphertext property of PKE, as required. □

Claim 5.10. *Assuming* NIZK *satisfies computational zero knowledge, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, $\Pr[H_2(\lambda) = 1] - \Pr[H_1(\lambda) = 1] \leq \mu(\lambda)$.*

Proof. Suppose by way of contradiction, there exists a polynomial q such that $\Pr[H_2(\lambda) = 1] - \Pr[H_1(\lambda) = 1] > q(\lambda)$. We construct a PPT adversary B that breaks the computational zero knowledge property of NIZK:

1. The NIZK challenger starts by sampling a random bit $b \xleftarrow{R} \{0,1\}$ and sends crs_b, which is either equal to $\mathsf{crs}_0 \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ or $(\mathsf{crs}_1, \sigma) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda)$.
2. On input crs_b, algorithm B samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and sets $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_b)$. Algorithm B computes $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 1; r)$ for randomness $r \xleftarrow{R} \{0,1\}^\lambda$ and queries its oracle on the instance $(\mathsf{pk}, \mathsf{ct})$ with witness r.
   - If $b = 0$, the oracle returns $\pi_0 \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_0, (\mathsf{pk}, \mathsf{ct}), r)$.
   - If $b = 1$, the oracle returns $\pi_1 \leftarrow \mathsf{SimProve}(\mathsf{crs}_1, (\mathsf{pk}, \mathsf{ct}), \sigma)$.
3. Algorithm B computes $b' = \mathsf{Verify}(\mathsf{pp}, (\mathsf{pk}, \mathsf{ct}), \pi_b)$ and outputs $1 - b'$ as its guess for b.

Algorithm B runs in polynomial time by construction. To analyze the success probability of B, note that if the challenger chooses the bit $b = 0$, then the output of B corresponds to $1 - H_2(\lambda)$, so B wins if $H_2(\lambda)$ outputs 1. If $b = 1$, the output of B corresponds to $1 - H_1(\lambda)$, so B wins if $H_1(\lambda)$ outputs 0. Therefore, the success probability of B is given by

$$\frac{1}{2} \cdot \Pr[H_2(\lambda) = 1] + \frac{1}{2} \cdot \Pr[H_1(\lambda) = 0] = \frac{1}{2} + \frac{1}{2} \cdot (\Pr[H_2(\lambda) = 1] - \Pr[H_1(\lambda) = 1]) .$$

As $\Pr[H_2(\lambda) = 1] - \Pr[H_1(\lambda) = 1] > 1/q(\lambda)$ by assumption, this violates the computational zero knowledge property of NIZK, as required. □

Claim 5.11. *Assuming* NIZK *is complete, there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$, it holds that* $\Pr[H_2(\lambda) = 1] \geq 1 - \mu(\lambda)$.

Proof. By construction, $H_2(\lambda)$ corresponds to the completeness experiment for NIZK for the language $\mathcal{L}_\lambda$ with instance $(\mathsf{pk}, \mathsf{ct})$ for $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 1)$ and witness r. Thus, $(1^\lambda, (\mathsf{pk}, \mathsf{ct}), r) \in \mathbf{R}_{\mathcal{L}}$, so by completeness of NIZK, there exists a negligible function μ such that $\Pr[H_2(\lambda) = 1] \geq 1 - \mu(\lambda)$. □

This proof now follows from Claims 5.9 to 5.11. □

Theorem 5.12 (Mode Indistinguishability)**.** *Assuming* PKE *has pseudorandom ciphertexts and* NIZK *satisfies computational zero knowledge, then Construction 5.7 satisfies mode indistinguishability.*

Proof. Suppose by way of contradiction that mode indistinguishability does not hold. Then there exists a stateful PPT algorithm A and a polynomial q such that for infinitely many $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\mathsf{Expt}_A^{\mathsf{MI}}(\lambda) = 1\right] > 1/2 + 1/q(\lambda).$$

We now define the following sequence of hybrid experiments, each parameterized by the security parameter $\lambda \in \mathbb{N}$:

- $H_0(\lambda)$: This experiment corresponds to the mode indistinguishability experiment where b is fixed to 0, and the hybrid outputs the final value b' output by A.
- $H_1(\lambda)$: This hybrid is the same as H_0 except that the challenger samples $x \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 1)$ for true-instance queries, and $x \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, 0)$ for false-instance queries.
- $H_2(\lambda)$: This hybrid corresponds to the mode indistinguishability experiment where b is fixed to 1. Specifically, compared to hybrid $H_1(\lambda)$, the challenger now samples $(x, \pi) \leftarrow \mathsf{SampleTrue}(\mathsf{pp})$ for true-instance queries.

For $i \in \{1, 2\}$, we define $\delta_i(\lambda) = \Pr[H_i(\lambda) = 1] - \Pr[H_{i-1}(\lambda) = 1]$. Then, the success probability of A can be written as

$$\begin{aligned}
\Pr[A \text{ wins}] &= \frac{1}{2} \cdot \Pr[H_0(\lambda) = 0] + \frac{1}{2} \cdot \Pr[H_2(\lambda) = 1] \\
&= \frac{1}{2} \cdot \Pr[H_0(\lambda) = 0] + \frac{1}{2} \cdot (\Pr[H_0(\lambda) = 1] + \delta_1(\lambda) + \delta_2(\lambda)) \\
&= \frac{1}{2} + \frac{1}{2} \cdot (\delta_1(\lambda) + \delta_2(\lambda)) .
\end{aligned}$$

It follows from our assumption that, for infinitely many $\lambda \in \mathbb{N}$, $\delta_1(\lambda) + \delta_2(\lambda) > 2/q(\lambda)$. Thus, either $\delta_1(\lambda) > 1/q(\lambda)$ or $\delta_2(\lambda) > 1/q(\lambda)$. We show in the subsequent claims that if either case holds, it must contradict one the of assumptions in the theorem statement.

Claim 5.13. *If $\delta_1(\lambda) > 1/q(\lambda)$ for infinitely many $\lambda \in \mathbb{N}$, then* PKE *does not have pseudorandom ciphertexts.*

Proof. Let $Q = Q(\lambda)$ be polynomial upper bounding the number of queries that A makes in the mode indistinguishability experiment. We construct an adversary B that breaks the pseudorandom ciphertext property of PKE:

1. At the beginning of the security game, algorithm B receives a public key pk as input from the PKE challenger.

2. Algorithm B samples $(\mathsf{crs}_{\mathsf{NIZK}}, \sigma) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda)$ and sets $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}})$, $\mathsf{td} = \sigma$. It gives pp to A.

3. Algorithm B samples an index $i^\star \xleftarrow{R} [0, Q(\lambda) - 1]$. Whenever A makes its ith query, algorithm B responds as follows.

   - If $i < Q(\lambda) - i^\star$, algorithm B responds as in hybrid $H_0(\lambda)$ by sampling a random $x \xleftarrow{R} \mathcal{X}_\lambda$ and additionally providing a proof $\pi = \mathsf{CreateProof}(\mathsf{td}, x)$ on queries for true instances.
   - If $i = Q(\lambda) - i^\star$, algorithm B sends the message $\mathsf{msg} = 1$ to the PKE challenger if A made a query for a true instance and the message $\mathsf{msg} = 0$ for a false query. The challenger replies with a challenge ciphertext $\mathsf{ct}^\star$ which algorithm B forwards to A. In addition, if the query was for a true instance, algorithm B also gives a proof $\pi = \mathsf{CreateProof}(\mathsf{td}, x)$ to A. Here, the challenge ciphertext is computed as follows. The challenger samples a bit $\hat{b} \xleftarrow{R} \{0,1\}$ and sets the challenge ciphertext $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$ if $\hat{b} = 0$ and $\mathsf{ct}^\star \xleftarrow{R} \{0,1\}^{\ell(\lambda)}$ if $\hat{b} = 1$.
   - If $i > Q(\lambda) - i^\star$, B sets $x \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$ on false-instance queries and $x \leftarrow \mathsf{Enc}(\mathsf{pk}, 1)$ on true-instance queries. On true-instance queries, it also provides a proof $\pi = \mathsf{CreateProof}(\mathsf{td}, x)$ for true-instance queries.

4. A computes a guess b' for b and B outputs the guess $\hat{b}' = 1 - b'$.

It is clear that A runs in polynomial-time if B does. To analyze the success probability of B, we define the following sub-hybrid experiments.

- $H_{0,i}(\lambda)$: In this hybrid, B responds to the first $Q(\lambda) - i$ queries according to H_0 and the final i queries according to H_1.

Note that $H_{0,0}(\lambda)$ corresponds to $H_0(\lambda)$ and $H_{0,Q(\lambda)}(\lambda)$ corresponds to $H_1(\lambda)$. For a fixed index $i^\star = i$, we observe that if $\hat{b} = 1$, then A's response is distributed as in hybrid $H_{0,i}$, and if $\hat{b} = 0$, then A's response is distributed as in hybrid $H_{0,i+1}$. It follows that B succeeds with probability

$$
\begin{aligned}
&\sum_{i=0}^{Q(\lambda)-1} \Pr[i^\star = i] \cdot \Pr[\hat{b}' = b' \mid i^\star = i] \\
&= \frac{1}{Q(\lambda)} \cdot \sum_{i=0}^{Q(\lambda)-1} \left(\frac{1}{2} \cdot \Pr[H_{0,i}(\lambda) = 0] + \frac{1}{2} \cdot \Pr[H_{0,i+1}(\lambda) = 1] \right) \\
&= \frac{1}{2} + \frac{1}{2Q(\lambda)} \cdot \sum_{i=0}^{Q(\lambda)-1} \left(\Pr[H_{0,i+1}(\lambda) = 1] - \Pr[H_{0,i}(\lambda) = 1] \right) \\
&= \frac{1}{2} + \frac{1}{2Q(\lambda)} \cdot \left(\Pr[H_1(\lambda) = 1] - \Pr[H_0(\lambda) = 1] \right).
\end{aligned}
$$

By our assumption, this is at least $1/2 + 1/(2 \cdot Q(\lambda) \cdot q(\lambda))$. Since Q and q are both polynomially bounded, this contradicts the psuedorandom ciphertext property of PKE, as required. □

Claim 5.14. *If $\delta_2(\lambda) > 1/q(\lambda)$ for infinitely many $\lambda \in \mathbb{N}$, then* NIZK *does not satisfy multi-theorem computational zero knowledge.*

Proof. We construct an adversary B that breaks the computational zero knowledge property of NIZK:

1. The NIZK challenger samples a bit $\hat{b} \xleftarrow{R} \{0,1\}$. If $\hat{b} = 0$, it sends $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$, and if $\hat{b} = 1$, it sends crs where $(\mathsf{crs}, \sigma) \leftarrow \mathsf{NIZK.SimSetup}(1^\lambda)$.

2. Algorithm B computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, sets $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs})$, and sends pp to A.

3. Whenever A makes a true-instance query, algorithm B samples $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, 1; r)$ for $r \xleftarrow{R} \{0,1\}^\lambda$ and queries the NIZK proof oracle on $((\mathsf{pk}, \mathsf{ct}), r)$ to get a proof π. Algorithm B responds to A with (ct, π). Whenever A makes a false-instance query, algorithm B samples $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$ and returns the instance ct.

4. A outputs a guess b' for b, and B outputs the same guess $\hat{b}' = b'$ for $\hat{b}$.

Algorithm B clearly runs in polynomial-time if A does. For correctness, observe that if $\hat{b} = 0$, then the view of A is identically distributed as in hybrid $H_1(\lambda)$, and if $\hat{b} = 1$, then the view of A is identically distributed as in hybrid $H_2(\lambda)$. So, B succeeds with probability

$$\Pr[B \text{ wins}] = \frac{1}{2} \cdot \Pr[H_1(\lambda) = 0] + \frac{1}{2} \cdot \Pr[H_2(\lambda) = 1] = \frac{1}{2} + \frac{1}{2} \cdot (\Pr[H_2(\lambda) = 1] - \Pr[H_1(\lambda) = 1]) .$$

As $\Pr[H_2(\lambda) = 1] - \Pr[H_1(\lambda) = 1] = \delta_2(\lambda) > 1/q(\lambda)$ by assumption, this violates the computational zero knowledge property of NIZK, as required. □

The proof now follows via Claims 5.13 and 5.14. □

Theorem 5.15 (Decidability in Trapdoor Mode)**.** *Assuming* PKE *is correct and* NIZK *satisfies completeness and is statistically sound, then Construction 5.7 satisfies decidability in trapdoor mode.*

Proof. For any $\lambda \in \mathbb{N}$, let $(\mathsf{pp}, \mathsf{td}) \leftarrow \mathsf{SetupAlt}(1^\lambda)$. Recall that $\mathsf{pp} = (\mathsf{pk}, \mathsf{crs}_{\mathsf{NIZK}})$ and $\mathsf{td} = \mathsf{sk}$, where $\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$. We show the two properties separately:

- **Accepting true instances:** Suppose that there exists a proof π such that $\mathsf{Verify}(\mathsf{pp}, x, \pi) = 1$. By construction, this implies that $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, (\mathsf{pk}, x), \pi) = 1$. By statistical soundness of NIZK, there exists a negligible function negl such that that $(\mathsf{pk}, x) \in \mathcal{L}_\lambda$ with probability at least $1 - \mathsf{negl}(\lambda)$. If $(\mathsf{pk}, x) \in \mathcal{L}_\lambda$, then there exists $r \in \{0,1\}^\lambda$ such that $\mathsf{PKE.Enc}(\mathsf{pk}, 1; r) = x$. By correctness of PKE, this means that $\mathsf{PKE.Dec}(\mathsf{sk}, x) = 1$. In this case, $\mathsf{TDDecide}(\mathsf{pp}, x)$ outputs 1 with probability at least $1 - \mathsf{negl}(\lambda)$, and the claim follows.
- **Rejecting false instances:** Suppose $x \leftarrow \mathsf{SampleFalse}(\mathsf{pp})$. This implies that $x \in \mathsf{Supp}(\mathsf{PKE.Enc}(\mathsf{pk}, 0))$, so $\mathsf{PKE.Dec}(\mathsf{sk}, x) = 0$. That immediately implies that $\mathsf{TDDecide}(\mathsf{td}, x)$ outputs 0, as required. □

5.4 Constructing Optimal Broadcast Encryption

We are now ready to construct an optimal broadcast encryption scheme from witness encryption and function-binding hash functions. Our construction follows the following template:

- We first construct a (non-optimal) broadcast encryption scheme with short ciphertexts and secret keys, but *long* public parameters. In particular, the public parameters contains a *uniform* random string of length n, where n is the number of users.
- To obtain an optimal broadcast encryption scheme in the random oracle model, we instantiate the above construction, but now derive the long random string in the public parameters as the output of a random oracle. The rest of the scheme is unchanged. The random oracle is only used to "compress" the long public parameters.

We now describe our broadcast encryption scheme:

Construction 5.16 (Broadcast Encryption with Long Public Key)**.** Let $s = s(\lambda)$, $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, $d = d(\lambda)$, and $r = r(\lambda)$ be polynomials, and let $m = m(\lambda)$ be any function. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ be a message space of length m. Let $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of strings of size ℓ_{blk} that is efficiently-recognizable. Our broadcast encryption scheme relies on the following primitives:

- First, let $\mathsf{TPG} = (\mathsf{TPG.Setup}, \mathsf{TPG.CreateProof}, \mathsf{TPG.Verify}, \mathsf{TPG.SetupAlt}, \mathsf{TPG.SampleTrue}, \mathsf{TPG.SampleFalse}, \mathsf{TPG.TDDecide})$ be a trapdoor proof generator for $\mathcal{X}$. For all security parameters $\lambda \in \mathbb{N}$ and parameters $(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{td}_{\mathsf{TPG}}) \in \mathsf{Supp}\left(\mathsf{TPG.SetupAlt}(1^\lambda)\right)$, we require that $\mathsf{TPG.TDDecide}(\mathsf{td}_{\mathsf{TPG}}, \cdot)$ can be computed by a circuit of size $s(\lambda)$ and depth $d(\lambda)$.
- Let $\mathsf{FBH} = (\mathsf{FBH.Setup}, \mathsf{FBH.SetupBinding}, \mathsf{FBH.Hash}.\mathsf{FBH.ProveOpen}, \mathsf{FBH.VerOpen})$ be a function-binding hash for the class $\mathcal{F}$ of disjunctions of block functions for input length ℓ_{blk}, size s, and depth d (Definition 3.3).

- Let WE be a witness encryption scheme for the language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda\in\mathbb{N}}$ defined by the relation $\mathbf{R}_\mathcal{L}$ as follows. An instance of the language $\mathcal{L}_\lambda$ has the form $(\mathsf{hk}, \mathsf{pp}_{\mathsf{TPG}}, \mathsf{dig})$ and the relation $\mathbf{R}_\mathcal{L}$ is given by

$$\begin{aligned}&\mathbf{R}_\mathcal{L}(1^\lambda, (\mathsf{hk}, \mathsf{pp}_{\mathsf{TPG}}, \mathsf{dig}), (i, \mathsf{hinp}_i, \pi_{\mathsf{TPG}}, \pi_{\mathsf{FBH}})) = 1\\ &\Leftrightarrow \mathsf{TPG.Verify}(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hinp}_i, \pi_{\mathsf{TPG}}) = 1 \ \wedge \mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{i\}, \{(i, \mathsf{hinp}_i)\}, \pi_{\mathsf{FBH}}) = 1.\end{aligned}$$

We construct an optimal broadcast encryption scheme $\mathsf{BE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^n)$: On input the security parameter λ and a bound on the number of users n, the setup algorithm first samples $(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{td}_{\mathsf{TPG}}) \leftarrow \mathsf{TPG.Setup}(1^\lambda)$ and $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, n)$. Then, for each $i \in [n]$, it samples $x_i \xleftarrow{R} \mathcal{X}_\lambda$. It now outputs the public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hk}, (x_1, \ldots, x_n))$ and master secret key $\mathsf{msk} = (\mathsf{td}_{\mathsf{TPG}}, (x_1, \ldots, x_n))$.
- $\mathsf{KeyGen}(\mathsf{msk}, i)$: On input the master secret key $\mathsf{msk} = (\mathsf{td}_{\mathsf{TPG}}, (x_1, \ldots, x_n))$, the key-generation algorithm computes the proof $\pi_{\mathsf{TPG}} = \mathsf{TPG.CreateProof}(\mathsf{td}_{\mathsf{TPG}}, x_i)$, and outputs the secret key $\mathsf{sk}_i = \pi_{\mathsf{TPG}}$.
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, S)$: On input the public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hk}, (x_1, \ldots, x_n))$, a message $\mathsf{msg} \in \mathcal{M}_\lambda$, and a set $S = \{i_1, \ldots, i_k\} \subseteq [n]$ for some $k \le n$ where $i_1 < \cdots < i_k$, the encryption algorithm computes the digest $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (x_{i_1}, \ldots, x_{i_k}))$ and outputs $\mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathsf{msg}, (\mathsf{hk}, \mathsf{pp}_{\mathsf{TPG}}, \mathsf{dig}))$.
- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), S)$: On input the public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hk}, (x_1, \ldots, x_n))$, a ciphertext ct, an index/secret-key pair (j, sk_j), and a set $S = \{i_1, \ldots, i_k\} \subseteq [n]$ for some $k \le n$ where $i_1 < \cdots < i_k$, the decryption algorithm computes $\pi_{\mathsf{FBH}} = \mathsf{FBH.ProveOpen}(\mathsf{hk}, (x_{i_1}, \ldots, x_{i_k}), \{j'\})$, where $j' \in [k]$ is the index such that $j = i_{j'} \in S$. It then outputs $\mathsf{msg} = \mathsf{WE.Dec}(1^\lambda, \mathsf{ct}, (j', x_j, \mathsf{sk}_j, \pi_{\mathsf{FBH}}))$.

Correctness and efficiency. We first that the broadcast encryption scheme in Construction 5.16 satisfies correctness and has succinct ciphertexts and secret keys.

Theorem 5.17 (Correctness and Efficiency)**.** *Assuming* TPG *and* WE *are correct and efficient, and* FBH *satisfies completeness and efficiency, then Construction 5.16 is correct and has succinct ciphertexts and secret keys.*

Proof. We consider each property separately:

- **Succinct ciphertexts and secret keys:** Take any $\lambda \in \mathbb{N}$, $n \in \mathbb{N}$, $\mathsf{msg} \in \mathcal{M}_\lambda$, $S \subseteq [n]$, $(\mathsf{pp}, \mathsf{msk}) \in \mathrm{Supp}\left(\mathsf{Setup}(1^\lambda, n)\right)$ and $\mathsf{ct} \in \mathrm{Supp}\left(\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, S)\right)$. To show that the scheme has succinct ciphertexts, we need to show that $|\mathsf{ct}| \in \mathrm{poly}(\lambda, m(\lambda), \log n)$. First, since WE is efficient, $|\mathsf{ct}| \in \mathrm{poly}(\lambda, m(\lambda), T)$ where T is the time required to check $\mathbf{R}_\mathcal{L}$ on instances in $\mathcal{L}_\lambda$. Checking instances of $\mathbf{R}_\mathcal{L}$ requires running $\mathsf{TPG.Verify}(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hinp}_i, \pi_{\mathsf{TPG}})$ and $\mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{i\}, \{(i, \mathsf{hinp}_i)\}, \pi_{\mathsf{FBH}})$. It suffices to bound the length of the inputs to TPG.Verify and FBH.VerOpen:
  - For the inputs TPG.Verify, note that $|\mathsf{pp}_{\mathsf{TPG}}| \in \mathrm{poly}(\lambda)$, $|\mathsf{hinp}_i| = \ell_{\mathsf{blk}}(\lambda)$, and $|\pi_{\mathsf{TPG}}| \in \mathrm{poly}(\lambda, \ell_{\mathsf{blk}}(\lambda))$, which are all of size $\mathrm{poly}(\lambda)$ by assumption.
  - For the inputs to FBH.VerOpen, first observe that $|\mathsf{hk}| \in \mathrm{poly}(\lambda, s(\lambda), \log n)$ by the efficiency of FBH, which is $\mathrm{poly}(\lambda, \log n)$ as s is polynomially-bounded. Also by the efficiency of FBH, $|\mathsf{dig}|, |\pi_{\mathsf{FBH}}| \in \mathrm{poly}(\lambda, \log |S|)$. Lastly, $|i| \le \log n$ and $|\mathsf{hinp}_i| = \ell_{\mathsf{blk}}(\lambda) \in \mathrm{poly}(\lambda)$. Thus the inputs to FBH.VerOpen have size at most $\mathrm{poly}(\lambda, \log n)$, as required.

  To argue that the scheme has short secret keys, take any index $i \in [n]$ and consider any secret key $\mathsf{sk}_i \in \mathrm{Supp}\left(\mathsf{KeyGen}(\mathsf{msk}, i)\right)$. By construction, $\mathsf{sk}_i = \pi_{\mathsf{TPG}}$. By our above analysis, $|\pi_{\mathsf{TPG}}| = \mathrm{poly}(\lambda, \ell_{\mathsf{blk}}) = \mathrm{poly}(\lambda)$.
- **Correctness:** Let $n = n(\lambda)$ be a bound on the number of participants. Take any $\lambda \in \mathbb{N}$, subset of users $S = \{i_1, \ldots, i_k\} \subseteq [n]$, index $j \in S$, and message $\mathsf{msg} \in \mathcal{M}_\lambda$. The, we have the following:
  - Let $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda))$, where $\mathsf{pp} = (\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hk}, (x_1, \ldots, x_n))$, and $\mathsf{msk} = \mathsf{td}_{\mathsf{TPG}}$. Let $\mathsf{sk}_j \leftarrow \mathsf{KeyGen}(\mathsf{msk}, j)$. By construction, $\mathsf{sk}_j = \pi_{\mathsf{TPG}} = \mathsf{TPG.CreateProof}(\mathsf{td}_{\mathsf{TPG}}, x_j)$.

- By correctness of TPG, $\mathsf{TPG.Verify}(\mathsf{pp}_{\mathsf{TPG}}, x_j, \mathsf{sk}_j) = 1$ with probability at least $1 - \mu(\lambda)$ for some negligible function $\mu = \mu(\lambda)$.
- Let $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, S)$. By construction, Enc computes $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (x_{i_1}, \ldots, x_{i_k}))$. The ciphertext ct is a witness encryption of msg for the instance $(\mathsf{hk}, \mathsf{pp}_{\mathsf{TPG}}, \mathsf{dig})$.
- Consider the output of $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), S)$. Let $j' \in [k]$ be the index such that $j = i_{j'}$. Now, the decryption algorithm computes $\pi_{\mathsf{FBH}} = \mathsf{FBH.ProveOpen}(\mathsf{hk}, (x_{i_1}, \ldots, x_{i_k}), \{j'\})$. By completeness of FBH, $\mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, S, \{(j', x_j)\}, \pi_{\mathsf{FBH}}) = 1$. Thus, $(j', x_j, \mathsf{sk}_j, \pi_{\mathsf{FBH}}) = (j', x_i, \pi_{\mathsf{TPG}}, \pi_{\mathsf{FBH}})$ is a valid witness for the instance $(\mathsf{hk}, \mathsf{pp}_{\mathsf{TPG}}, \mathsf{dig})$ with probability at least $1 - \mu(\lambda)$. By correctness of WE, Dec outputs msg with the same probability. Since μ is negligible, correctness follows. □

Security. Next, we show that Construction 5.16 satisfies semi-static security. The structure of the proof is similar to that of Theorem 4.5, but we provide the details for completeness and to provide another example for how to use function-binding hash functions in conjunction with witness encryption.

Theorem 5.18 (Semi-Static Security). *Assuming* TPG *satisfies mode indistinguishability and trapdoor decidability,* FBH *satisfies computational function hiding and statistical function binding, and* WE *satisfies message indistinguishability, then Construction 5.16 satisfies semi-static security.*

Proof. Suppose by way of contradiction that semi-static security does not hold. Namely, there exists a stateful PPT algorithm A, an efficiently computable function $n \in \mathsf{poly}(\lambda)$, a polynomial q, and an infinite set $\Lambda \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$, it holds that

$$\Pr\left[\mathsf{Expt}^{\mathsf{BE,SS}}_{A,n(\lambda)}(\lambda) = 1\right] \geq 1/2 + 1/q(\lambda).$$

For fixed parameters as above, we define the following sequence of hybrid experiments for each $\lambda \in \mathbb{N}$.

- $H_{b,0}(\lambda)$: This hybrid is identical to the experiment $\mathsf{Expt}^{\mathsf{BE,SS}}_{A,n(\lambda)}(\lambda)$ for the fixed value of b.
- $H_{b,1}(\lambda)$: This hybrid is the same as $H_{b,0}$ except the challenger samples the public parameters pp using the following modified algorithm:
  - First, it samples $(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{td}_{\mathsf{TPG}}) \leftarrow \mathsf{TPG.SetupAlt}(1^\lambda)$.
  - The hash key $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, n)$ is sampled as in $H_{b,0}$.
  - Let $S \subseteq [n]$ be the set the adversary commits to at the beginning of the security game. For each $i \in [n] \setminus S$, the challenger samples $(x_i, \pi_i) \leftarrow \mathsf{TPG.SampleTrue}(\mathsf{pp}_{\mathsf{TPG}})$. For each $i \in S$, the challenger samples $x_i \leftarrow \mathsf{TPG.SampleFalse}(\mathsf{pp}_{\mathsf{TPG}})$.

  The public parameters pp is still defined to be $\mathsf{pp} = (\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hk}, (x_1, \ldots, x_n))$. Then, when answering the key-generation queries for an index $i \in [n] \setminus S$, the challenger responds with π_i.
- $H_{b,2}(\lambda)$: In this hybrid, the challenger instead computes $\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^\lambda, n(\lambda), f_g)$ where f_g is the function from Eq. (3.1) and g is the function $\mathsf{TPG.TDDecide}(\mathsf{td}_{\mathsf{TPG}}, \cdot)$.
- $H_{b,3}(\lambda)$: In this hybrid, instead of encrypting msg_b in the semi-static security game, the challenger encrypts the lexicographically first message $\mathsf{msg}^\star \in \mathcal{M}_\lambda$: $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}^\star, S^\star)$.

Since the challenge bit is sampled uniformly (i.e., $b \xleftarrow{R} \{0,1\}$), the success probability of A on any security parameter λ is given by

$$\Pr[A \text{ wins}] = \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right).$$

For each $b \in \{0,1\}$ and $i \in \{1,2,3\}$, we define

$$\delta_{b,i}(\lambda) = \Pr[H_{b,i-1}(\lambda) = 1] - \Pr[H_{b,i}(\lambda) = 1].$$

It follows by our assumption that for all $\lambda \in \Lambda$,

$$\frac{1}{2} + \frac{1}{q(\lambda)} \leq \frac{1}{2} \cdot \big(\Pr[H_{0,3}(\lambda) = 1] + \delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) + \delta_{0,3}(\lambda) \\ + \Pr[H_{1,3}(\lambda) = 1] + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda) + \delta_{1,3}(\lambda) \big).$$

Since $\Pr[H_{b,3}(\lambda) = 1] \leq 1/2$ for each $b \in \{0,1\}$, it follows that for all $\lambda \in \Lambda$ that

$$\frac{1}{q(\lambda)} \leq \frac{1}{2} \cdot \left(\delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) + \delta_{0,3}(\lambda) + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda) + \delta_{1,3}(\lambda) \right).$$

As Λ is an infinitely large subset of $\mathbb{N}$, it must be the case that for some $b \in \{0,1\}$ and $i \in \{1,2,3\}$, $\delta_{b,i}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. In the following three claims, we show that this must contradict one of our assumptions from the statement of Theorem 5.18. We prove the following claims for $b = 0$ without loss of generality.

Claim 5.19. *If $\delta_{0,1}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then* TPG *does not satisfy mode indistinguishability.*

Proof. We construct a stateful PPT adversary B that uses the adversary A to break the mode indistinguishability of TPG. We define B as follows:

1. At the beginning of the game, algorithm B receives the public parameters $\mathsf{pp}_{\mathsf{TPG}}$ for TPG as input from the challenger. Specifically, the mode-indistinguishability challenger samples a bit $\hat{b} \xleftarrow{R} \{0,1\}$ and computes $(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{td}) \leftarrow \mathsf{TPG.Setup}(1^\lambda)$ if $\hat{b} = 0$ and $(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{td}) \leftarrow \mathsf{TPG.SetupAlt}(1^\lambda)$ if $\hat{b} = 1$.
2. Algorithm B computes $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, n)$ as in $H_{0,0}$ and $H_{0,1}$.
3. Algorithm B starts running adversary A who starts by committing to a set $S \subseteq [n]$. For each $i \in S$, algorithm B makes a false-instance query to its challenger to obtain x_i. For each $i \in [n] \setminus S$, algorithm B makes a true-instance query to its challenger to obtain (x_i, π_i).
   - If $\hat{b} = 0$, then in both cases, $x_i \xleftarrow{R} \mathcal{X}_\lambda$ and for the true-instance queries, $\pi_i \leftarrow \mathsf{CreateProof}(\mathsf{td}, x_i)$.
   - If $\hat{b} = 1$, then for the true-instance queries, the challenger samples $(x_i, \pi_i) \leftarrow \mathsf{SampleTrue}(\mathsf{pp}_{\mathsf{TPG}})$ and for the false-instance queries, $x_i \leftarrow \mathsf{SampleFalse}(\mathsf{pp}_{\mathsf{TPG}})$.

   Algorithm B gives $\mathsf{pp} = (\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hk}, (x_1, \ldots, x_n))$ to A.
4. Whenever algorithm A makes a key-generation query for an index $i \in [n] \setminus S$, algorithm B replies with the secret key $\mathsf{sk}_i = \pi_i$.
5. When algorithm A makes its challenge query on messages $\mathsf{msg}_0, \mathsf{msg}_1$ and a set $S^\star \subseteq S$, algorithm B computes $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}_0, S^\star)$ as in $H_{0,0}$ and $H_{0,1}$ (using the pp constructed above).
6. Algorithm A responds with a bit $b' \in \{0,1\}$, which B also outputs.

If A is efficient, then so is B, so it is sufficient to analyze the advantage of B. By construction, if the mode indistinguishability challenger samples $\hat{b} = 0$, then the view of A is distributed exactly as in $H_{0,0}$, and if $\hat{b} = 1$ is sampled, then the view of A is distributed exactly as in $H_{0,1}$. Therefore, when $\hat{b} = 0$, algorithm B succeeds if $b' = 0$ which means that $H_{0,0}(\lambda) = 1$, and when $\hat{b} = 1$, algorithm B succeeds if $b' = 1$ which means that $H_{0,1}(\lambda) = 0$. Thus, the success probability of B is given by

$$\Pr[\hat{b}' = \hat{b}] = \frac{1}{2} \cdot \Pr[H_{0,0}(\lambda) = 1] + \frac{1}{2} \cdot \big(1 - \Pr[H_{0,1}(\lambda) = 1]\big) = \frac{1}{2} + \delta_{0,1}(\lambda).$$

If $\delta_{0,1}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then this violates the mode indistinguishability property of TPG. □

Claim 5.20. *If $\delta_{0,2}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then* FBH *does not satisfy computational function hiding.*

Proof. The proof of this claim follows nearly identically to the proof of Claim 4.7. □

Claim 5.21. *If $\delta_{0,3}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then either (1) WE does not satisfy message indistinguishability; (2) FBH does not satisfy statistical function binding; or (3) TPG does not satisfy trapdoor decidability.*

Proof. We construct a PPT adversary B that uses the adversary A to break the message indistinguishability security of WE or we show that either A breaks the statistical function binding of FBH or trapdoor decidability of TPG. We construct algorithm B as follows:

1. Algorithm B takes 1^λ as input and starts running algorithm A. After algorithm A commits to its set $S \subseteq [n]$, it constructs the public parameters pp exactly as described in $H_{0,3}$. Similarly, it answers key-generation queries according to the specification in $H_{0,3}$.

2. When algorithm A makes its challenge query on messages $\mathsf{msg}_0, \mathsf{msg}_1$ and a set $S^\star = \{i_1, \ldots, i_k\} \subseteq S$, algorithm B computes $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (x_{i_1}, \ldots, x_{i_k}))$. It then outputs the statement $x = (\mathsf{hk}, \mathsf{pp}_{\mathsf{TPG}}, \mathsf{dig})$ and challenge messages $(\mathsf{msg}_0, \mathsf{msg}^\star)$, where $\mathsf{msg}^\star$ is the lexicographically-first message in $\mathcal{M}_\lambda$.

3. The FBH challenger replies with a challenge ciphertext $\mathsf{ct}^\star$ which algorithm B forwards to A. Specifically, the witness encryption challenger samples a random bit $\hat{b} \xleftarrow{R} \{0,1\}$ and sets $\mathsf{ct}^\star \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathsf{msg}_0, x)$ if $\hat{b} = 0$ and $\mathsf{ct}^\star \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathsf{msg}^\star, x)$ if $\hat{b} = 1$.

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

It A is efficient, then so is B, so it suffices to compute the advantage of algorithm B. For any $\lambda \in \mathbb{N}$, let $\mathsf{Wit}(\lambda)$ be the event that there exists a valid witness $(j, \mathsf{hinp}_j, \pi_{\mathsf{TPG}}, \pi_{\mathsf{FBH}})$ for the instance $x = (\mathsf{hk}, \mathsf{pp}_{\mathsf{TPG}}, \mathsf{dig})$ such that

$$\mathsf{TPG.Verify}(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hinp}_j, \pi_{\mathsf{TPG}}) = 1 \wedge \mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{j\}, \{(j, \mathsf{hinp}_j)\}, \pi_{\mathsf{FBH}}) = 1. \tag{5.1}$$

By construction, the view of A is identical to hybrid $H_{0,2}(\lambda)$ if $\hat{b} = 0$ and is identical to hybrid $H_{0,3}(\lambda)$ if $\hat{b} = 1$. When $\hat{b} = 0$, algorithm A outputs $b' = 0$ if $H_{0,2}(\lambda) = 1$ (since $b = 0$). Similarly, when $\hat{b} = 1$, algorithm A outputs $b' = 1$ if $H_{0,3}(\lambda) = 0$. Additionally, if $\mathsf{Wit}(\lambda)$ holds, then B always loses the message indistinguishability game since it implies that $x \in \mathcal{L}_\lambda$. Thus, the success probability of B is at least

$$\begin{aligned}
\Pr[B \text{ wins}] &\geq \Pr[\hat{b}' = \hat{b}] - \Pr[\mathsf{Wit}(\lambda)] \\
&\geq \frac{1}{2} \cdot \Pr[H_{0,2}(\lambda) = 1] + \frac{1}{2} \cdot (1 - \Pr[H_{0,3}(\lambda) = 1]) - \Pr[\mathsf{Wit}(\lambda)] \\
&\geq \frac{1}{2} + \frac{1}{2} \cdot \delta_{0,3}(\lambda) - \Pr[\mathsf{Wit}(\lambda)].
\end{aligned}$$

If $\Pr[\mathsf{Wit}(\lambda)] \leq 1/(12q(\lambda))$ for infinitely many $\lambda \in \Lambda$, then this violates message indistinguishability of WE since $\delta_{0,3}(\lambda) \geq 1/(3q(\lambda))$. To complete the proof, we show that if $\Pr[\mathsf{Wit}(\lambda)] > 1/(12q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then we can either break the statistical function binding of FBH or the statistical binding for false instances and trapdoor decidability of TPG. Suppose

$$\Pr[\mathsf{Wit}(\lambda)] > 1/(12q(\lambda)) \tag{5.2}$$

for infinitely many $\lambda \in \mathbb{N}$. This means that there exists a witness $(j, \mathsf{hinp}_j, \pi_{\mathsf{TPG}}, \pi_{\mathsf{FBH}})$ that satisfies Eq. (5.1) with probability at least $1/(12q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. In the above construction, $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (x_{i_1}, \ldots, x_{i_k}))$. Recall also that in $H_{0,3}$, the hash key hk is sampled to be function binding for the function f_g where

$$g(x) \coloneqq \mathsf{TPG.TDDecide}(\mathsf{td}_{\mathsf{TPG}}, x).$$

Let $\mathsf{E}(\lambda)$ be the event (taken over the random coins of SetupAlt and SampleFalse) that there exists an index $\ell \in [k]$ such that $g(x_{i_\ell}) = 1$. We consider two cases:

- Suppose for infinitely many $\lambda \in \mathbb{N}$, $\Pr[\mathsf{E}(\lambda)] \leq 1/(24q(\lambda))$. Under the assumption that Eq. (5.2), this means that $\Pr[\mathsf{Wit}(\lambda) \wedge \neg\mathsf{E}(\lambda)] > 1/(24q(\lambda))$. In this case, for all $\ell \in [k]$, $g(x_{i_\ell}) = 0$, so

$$f_g(x_{i_1}, \ldots, x_{i_k}) = \bigvee_{\ell \in [k]} g(x_{i_\ell}) = 0.$$

  Since $\mathsf{Wit}(\lambda)$ holds, there exists a witness $(j, \mathsf{hinp}_j, \pi_{\mathsf{TPG}}, \pi_{\mathsf{FBH}})$ that satisfies Eq. (5.1) for $x = (\mathsf{hk}, \mathsf{pp}_{\mathsf{TPG}}, \mathsf{dig})$ from the experiment. This means that $\mathsf{TPG.Verify}(\mathsf{pp}_{\mathsf{TPG}}, \mathsf{hinp}_j, \pi_{\mathsf{TPG}}) = 1$. Then there are two possibilities:

  - Suppose $\mathsf{TPG.TDDecide}(\mathsf{td}_{\mathsf{TPG}}, \mathsf{hinp}_j) = 0$. This violates the trapdoor decidability property of TPG.
  - Suppose $g(\mathsf{hinp}_j) = \mathsf{TPG.TDDecide}(\mathsf{td}_{\mathsf{TPG}}, \mathsf{hinp}_j) = 1$. Then the output of f_g on any input that contains hinp_j is $1 \neq f_g(x_{i_1}, \ldots, x_{i_k}) = 0$. Since $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, (x_{i_1}, \ldots, x_{i_k}))$, the existence of an opening π_{FBH} for (j, hinp_j) with respect to dig breaks statistical function binding of FBH.

  If $\Pr[\mathsf{Wit}(\lambda) \wedge \neg\mathsf{E}(\lambda)] > 1/(24q(\lambda))$, then at least one of the two events above happens with probability at least $1/(48q(\lambda))$. This breaks either statistical function binding of FBH or trapdoor decidability of TPG.

- Suppose for infinitely many $\lambda \in \mathbb{N}$, $\Pr[\mathsf{E}(\lambda)] > 1/(24q(\lambda))$. This means that with probability $1/(24q(\lambda))$, there exists an index $\ell \in [k]$ such that $g(x_{i_\ell}) = \mathsf{TPG.TDDecide}(\mathsf{td}_{\mathsf{TPG}}, x_{i_\ell}) = 1$. Since in $H_{0,2}$ and $H_{0,3}$, the challenger samples $x_{i_\ell} \leftarrow \mathsf{SampleFalse}(\mathsf{pp}_{\mathsf{TPG}})$, this translates to an attack on the trapdoor decidability property of TPG with advantage $1/(24q(\lambda) \cdot k(\lambda))$, which is non-negligible since $k \leq n$ is polynomially-bounded.

We conclude that assuming statistical function binding of FBH and trapdoor decidability of TPG, it cannot be the case that $\Pr[\mathsf{Wit}(\lambda)] > 1/(12q(\lambda))$, and the claim follows. □

The proof follows from Claims 5.19 to 5.21. □

Remark 5.22 (Optimal Broadcast Encryption in the Random Oracle Model)**.** Construction 5.16 gives a broadcast encryption scheme with short ciphertexts and secret keys. However, the public parameters are long (i.e., scales linearly with the number of users n). Specifically, the public parameters consist of the public parameters $\mathsf{pp}_{\mathsf{TPG}}$ for the trapdoor proof system, a hash key hk for a function-binding hash function, and n random values $x_1, \ldots, x_n \xleftarrow{R} \mathcal{X}$ from the domain of the trapdoor proof system. By definition, the size of the public parameters $\mathsf{pp}_{\mathsf{TPG}}$ is $\mathsf{poly}(\lambda)$ and the size of the hash key hk is $\mathsf{poly}(\lambda, \log n)$; see Theorem 5.17 for a more detailed calculation. Since the points $x_1, \ldots, x_n$ are uniformly random over $\mathcal{X}$ (and independent of all other parameters), we can "compress" them in the random oracle model. Namely, suppose $\mathcal{O} \colon [n] \to \mathcal{X}$ is a hash function modeled as a random oracle. Then, in Construction 5.16, we can define $x_i \leftarrow \mathcal{O}(i)$. In this way, the public parameters only needs to contain $\mathsf{pp}_{\mathsf{TPG}}$ and hk, both of which are short. This yields an optimal broadcast encryption scheme from witness encryption, function-binding hash functions, and trapdoor proof generators in the random oracle model. The resulting construction satisfies semi-static security, but can be generically boosted to full adaptive security via the Gentry-Waters [GW09] transformation (which also relies on the random oracle model).

6 Registered Attribute-Based Encryption

In this section, we recall the notion of a *slotted* registered attribute-based encryption (ABE) scheme [HLWW23] and then show how to construct it from a witness encryption scheme together with a function binding hash function. To obtain a standard registered ABE scheme (without slots), we can then apply the generic transformation from [HLWW23] (restated in Theorem A.4). We recall the formal definition of a registered ABE scheme from [HLWW23] in Appendix A. Our (slotted) registered ABE supports general policies over an attribute universe of super-polynomial size, an unbounded number of users, and has a transparent setup. In contrast, the pairing-based construction from [HLWW23] could only support monotone Boolean formula policies over a polynomial-size attribute universe, an a priori bounded number of users, and relied on a *structured* reference string whose size scales quadratically with the number of users. On the flip side, the pairing-based construction from [HLWW23] is adaptively secure whereas our construction is only selectively secure.

6.1 Slotted Registered ABE Definition

In this section, we recall the notion of a *slotted* registered ABE scheme from [HLWW23]. In our setting, each user is associated with an attribute of length $\ell = \ell(\lambda)$ and each ciphertext is associated with a policy $P\colon \{0,1\}^{\ell(\lambda)} \to \{0,1\}$ taken from some policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$. We say that an attribute x satisfies the policy P if $P(x) = 1$. We now recall the formal definition. We present the definition for the setting with a large (i.e., super-polynomial size) attribute universe and for supporting an arbitrary number of users (c.f., [HLWW23, Remark 6.10]):

Definition 6.1 (Slotted Registered ABE [HLWW23, adapted])**.** Let $\ell = \ell(\lambda)$ be an attribute length and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a policy space on ℓ-bit inputs (i.e., $\mathcal{P}_\lambda$ is a set of functions $P\colon \{0,1\}^{\ell(\lambda)} \to \{0,1\}$). Let $m = m(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space with message length m. A *slotted registered attribute-based encryption scheme* for attributes of length ℓ, policy space $\mathcal{P}$, and message space $\mathcal{M}$ consists of polynomial-time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$[9] with the following syntax:

- $\mathsf{Setup}(1^\lambda, L) \to \mathsf{pp}$: A probabilistic algorithm that on input a security parameter λ and a number of slots L, outputs public parameters pp. We implicitly assume that pp contains 1^λ and L.
- $\mathsf{KeyGen}(\mathsf{pp}, i) \to (\mathsf{pk}, \mathsf{sk})$: A probabilistic algorithm that on input the public parameters pp and a slot index $i \in [L]$, outputs a public key pk and a secret key sk.
- $\mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)) \to (\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L)$: A deterministic algorithm that on input public parameters pp and a list of public keys with associated attributes $(\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)$ outputs a master public key mpk and a sequence of helper decryption keys $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$.
- $\mathsf{Enc}(\mathsf{mpk}, P, \mathsf{msg}) \to \mathsf{ct}$: A probabilistic algorithm that on input a master public key mpk, an access policy $P \in \mathcal{P}_\lambda$, and a message $\mathsf{msg} \in \mathcal{M}_\lambda$, outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{mpk}, \mathsf{hsk}, \mathsf{sk}, \mathsf{ct}) \to \mathsf{msg}$: A deterministic algorithm that on input a master public key mpk, a helper decryption key hsk, a secret key sk, and a ciphertext ct, outputs a message $\mathsf{msg} \in \mathcal{M}_\lambda \cup \{\bot\}$.

We require that $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$ satisfy the following properties:

- **Compactness:** There exists a polynomial p such that for any $\lambda \in \mathbb{N}$, $L \in \mathbb{N}$, public parameters $\mathsf{pp} \in \mathrm{Supp}\left(\mathsf{Setup}(1^\lambda, L)\right)$, public/secret key-pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \in \mathrm{Supp}\left(\mathsf{KeyGen}(\mathsf{pp}, i)\right)$, and attributes $x_i \in \{0,1\}^{\ell(\lambda)}$ for each $i \in [L]$, and setting $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) = \mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$, the following holds:
  - $|\mathsf{mpk}| \le p(\lambda, \ell, \log L)$, and
  - $|\mathsf{hsk}_i| \le p(\lambda, \ell, \log L)$ for each $i \in [L]$.

  This compactness notion requires that the size of the master public key and the helper decryption keys scale polylogarithmically with the size of the attribute universe ($|\{0,1\}^\ell| = 2^\ell$). We can define a weaker notion of compactness where the size of the master public key and the helper decryption keys scale polynomially with the size of the attribute universe. This weaker notion is the default in [HLWW23], as the parameters of their pairing-based construction scale polynomially with the size of the attribute universe.
- **Correctness:** For any $\lambda \in \mathbb{N}, L \in \mathbb{N}, i \in [L]$, message $\mathsf{msg} \in \mathcal{M}_\lambda$, attributes $x_1, \ldots, x_L \in \{0,1\}^{\ell(\lambda)}$, policy $P \in \mathcal{P}_\lambda$ where $P(x_i) = 1$, $\mathsf{pp} \in \mathrm{Supp}\left(\mathsf{Setup}(1^\lambda, L)\right)$, $(\mathsf{pk}_i, \mathsf{sk}_i) \in \mathrm{Supp}\left(\mathsf{KeyGen}(\mathsf{pp}, i)\right)$, and any public keys $\{\mathsf{pk}_j\}_{j\neq i\in[L]}$ (which may be correlated with pk_i), let $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) = \mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$. Then, it holds that
$$\Pr\left[\begin{array}{l|c} \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, P, \mathsf{msg}) & \\ \mathsf{msg}' = \mathsf{Dec}(\mathsf{mpk}, \mathsf{hsk}_i, \mathsf{sk}_i, \mathsf{ct}) & : \ \mathsf{msg}' = \mathsf{msg} \end{array}\right] = 1.$$

[9] The definition from [HLWW23] also includes an algorithm $\mathsf{IsValid}(\mathsf{pp}, i, \mathsf{pk}_i) \to \{0,1\}$ that checks if a given public key is valid. Our construction does not require this check. However, to match their syntax, we could define it to simply output 1 on any public key of the correct length with respect to the public parameters pp.

- **Adaptive security:** For all stateful PPT adversaries A and efficiently-computable function $L \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Expt}^{\mathsf{srABE}}_{A,L(\lambda)}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

  where $\mathsf{Expt}^{\mathsf{srABE}}_{A,L}(\lambda)$ is defined via the following security game between the adversary A and a challenger on common input 1^λ:

  - **Setup phase:** The challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, L)$ and sends pp to A. The challenger initializes a counter $\mathsf{ctr} := 0$, a dictionary D, and a set of (corrupted) slot indices $C := \emptyset$.
  - **Pre-challenge query phase:** The adversary A can now issue the following queries:
    - **Key-generation query:** In a key-generation query, the adversary specifies a slot index $i \in [L]$. The challenger increments the counter value $\mathsf{ctr} := \mathsf{ctr} + 1$, samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$ and replies with $(\mathsf{ctr}, \mathsf{pk})$ to A. The challenger adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}, \mathsf{sk})$ to the dictionary D.
    - **Corruption query:** In a corruption query, the adversary specifies a counter value $c \in [\mathsf{ctr}]$. In response, the challenger looks up the tuple $(i, \mathsf{pk}, \mathsf{sk}) := \mathsf{D}[c]$ and replies to A with sk.
  - **Challenge phase:** For each slot $i \in [L]$, the adversary A specifies a tuple $(c_i, x_i, \mathsf{pk}^\star_i)$ where either $c_i \in \{1, \ldots, \mathsf{ctr}\}$ to reference a challenger-generated key or $c_i = \perp$ to reference a key outside this set. The adversary also sends a challenge policy $P^\star \in \mathcal{P}_\lambda$ and two messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}_\lambda$. The challenger responds by first constructing pk_i as follows for each $i \in [L]$:
    - If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, then the challenger sets $(i', \mathsf{pk}, \mathsf{sk}) := \mathsf{D}[c_i]$. If $i = i'$, then the challenger sets $\mathsf{pk}_i := \mathsf{pk}$. Moreover, if the adversary previously issued a corruption query on counter c_i, then the challenger adds the slot index i to C. Otherwise, if $i \neq i'$, then the experiment halts and outputs 0.
    - If $c_i = \perp$, the challenger sets $\mathsf{pk}_i := \mathsf{pk}^\star_i$ and adds the slot index i to C.

    The challenger computes $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) = \mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$, samples a random bit $b \leftarrow \{0, 1\}$, known as the *challenge bit*, and replies with the challenge ciphertext $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{mpk}, P^\star, \mathsf{msg}_b)$.[10]
  - **Output phase:** At the end of the experiment, the adversary A outputs a bit $b' \in \{0, 1\}$. We say that algorithm A is *admissible* if $P^\star(x_i) = 0$ for all $i \in C$ (i.e., the attributes associated with corrupted slots do not satisfy the challenge policy). The experiment outputs 1 if $b' = b$ and A is admissible. Otherwise, the experiment outputs 0.

Selective security. We now consider two versions of selective security that relax the notion of adaptive security defined above. We consider (1) policy-selective security, where the adversary has to declare its challenge policy at the beginning of the security game; and (2) policy-selective security without corruptions where the adversary has to declare its challenge policy at the beginning of the security game, and moreover, it cannot make any corruption queries. Our construction of slotted registered ABE will satisfy the weakest notion of policy-selective security without corruptions. In Appendix C, we show how to adapt the two-key approach from [GW09] to transform a slotted registered ABE scheme that satisfies policy-selective security without corruptions into one that satisfies policy-selective security with corruptions in the random oracle model. We now give a formal definition of our relaxed security definitions:

Definition 6.2 (Policy-Selective Security)**.** We say that a slotted registered ABE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$ satisfies policy-selective security if instead of adaptive security, it satisfies the following:

- **Policy-selective security:** For all stateful PPT adversaries A and efficiently-computable function $L \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Expt}^{\mathsf{srABE,PS}}_{A,L(\lambda)}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

[10] Note that because Aggregate is deterministic and can be run by A itself, there is no need to additionally provide $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L)$ to A. Similarly, there is no advantage to allowing the adversary to select the challenge policy and messages after seeing the aggregated key.

where $\mathsf{Expt}^{\mathsf{srABE,PS}}_{A,L}(\lambda)$ is the adaptive security game $\mathsf{Expt}^{\mathsf{srABE}}_{A,L}(\lambda)$ with the following modification:

- **Setup phase:** At the beginning of the setup phase, the adversary sends the challenge policy $P^\star$ to the challenger. The rest of the setup phase proceeds as in $\mathsf{Expt}^{\mathsf{srABE}}_{A,L}(\lambda)$.

The remainder of $\mathsf{Expt}^{\mathsf{srABE,PS}}_{A,L}(\lambda)$ proceeds exactly as defined in $\mathsf{Expt}^{\mathsf{srABE}}_{A,L}(\lambda)$.

Definition 6.3 (Policy-Selective Security without Corruptions)**.** We say that a slotted registered ABE scheme (Setup, KeyGen, Aggregate, Enc, Dec) satisfies policy-selective security without corruptions if instead of adaptive security, it satisfies the following:

- **Policy-selective security without corruptions:** For all stateful PPT adversaries A and efficiently-computable function $L \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Expt}^{\mathsf{srABE,PSWC}}_{A,L(\lambda)}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

where $\mathsf{Expt}^{\mathsf{srABE,PSWC}}_{A,L}(\lambda)$ is the policy-selective security game $\mathsf{Expt}^{\mathsf{srABE,PS}}_{A,L}(\lambda)$ with the following modifications:

- **Pre-challenger query phase:** Same as $\mathsf{Expt}^{\mathsf{srABE}}_{A,L}(\lambda)$ except the adversary A is not allowed to make any corruption queries.

The remainder of $\mathsf{Expt}^{\mathsf{srABE,PSWC}}_{A,L}(\lambda)$ proceeds exactly as defined in $\mathsf{Expt}^{\mathsf{srABE}}_{A,L}(\lambda)$.

6.2 Constructing Slotted Registered ABE

We now show how to construct a slotted registered ABE scheme from witness encryption, function binding hash functions, and public-key encryption. Our scheme satisfies policy-selective security without corruptions. We show in Appendix C how to use this to generically transform this scheme into one that satisfies policy-selective security with corruptions in the random oracle model. As was the case for our flexible broadcast encryption scheme (Construction 4.3), the public parameters for our registered ABE scheme is a uniform random string, and thus, our scheme supports a transparent setup (see Remark 6.10).

Construction 6.4 (Slotted Registered ABE)**.** Let $\ell = \ell(\lambda)$ be the attribute length and $m = m(\lambda)$ be the message length. Let $s = s(\lambda, \ell)$, $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, and $d = d(\lambda, \ell)$ be polynomials. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space for message of length m. Let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies on ℓ-bit attributes (i.e., $\mathcal{P}_\lambda$ consists of functions $P\colon \{0,1\}^{\ell(\lambda)} \to \{0,1\}$) where each policy $P \in \mathcal{P}_\lambda$ can be implemented by a Boolean circuit of size s and depth d. Our construction of slotted registered ABE relies on the following primitives:

- Let $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be a semantically-secure public-key bit encryption scheme where, for $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{Supp}\left(\mathsf{PKE.KeyGen}(1^\lambda)\right)$, ciphertexts have length at most $\ell_{\mathsf{blk}}(\lambda)$ and decryption can be computed by a circuit of size s and depth d.
- Let $\mathsf{FBH} = (\mathsf{FBH.Setup}, \mathsf{FBH.SetupBinding}, \mathsf{FBH.Hash}, \mathsf{FBH.ProveOpen}, \mathsf{FBH.VerOpen})$ be a function-binding hash for the class $\mathcal{F}$ of disjunctions of block functions for input length $\ell_{\mathsf{blk}}(\lambda) + \ell(\lambda)$, size $2s + 1$, and depth d (Definition 3.3).
- Let $\mathsf{WE} = (\mathsf{WE.Enc}, \mathsf{WE.Dec})$ be a witness encryption scheme for the language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda\in\mathbb{N}}$ defined by the relation $\mathbf{R}_{\mathcal{L}}$ as follows. Instances of the language $\mathcal{L}_\lambda$ are of the form $(\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}, P)$ and the relation $\mathbf{R}_{\mathcal{L}}$ is given by

$$\begin{aligned}&\mathbf{R}_{\mathcal{L}}((\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}, P), (i, \mathsf{ct}, x, r, \pi)) = 1\\ &\Leftrightarrow \mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; r) \wedge P(x) = 1 \ \wedge \mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{i\}, \{(i, (\mathsf{ct}, x))\}, \pi) = 1.\end{aligned}$$

We construct a slotted registered ABE scheme $\mathsf{srABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$ with attribute length ℓ, policy space $\mathcal{P}$, and message space $\mathcal{M}$ as follows:

- $\mathsf{Setup}(1^\lambda, L)$: On input the security parameter λ and the number of slots L, the setup algorithm samples $(\mathsf{pk}_{\mathsf{PKE}}, \mathsf{sk}_{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$, $\mathsf{hk} \leftarrow \mathsf{FBH.Setup}(1^\lambda, L)$, and outputs public parameters $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$.
- $\mathsf{KeyGen}(\mathsf{pp}, i)$: On input the public parameters $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$ and an index $i \in [L]$, the key-generation algorithm samples $r \xleftarrow{R} \{0,1\}^\lambda$ and computes $\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; r)$. It then outputs the public key $\mathsf{pk} = \mathsf{ct}$ and secret key $\mathsf{sk} = r$.
- $\mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$: On input the public parameters $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$ and a ordered list of public keys pk_i with associated attributes $x_i \in \{0,1\}^\ell$, the aggregation algorithm first computes the digest $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, ((\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)))$. Then, for each $i \in [L]$, it computes the proof $\pi_i = \mathsf{FBH.ProveOpen}(\mathsf{hk}, ((\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)), \{i\})$. Finally, it outputs the master public key $\mathsf{mpk} = (\mathsf{pp}, \mathsf{dig})$ and the helper decryption key $\mathsf{hsk}_i = (i, \pi_i, \mathsf{pk}_i, x_i)$ for each $i \in [L]$.
- $\mathsf{Enc}(\mathsf{mpk}, P, \mathsf{msg})$: On input the master public key $\mathsf{mpk} = ((\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk}), \mathsf{dig})$, a policy $P \in \mathcal{P}_\lambda$, and a message $\mathsf{msg} \in \mathcal{M}_\lambda$, the encryption algorithm outputs $\mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathsf{msg}, (\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}, P))$.
- $\mathsf{Dec}(\mathsf{mpk}, \mathsf{hsk}, \mathsf{sk}, \mathsf{ct})$: On input the master public key mpk, the helper decryption key $\mathsf{hsk} = (i, \pi, \mathsf{pk}, S)$, a secret key sk, and a ciphertext ct, the decryption algorithm outputs $\mathsf{msg} = \mathsf{WE.Dec}(1^\lambda, \mathsf{ct}, (i, \mathsf{pk}, S, \mathsf{sk}, \pi))$.

Correctness and efficiency. We start by showing that Construction 6.4 satisfies (perfect) correctness and compactness.

Theorem 6.5 (Correctness and Efficiency)**.** *Assuming* PKE *and* WE *are correct and* FBH *is complete, then Construction 6.4 is compact and correct.*

Proof. We show each property separately:

- **Compactness:** Compactness follows by efficiency of WE and FBH. Specifically, $|\mathsf{mpk}| = |\mathsf{pp}| + |\mathsf{dig}|$. First, $|\mathsf{pp}| \in \mathsf{poly}(\lambda, \log L)$ by construction of Setup. Similarly, $|\mathsf{dig}| \in \mathsf{poly}(\lambda, \log L)$ by efficiency of FBH, so the master public key is short. Next, consider the helper decryption keys. For any slot $i \in [L]$, we have $|\mathsf{hsk}_i| = |i| + |\pi_i| + |\mathsf{pk}_i| + |x_i|$. By construction, $|i| \le \log L$, $|\pi_i| \in \mathsf{poly}(\lambda, \log L)$ by efficiency of FBH and $|\mathsf{pk}_i| \in \mathsf{poly}(\lambda)$ by efficiency of PKE, and $|x_i| = \ell$. Thus, $|\mathsf{hsk}_i| \in \mathsf{poly}(\lambda, \ell, \log L)$, as required.
- **Correctness:** Take any security parameter $\lambda \in \mathbb{N}$, number of slots $L \in \mathbb{N}$, index $i \in [L]$, message $\mathsf{msg} \in \mathcal{M}_\lambda$, attributes $x_1, \ldots, x_L \in \{0,1\}^\ell$, policy $P \in \mathcal{P}_\lambda$ such that $P(x_i) = 1$. Then, we have the following:
  - Let $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk}) \leftarrow \mathsf{Setup}(1^\lambda, L)$, and for each $i \in [L]$, let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$. By construction, this means that $\mathsf{pk}_i = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; \mathsf{sk}_i)$.
  - Let $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) = \mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$ and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, P, \mathsf{msg})$. By construction, $\mathsf{ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathsf{msg}, (\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}, P))$. The encryption algorithm computes the digest as $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, ((\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)))$.
  - We argue that $\mathsf{Dec}(\mathsf{mpk}, \mathsf{hsk}_i, \mathsf{sk}_i, \mathsf{ct}) = \mathsf{msg}$. Specifically, we show that using $\mathsf{hsk}_i = (i, \pi_i, \mathsf{pk}_i, x_i)$ and sk_i, we can construct a valid witness for the statement $(\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}, P) \in \mathcal{L}_\lambda$. Specifically, $(i, \mathsf{pk}_i, x_i, \mathsf{sk}_i, \pi_i)$ is a valid witness. By construction $\pi_i = \mathsf{FBH.ProveOpen}(\mathsf{hk}, ((\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)), \{i\})$. By completeness of FBH, it follows that $\mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{i\}, \{(i, (\mathsf{pk}_i, x_i))\}, \pi_i) = 1$. Moreover, $\mathsf{pk}_i = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; \mathsf{sk}_i)$ and $P(x_i) = 1$ so we conclude that $(i, \mathsf{pk}_i, x_i, \mathsf{sk}_i, \pi_i)$ is a valid witness. The claim now follows by correctness of WE. □

Selective security. We now show that Construction 6.4 satisfies policy-selective security without corruptions. In Appendix C, we show how to generically transform this scheme into one that satisfies policy-selective security with corruptions in the random oracle mode. The analysis follows a similar structure as the security analysis for our flexible broadcast encryption scheme (Theorem 4.5).

Theorem 6.6 (Policy-Selective Security without Corruptions)**.** *Assuming* PKE *is semantically secure and satisfies perfect correctness,* FBH *satisfies computational function hiding and statistical function binding, and* WE *satisfies message indistinguishability, then Construction 6.4 satisfies policy-selective security without corruptions.*

Proof. Suppose by way of contradiction that srABE does not satisfy policy-selective security without corruptions. Namely, there exists a stateful PPT adversary A, an efficiently computable function $L \in \mathsf{poly}(\lambda)$, a polynomial q, and an infinite set $\Lambda \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$, it holds that

$$\Pr\left[\mathsf{Expt}^{\mathsf{srABE},\mathsf{PSWC}}_{A,L(\lambda)}(\lambda) = 1\right] \geq 1/2 + 1/q(\lambda).$$

For fixed parameters as above, we define the following sequence of hybrid experiments for each $\lambda \in \mathbb{N}$.

- $H_{b,0}(\lambda)$: This is the experiment $\mathsf{Expt}^{\mathsf{srABE},\mathsf{PSWC}}_{A,L(\lambda)}(\lambda)$ where the challenge bit is fixed to the value b.
- $H_{b,1}(\lambda)$: This hybrid is identical to the previous experiment, except in key generation queries, the challenger samples $(\mathsf{pk}, \mathsf{sk})$ such that pk is an encryption of 0. Specifically, the challenger responds to key-generation queries by sampling $(\mathsf{pk}, \mathsf{sk}) = (\mathsf{ct}, r)$ where $r \leftarrow \{0,1\}^\lambda$ and $\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 0; r)$ and replying with pk.
- $H_{b,2}(\lambda)$: In this hybrid, the challenger instead computes $\mathsf{hk} \leftarrow \mathsf{FBH.SetupBinding}(1^\lambda, L(\lambda), f_g)$ where f_g is the function from Eq. (3.1) and g is defined as follows

  $$g(\mathsf{ct}, x) := \begin{cases} 1 & \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{ct}) = 1 \wedge P^\star(x) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

  Note that the secret key $\mathsf{sk}_{\mathsf{PKE}}$ and the challenge policy $P^\star$ is hard-coded into the description of g.
- $H_{b,3}(\lambda)$: In this hybrid, instead of encrypting msg_b in the policy-selective security game without corruptions, the challenger encrypts the lexicographically-first message $\mathsf{msg}^\star \in \mathcal{M}_\lambda$: $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{mpk}, P^\star, \mathsf{msg}^\star)$.

Since the challenge bit is sampled uniformly (i.e., $b \xleftarrow{R} \{0,1\}$), the success probability of A on any security parameter λ is equal to

$$\Pr[A \text{ wins}] = \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right).$$

For each $b \in \{0,1\}$ and $i \in \{1,2,3\}$, we define

$$\delta_{b,i}(\lambda) = \Pr[H_{b,i-1}(\lambda) = 1] - \Pr[H_{b,i}(\lambda) = 1].$$

It follows by our assumption that for all $\lambda \in \Lambda$,

$$\begin{aligned} \frac{1}{2} + \frac{1}{q(\lambda)} &\leq \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right) \\ &= \frac{1}{2} \cdot \big(\Pr[H_{0,3}(\lambda) = 1] + \delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) + \delta_{0,3}(\lambda) \\ &\qquad + \Pr[H_{1,3}(\lambda) = 1] + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda) + \delta_{1,3}(\lambda)\big). \end{aligned}$$

As $H_{b,3}$ is independent of b, it must be the case that $\Pr[H_{b,3}(\lambda) = 1] \leq 1/2$ for each $b \in \{0,1\}$. Therefore, for all $\lambda \in \Lambda$,

$$\frac{1}{q(\lambda)} \leq \frac{1}{2} \cdot \left(\delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) + \delta_{0,3}(\lambda) + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda) + \delta_{1,3}(\lambda)\right).$$

As Λ is an infinitely large subset of $\mathbb{N}$, it must be the case that, for some $b \in \{0,1\}$ and $i \in \{1,2,3\}$, $\delta_{b,i}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. We show in the following claims that this must contradict one of our assumptions from the statement of Theorem 6.6. We prove the following claims for $b = 0$ without loss of generality.

Claim 6.7. *If $\delta_{0,1}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then* PKE *does not satisfy semantic security.*

Proof. Let $Q(\lambda)$ be an upper bound on the number of key-generation queries that A makes in the pre-challenge query phase of the policy-selective security game without corruptions. Note that Q is polynomially-bounded since A is efficient. We construct a stateful PPT adversary B that uses A to break the semantic security of PKE:

1. Algorithm B receives a public key $\widehat{\mathsf{pk}}$ for the PKE scheme. As PKE is a bit encryption scheme, B outputs challenge messages $\mathsf{msg}_0 = 0$ and $\mathsf{msg}_1 = 1$.

2. Algorithm B receives a challenge ciphertext $\widehat{\mathsf{ct}}$ as input. The challenger samples $\hat{b} \xleftarrow{R} \{0,1\}$ and sets $\widehat{\mathsf{ct}} \leftarrow \mathsf{PKE.Enc}(\widehat{\mathsf{pk}}, \hat{b})$.

   (a) Algorithm B computes $Q(\lambda)$ and samples a uniformly random index $i^\star \leftarrow [Q(\lambda)]$.

   (b) Algorithm B starts running A, which starts by committing to a challenge policy $P^\star$. Algorithm B then computes the public parameters $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$ as in Setup except using $\mathsf{pk}_{\mathsf{PKE}} \coloneqq \widehat{\mathsf{pk}}$. Algorithm B sends the public parameters pp to A.

   (c) For each counter value $\mathsf{ctr} \in [Q(\lambda)]$, algorithm B computes the response pk to A's key-generation queries for ctr as follows:

   - If $\mathsf{ctr} = i^\star$ for an key-generation query, B sets $\mathsf{pk} = \widehat{\mathsf{ct}}$.
   - If $\mathsf{ctr} < i^\star$ for an key-generation query, B samples $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$ and sets $\mathsf{pk} = \mathsf{PKE.Enc}(\widehat{\mathsf{pk}}, 0; \mathsf{sk})$.
   - If $\mathsf{ctr} > i^\star$ for an key-generation query, B samples $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$ and sets $\mathsf{pk} = \mathsf{PKE.Enc}(\widehat{\mathsf{pk}}, 1; \mathsf{sk})$.

   (d) In the challenge phase, algorithm B receives tuples $(c_i, x_i, \mathsf{pk}_i^\star)$ from A and constructs the challenge ciphertext $\mathsf{ct}^\star$ using the procedure described in hybrid $H_{0,0}(\lambda)$. Algorithm B gives $\mathsf{ct}^\star$ to A.

   (e) The adversary A outputs a bit $b' \in \{0,1\}$, which B also outputs.

If A is efficient, then so is B by construction. To analyze the success probability, we define a sequence of sub-hybrids. For each $i \in [0, Q(\lambda)]$, we define $H_{0,0,i}(\lambda)$ as follows:

- $H_{0,0,i}(\lambda)$: Same as $H_{0,0}(\lambda)$ except the challenger answers the first $i-1$ key-generation queries using the procedure from $H_{0,1}(\lambda)$. More precisely, when $\mathsf{ctr} < i$, the challenger sets $\mathsf{pk} = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 0; \mathsf{sk})$ for a random $\mathsf{sk} \xleftarrow{R} \{0,1\}^\lambda$ (i.e., using the same procedure as in $H_{0,1}(\lambda)$).

Note that hybrid $H_{0,0,0}$ is identical to $H_{0,0}$, and hybrid $H_{0,0,Q(\lambda)}(\lambda)$ is identical to $H_{0,1}(\lambda)$. Therefore, it holds that

$$\begin{aligned}\delta_{0,1}(\lambda) &= \Pr[H_{0,0}(\lambda) = 1] - \Pr[H_{0,1}(\lambda) = 1] \\ &= \sum_{i=1}^{Q(\lambda)} \left(\Pr[H_{0,0,i-1}(\lambda) = 1] - \Pr[H_{0,0,i}(\lambda) = 1]\right)\end{aligned}$$

We proceed to analyze the success probability of B conditioned on $i^\star$ being equal to a fixed values of i. Note that when $\hat{b} = 1$, the view of A corresponds to hybrid $H_{0,0,i}$ since the public key pk for counter value i is an encryption of 0, and when $\hat{b} = 0$, the view of A corresponds to hybrid $H_{0,0,i-1}$. Since B outputs b', it follows that if $\hat{b} = 1$, then $b' = \hat{b}$ whenever $b' = 1$, or equivalently, when $H_{0,0,i}(\lambda) = 0$ (i.e., algorithm A loses since b is fixed to 0). Similarly, if $\hat{b} = 0$, then $b' = \hat{b}$ whenever $b' = 0$, or equivalently, when $H_{0,0,i-1}(\lambda) = 1$ (i.e., A wins). The success probability of B for any $i \in [L(\lambda)]$ conditioned on $i^\star = i$ is given by the following:

$$\begin{aligned}\Pr[b' = \hat{b} \mid i^\star = i] &= \frac{1}{2} \cdot \left(1 - \Pr[H_{0,0,i}(\lambda) = 1]\right) + \frac{1}{2} \cdot \Pr[H_{0,0,i-1}(\lambda) = 1] \\ &= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[H_{0,0,i-1}(\lambda) = 1] - \Pr[H_{0,0,i}(\lambda) = 1]\right).\end{aligned}$$

It follows that the success probability of B, where B chooses a random $i^\star$, is equal to

$$\begin{aligned}
\Pr[b' = \hat{b}] &= \sum_{i=1}^{Q(\lambda)} \Pr[i^\star = i] \cdot \Pr[b' = \hat{b} \mid i^\star = i] \\
&= \frac{1}{Q(\lambda)} \cdot \sum_{i=1}^{Q(\lambda)} \left(\frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[H_{0,0,i-1}(\lambda) = 1] - \Pr[H_{0,0,i}(\lambda) = 1] \right) \right) \\
&= \frac{1}{2} + \frac{1}{2 \cdot Q(\lambda)} \cdot \sum_{i=1}^{Q(\lambda)} \left(\Pr[H_{0,0,i-1}(\lambda) = 1] - \Pr[H_{0,0,i}(\lambda) = 1] \right) \\
&= \frac{1}{2} + \frac{1}{2 \cdot Q(\lambda)} \cdot \delta_{0,1}(\lambda).
\end{aligned}$$

As $Q \in \mathsf{poly}(\lambda)$ and $\delta_{0,1}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, it follows that B succeeds with inverse polynomial probability for infinitely many $\lambda \in \mathbb{N}$, which violates the semantic security of PKE, as required. □

Claim 6.8. *If $\delta_{0,2}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then* FBH *does not satisfy computational function hiding.*

Proof. The proof of this claim follows nearly identically to the proof of Claim 4.7. □

Claim 6.9. *If $\delta_{0,3}(\lambda) \geq 1/(3q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$ and* PKE *satisfies (perfect) correctness, then either* WE *does not satisfy message indistinguishability or* FBH *does not satisfy statistical function binding.*

Proof. We construct a PPT adversary B that uses adversary A to either break the message indistinguishability security of WE, or we show that A can be used to break statistical function binding of FBH. We define B as follows:

1. Algorithm B starts running A. The adversary A sends the challenger policy $P^\star$ to B. Algorithm B computes $\mathsf{pp} = (\mathsf{pk}_{\mathsf{PKE}}, \mathsf{hk})$ according to the specification in hybrid $H_{0,2}(\lambda)$ and sends pp to A.
2. Whenever A makes a key-generation query, algorithm B responds as in hybrid $H_{0,2}(\lambda)$.
3. In the challenge phase, algorithm A outputs two messages $\mathsf{msg}'_0, \mathsf{msg}'_1$ and tuples $(c_i, x_i, \mathsf{pk}^\star_i)$ for each slot $i \in [L(\lambda)]$. Algorithm B computes public key pk_i for each slot as in hybrid $H_{0,2}(\lambda)$ and computes $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, ((\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)))$ as done in Aggregate.
4. Algorithm B outputs the statement $(\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}, P^\star)$ and a pair of messages $(\mathsf{msg}_0, \mathsf{msg}_1)$ where $\mathsf{msg}_0 = \mathsf{msg}'_0$ and $\mathsf{msg}_1 = \mathsf{msg}^\star$ is the lexicographically first message in $\mathcal{M}_\lambda$.
5. Algorithm B receives a challenge ciphertext $\mathsf{ct}^\star$ corresponding to $\mathsf{WE.Enc}(1^\lambda, \mathsf{msg}_{\hat{b}}, x)$, where $\hat{b} \xleftarrow{R} \{0,1\}$ is sampled by the challenger. Algorithm B gives $\mathsf{ct}^\star$ to A.
6. At the end of the experiment, algorithm A outputs a bit $b' \in \{0,1\}$, which B also outputs.

If A is efficient, then so is B. It suffices to analyze the success probability of B. Let $\mathsf{Wit}(\lambda)$ be the event that there exists a valid witness $(j, \mathsf{ct}, x, r, \pi)$ for the instance $(\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}, P^\star)$ such that

$$\mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 1; r) \wedge P^\star(x) = 1 \wedge \mathsf{FBH.VerOpen}(\mathsf{hk}, \mathsf{dig}, \{j\}, \{(j, (\mathsf{ct}, x))\}, \pi) = 1. \tag{6.1}$$

Note that the view of A is identical to hybrid $H_{0,2}(\lambda)$ if $\hat{b} = 0$ and is identical to $H_{0,3}(\lambda)$ if $\hat{b} = 1$ since $\mathsf{msg}_1 = \mathsf{msg}^\star$. Furthermore, when $\hat{b} = 0$, B wins whenever $b' = 0$, so $H_{0,2}(\lambda) = 1$ since b is fixed to 0. When $\hat{b} = 1$, B wins whenever $b' = 1$, so $H_{0,3}(\lambda) = 0$ since b is fixed to 0. Additionally, if $\mathsf{Wit}(\lambda)$ holds, then B always loses the message indistinguishability game since it implies that $x \in \mathcal{L}_\lambda$. Thus, the success probability of B is at least

$$\begin{aligned}
\Pr[B \text{ wins}] &\geq \Pr[\hat{b}' = \hat{b}] - \Pr[\mathsf{Wit}(\lambda)] \\
&\geq \frac{1}{2} \cdot \Pr[H_{0,2}(\lambda) = 1] + \frac{1}{2} \cdot \left(1 - \Pr[H_{0,3}(\lambda) = 1]\right) - \Pr[\mathsf{Wit}(\lambda)] \\
&= \frac{1}{2} + \frac{1}{2} \cdot \delta_{0,3}(\lambda) - \Pr[\mathsf{Wit}(\lambda)]
\end{aligned}$$

If $\Pr[\mathsf{Wit}(\lambda)] \leq 1/(12q(\lambda))$ for infinitely many $\lambda \in \Lambda$, then this violates the message indistinguishability of WE since $\delta_{0,3}(\lambda) \geq 1/(3q(\lambda))$. We next argue below that if $\Pr[\mathsf{Wit}(\lambda)] > 1/(12q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then this violates the statistical function binding of FBH, as required for the statement of the claim. Suppose $\Pr[\mathsf{Wit}(\lambda)] > 1/(12q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$ (i.e., Eq. (6.1) occurs with probability at least $1/(12q(\lambda))$). In the above construction, $\mathsf{dig} = \mathsf{FBH.Hash}(\mathsf{hk}, ((\mathsf{pk}_1, x_1) \ldots, (\mathsf{pk}_{L(\lambda)}, x_L)))$. For each $i \in [L]$, we have that

$$g(\mathsf{pk}_i, x_i) = 1 \quad \text{if and only if} \quad \mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{pk}_i) = 1 \wedge P^\star(x_i) = 1.$$

By construction of hybrid $H_{0,2}(\lambda)$, it must be the case that either:

- $i \in C$, so $P^\star(x_i) = 0$ for admissible A; or
- $i \notin C$ and corresponds to a key-generation query where $\mathsf{pk}_i \in \mathrm{Supp}\left(\mathsf{PKE.Enc}(\mathsf{pk}_{\mathsf{PKE}}, 0)\right)$. For this case, perfect correctness of PKE implies that $\mathsf{PKE.Dec}(\mathsf{sk}_{\mathsf{PKE}}, \mathsf{pk}_i) = 0$.

Thus, for all $i \in [L]$, we have that $g(\mathsf{pk}_i, x_i) = 0$, and correspondingly,

$$f_g\left((\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)\right) = \bigvee_{j \in [L]} g(\mathsf{pk}_j, x_j) = 0.$$

Therefore, if Eq. (6.1) is satisfied for some witness $(j, \mathsf{ct}, x, r, \pi)$ corresponding to instance $(\mathsf{hk}, \mathsf{pk}_{\mathsf{PKE}}, \mathsf{dig}, P^\star)$, then it must be the case that $g(\mathsf{ct}, x) = 1$. But then the output of f_g on *any* input that contains (ct, x) is necessarily $1 \neq f_g((\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$. As such, if $\mathsf{Wit}(\lambda)$ holds with probability at least $1/(12q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then there exists an adversary that break statistical function binding of FBH with the same (non-negligible) advantage. □

This proof follows by combining Claims 6.7 to 6.9. □

Remark 6.10 (Transparent Setup)**.** As as the case with the flexible broadcast encryption scheme from Section 4, the public parameters in Construction 6.4 is pseudorandom with a suitable instantiation of the function-binding hash function and public-key encryption scheme (see Remark 4.9). Correspondingly, we obtain slotted registered ABE scheme for general policies with a *transparent* setup.

Instantiation. Combining our slotted registered ABE scheme from Construction 6.4 with the generic transformation from [HLWW23] (see also Appendix A), we obtain a registered ABE scheme for general policies. We summarize this construction below:

Corollary 6.11 (Registered ABE for General Policies)**.** *Let λ be a security parameter and $\ell = \ell(\lambda)$ be an attribute length parameter. Let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of policies on ℓ-bit attributes where each policy $P \in \mathcal{P}_\lambda$ can be implemented by a Boolean circuit of size at most $s = s(\lambda)$. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ be a message space for messages of length m. Then, assuming witness encryption for* NP *and the plain learning with errors (*LWE*) assumption, there exists a registered ABE scheme for attributes of length ℓ, policy family $\mathcal{P}$, and message space $\mathcal{M}$ with the following properties:*

- *The public parameters consists of a* uniform *random string of length* $\mathrm{poly}(\lambda, s)$.
- *The running time of key generation is* $\mathrm{poly}(\lambda)$.
- *The size of the master public key and the size of the helper decryption keys is* $\mathrm{poly}(\lambda, s, \log L)$ *where L is the number of registered users.*
- *Both the encryption and decryption algorithms run in time* $\mathrm{poly}(\lambda, s, m, \log L)$.

The scheme satisfies policy-selective security without corruptions. In the random oracle model, we can further amplify the scheme to satisfy policy-selective security with corruptions (see Appendix C).

Remark 6.12 (Comparison with [HLWW23])**.** Corollary 6.11 is the first registered ABE scheme that supports general policies that does not rely on indistinguishability obfuscation. The parameter sizes of Corollary 6.11 are consistent with those obtained from the obfuscation-based construction from [HLWW23]. However, the obfuscation-based construction satisfies *adaptive* security whereas Corollary 6.11 only provides policy-selective security without corruptions. To conclude, we also compare Corollary 6.11 with the pairing-based construction from [HLWW23]:

- **Setup assumptions:** The pairing-based construction relies on a *structured* public parameters whose size scales *quadratically* with the maximum number of registered users. Thus, [HLWW23] requires a trusted setup to generate the public parameters. The public parameters in Corollary 6.11 consist of a uniform random string and thus, the scheme supports a *transparent* setup.
- **Unbounded users:** The pairing-based construction imposes an a priori polynomial bound on the maximum number of registered users L in the system, and moreover, the size of the public parameters, the running time of key generation, and the running time of registration scales with the bound L. Corollary 6.11 supports an arbitrary number of users (e.g., an implicit bound of 2^λ users).
- **Policy family:** The pairing-based construction supports monotone Boolean formulas whereas Corollary 6.11 supports arbitrary circuit policies.
- **Security:** The pairing-based construction follows a dual-system methodology and satisfies adaptive security. In contrast, Corollary 6.11 satisfies the weaker notion of policy-selective security without corruptions (and with corruptions if we work in the random oracle model).

7 Flexible and Distributed Broadcast Encryption from Registered ABE

In this section, we show that it is also possible to *generically* obtain either a flexible broadcast encryption scheme or a distributed broadcast encryption scheme from a registered ABE scheme (defined in Appendix A). For instance, an alternative approach to obtaining flexible broadcast encryption is to apply our transformation to the registered ABE scheme from Section 6. We do note however that the scheme obtained via our generic transformation satisfies a weaker notion of *static* security (Definition 7.3) as opposed to the notion of semi-static security satisfied by Construction 4.3. We can boost semi-static security to adaptive security (see Appendix B) in the random oracle model, but an analogous notion is not known starting from only a statically-secure scheme.

We also note that we can apply our transformation to the previous pairing-based registered ABE scheme from [HLWW23]. For reasons that we discuss below (see Remark 7.6), our transformation in the pairing-based setting yields the more restricted notion of *distributed* broadcast encryption where each user's public key must be associated with a unique index.

Transformation overview. As mentioned in Section 1.1, our transformation only requires the registered ABE scheme to support a single attribute and a single "always-accept" policy. The idea is simple:

- The public parameters for the flexible broadcast encryption scheme is simple the public parameters for the registered ABE scheme.
- To join the system, a user generates a public key for the underlying registered ABE scheme.
- To encrypt to a collection of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$, the encrypter simulates the role of the key curator for the underlying registered ABE scheme and derives the master public key mpk associated with registering public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_k$ (with the dummy attribute). Then, the user encrypts the message with respect to mpk and the "always-accept" policy.
- To decrypt a ciphertext encrypted to $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$ using a secret key sk_i for some $i \in [k]$, the decrypter runs the same registration algorithm on the keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$ and uses it to compute the helper decryption key hsk_i for user i. Using its own secret key sk_i together with hsk_i, the user can now run the decryption algorithm for the registered ABE scheme to recover the message.

Critically, the above transformation requires that users can generate their public keys in the registered ABE scheme *independently* and without coordination. This is the notion satisfied by our registered ABE scheme from Section 6. In some registered ABE schemes such as the pairing-based construction of [HLWW23], user public keys are associated with an index, and functionality requires that different users register public keys to *distinct* indices. In this case, the above transformation is still applicable, except when users join the system, they must generate their key to a specific index and one can only broadcast to a set of public keys where each public key occupies a different slot. This corresponds to the notion of *distributed* broadcast encryption [BZ14]. We provide additional discussion in Remark 7.6.

Construction 7.1 (Flexible Broadcast Encryption from Registered ABE)**.** Let $m = m(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space with messages of length m. Let $\mathsf{rABE} = (\mathsf{rABE.Setup}, \mathsf{rABE.KeyGen}, \mathsf{rABE.RegPK}, \mathsf{rABE.Enc}, \mathsf{rABE.Update}, \mathsf{rABE.Dec})$ be a registered ABE scheme for 1-bit attributes and the policy family $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ where $\mathcal{P}_\lambda = \{P_{\mathsf{on}}\}$ and $P_{\mathsf{on}} : \{0,1\} \to \{1\}$ is the "always-accept" policy. We additionally require that rABE support *stateless* key generation (Remark A.5). We construct a flexible broadcast encryption scheme $\mathsf{FBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with message space $\mathcal{M}$ as follows:

- $\mathsf{Setup}(1^\lambda, n)$: On input the security parameter λ and a bound on the number of recipients n, the setup algorithm samples and outputs $\mathsf{pp} \leftarrow \mathsf{rABE.Setup}(1^\lambda)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters pp, the key-generation algorithm samples and outputs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{rABE.KeyGen}(\mathsf{pp})$. Here, we assume that rABE supports stateless key generation (Remark A.5) so rABE.KeyGen only takes pp as input.
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$: On input the public parameters pp, a message $\mathsf{msg} \in \mathcal{M}_\lambda$, and a list of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$, the encryption algorithm does the following:
  - Initializes an empty master public key and auxiliary input $(\mathsf{mpk}, \mathsf{aux}) := (\bot, \bot)$.
  - It now registers each public key pk_i with the dummy attribute 1. Specifically, for each $i \in [k]$, it computes $(\mathsf{mpk}, \mathsf{aux}) := (\mathsf{mpk}', \mathsf{aux}') = \mathsf{rABE.RegPK}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}_i, 1)$.
  - Finally, it encrypts the message with respect to the aggregated master public key and the "always-accept" policy: $\mathsf{ct} \leftarrow \mathsf{rABE.Enc}(\mathsf{mpk}, P_{\mathsf{on}}, \mathsf{msg})$.
- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$: On input the public parameters pp, a ciphertext ct, an index/secret-key pair (j, sk_j), and a list of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$, the decryption algorithm does the following:
  - Initializes an empty master public key and auxiliary input $(\mathsf{mpk}, \mathsf{aux}) := (\bot, \bot)$. Like Enc, the decryption algorithm now registers each public key pk_i with the dummy attribute 1. Specifically, for each $i \in [k]$, it computes $(\mathsf{mpk}, \mathsf{aux}) := (\mathsf{mpk}', \mathsf{aux}') = \mathsf{rABE.RegPK}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}_i, 1)$.
  - Compute the helper decryption key $\mathsf{hsk} = \mathsf{rABE.Update}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}_j)$ for public key pk_j.
  - Output $\mathsf{msg} = \mathsf{rABE.Dec}(\mathsf{mpk}, \mathsf{hsk}, \mathsf{sk}_j, \mathsf{ct})$.

Correctness and efficiency. We now show Construction 7.1 is correct and satisfies the efficiency requirements on a flexible broadcast encryption scheme.

Theorem 7.2 (Correctness and Efficiency)**.** *Assuming* rABE *is efficient, compact, and correct, then Construction 7.1 has succinct ciphertexts and is correct.*

Proof. We consider each property separately:

- **Succinct ciphertexts:** Consider a master public key mpk and ciphertext ct constructed by the encryption algorithm Enc. By compactness of rABE, we have that $|\mathsf{mpk}| \in \mathsf{poly}(\lambda, \ell, \log k)$. Since $\ell = 1$ and $|P_{\mathsf{on}}| \in O(1)$, it follows that $|\mathsf{ct}| \in \mathsf{poly}(\lambda, m(\lambda), \log k)$, as required.
- **Correctness:** Take any $\lambda \in \mathbb{N}$, $n \in \mathsf{poly}(\lambda)$, $k \le n(\lambda)$, $j \in [k]$, and $\mathsf{msg} \in \mathcal{M}_\lambda$. Then, we have the following:

- Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda))$, and for $i \in [k]$, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. By construction, pp and $(\mathsf{pk}_i, \mathsf{sk}_i)$ are sampled from $\mathsf{rABE.Setup}(1^\lambda)$ and $\mathsf{rABE.KeyGen}(\mathsf{pp})$, respectively.
- Let $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$. By construction, it follows that $\mathsf{ct} \leftarrow \mathsf{rABE.Enc}(\mathsf{mpk}, P_{\mathsf{on}}, \mathsf{msg})$ where mpk is derived from a series of calls to RegPK with each public key pk_i with the dummy attribute 1.
- We now argue that $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)) = \mathsf{msg}$. As rABE.RegPK is deterministic, Enc and Dec compute the same master public key mpk. As a result, if Dec does not output msg, that violates the correctness of rABE. So, there exists a negligible function negl such that Dec output msg with probability at least $1 - \mathsf{negl}(\lambda)$. Moreover, if rABE satisfies perfect correctness, then so does FBE. □

Static security. The flexible broadcast scheme in Construction 7.1 satisfies a weaker notion of *static* security, where the adversary has to commit to the exact set of keys in the challenge ciphertext; this is the analog of selective security in the context of traditional broadcast encryption. Static security is a weaker property than semi-static security (Definition 4.2) where the adversary is able to choose which subset of honestly-generated keys it can use in the challenge ciphertext. We formally define static security for flexible broadcast encryption and then show that Construction 7.1 is statically secure in Theorem 7.4.

Definition 7.3 (Static Security)**.** We say that a flexible broadcast encryption scheme (Setup, KeyGen, Enc, Dec) satisfies static security if instead of adaptive security, it satisfies the following:

- **Static security:** For all stateful PPT adversaries A and all efficiently-computable functions $n \in \mathsf{poly}(\lambda)$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Expt}^{\mathsf{FBE,Static}}_{A,n}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

  where $\mathsf{Expt}^{\mathsf{FBE,Static}}_{A,n}(\lambda)$ is defined via the following security game between the adversary A and a challenger on common input 1^λ:

  - **Setup phase:** The adversary starts by choosing an integer $k \leq n$. The challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n)$ and samples key-pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for each $i \in [k]$. It gives pp and $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$ to A.
  - **Challenge phase:** Algorithm A outputs two messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}_\lambda$. The challenger samples a bit $b \xleftarrow{R} \{0,1\}$ and computes an encryption of msg_b under the set $S^\star = (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$: $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}_b, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$. The challenger gives $\mathsf{ct}^\star$ to A.
  - **Output phase:** Algorithm A outputs a bit $b' \in \{0,1\}$. The experiment outputs 1 if $b' = b$ and 0 otherwise.

Theorem 7.4 (Static Security)**.** *Assuming* rABE *satisfies policy-selective security without corruption queries, then Construction 7.1 satisfies static security.*

Proof. Suppose by way of contradiction that FBE does not satisfy static security. Namely, there exists a stateful PPT adversary A, an efficiently computable function $n \in \mathsf{poly}(\lambda)$, and a polynomial q such that for infinitely many $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\mathsf{Expt}^{\mathsf{FBE,Static}}_{A,n(\lambda)}(\lambda) = 1\right] > 1/2 + 1/q(\lambda).$$

We construct an adversary B that breaks policy-selective security without corruption queries of rABE:

- **Setup phase:** Algorithm B sends the unique challenge policy $P^\star = P_{\mathsf{on}} \in \mathcal{P}_\lambda$ to the rABE challenger. Algorithm B then receives the public parameters pp from the challenger, which B then forwards to A. The adversary A specifies a number of keys $k \leq n(\lambda)$ that it will generate.
- **Pre-challenge query phase:** The adversary A makes k key-generation queries. On the i^{th} query, algorithm B makes a corresponding register-honest-key query to the rABE challenger for attribute set 1 to obtain a public key pk_i. Algorithm B sends pk_i to A.

- **Challenge phase:** In the challenge phase, A sends two messages $\mathsf{msg}_0, \mathsf{msg}_1$. Algorithm B sends the same messages $\mathsf{msg}_0, \mathsf{msg}_1$ to its challenger. The challenger samples a bit $b \xleftarrow{R} \{0,1\}$ and replies with the challenge ciphertext $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{mpk}, P^\star, \mathsf{msg}_b)$, which B forwards to A.
- **Output phase:** The adversary A outputs a guess b' for b, which B outputs.

If A is efficient, then so is B. To analyze the success probability of B, we observe that by construction of B and the definition of Enc, the challenge ciphertext $\mathsf{ct}^\star$ defined above is distributed exactly as the challenge ciphertext in the flexible broadcast encryption static security game. Specifically, this is because mpk is the result of registering keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_k$ in order, where each public key is registered with the attribute 1, and $P^\star = P_{\mathsf{on}}$ is the only allowable policy. Therefore, if A succeeds in guessing $b' = b$ with probability $1/2 + 1/q(\lambda)$, then so does B. □

Remark 7.5 (On Adaptively-Secure Flexible Broadcast from Registered ABE)**.** While we only know how to prove static security for Construction 7.1, we believe this is mostly a caveat of the registered ABE definition from [HLWW23]. Namely, the [HLWW23] security definition does not allow for more adaptive capabilities where the adversary can request that the challenger generate (honest) public keys and then *adaptively* choose which public keys it would like to register. It is interesting to properly define such a stronger notion, and it seems plausible that existing constructions can be adapted to satisfy this stronger property. We leave further exploration of the security notions underlying registered ABE to future work.

Remark 7.6 (Distributed Broadcast Encryption from Pairings)**.** Correctness of Construction 7.1 critically relies on the underlying registered ABE scheme supporting *stateless* key-generation (Remark A.5). This is because the subset of keys that are registered is determined at *encryption* time (rather than key-generation time); moreover, the scheme needs to support registering to any subset of the user-generated keys. Thus, if the scheme requires that users generate keys in a stateful manner, where each key can depend arbitrarily on the set of previously-registered keys, then Construction 7.1 is no longer correct. While Construction 6.4 (in conjunction with Theorem A.4) yields a registered ABE with stateless key generation, the previous pairing-based constructions of registered ABE [HLWW23] have a *stateful* key-generation procedure, and thus Construction 7.1 cannot be directly applied to [HLWW23] to obtain a pairing-based flexible broadcast encryption scheme.

Nonetheless, we observe that the previous pairing-based registered ABE scheme of [HLWW23] has the property that key generation does not depend *arbitrarily* on the auxiliary input. Instead, the key-generation algorithm only demands knowledge of an index, with the stipulation that every user has a distinct index. For instance, the index could be the current count on the number of registered users in the system. This means we can apply Construction 7.1 to the pairing-based registered ABE scheme to obtain a *distributed* broadcast encryption scheme [BZ14]. In a distributed broadcast encryption scheme, each user is associated with a unique index and when users generate their key, they generate it with respect to their particular index. This coincides with the semantics of the pairing-based registered ABE scheme from [HLWW23]. Thus, combining Construction 7.1 with the construction of [HLWW23], we obtain the first distributed broadcast encryption scheme using composite-order bilinear groups. The resulting construction supports an a priori bounded number of users and requires a structured reference string of size $n^2 \cdot \mathsf{polylog}(n)$, where n is the bound on the number of users. The scheme satisfies static security (analogous to Definition 7.3 and Theorem 7.4).

Acknowledgments

We thank Dan Boneh and Hoeteck Wee for helpful pointers on broadcast encryption. Cody Freitag's work was done while at Cornell Tech, and he is supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE–2139899, DARPA Award HR00110C0086, AFOSR Award FA9550-18-1-0267, NSF CNS-2128519, and DARPA under Agreement No. HR00112020023. Brent Waters is supported by NSF CNS-1908611, a Simons Investigator award, and the Packard Foundation Fellowship. David J. Wu is supported by NSF CNS-2151131, CNS-2140975, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[AL18] Prabhanjan Ananth and Alex Lombardi. Succinct garbling schemes from functional encryption through a local simulation paradigm. In *TCC*, 2018.

[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.

[BBK+23] Zvika Brakerski, Maya Farber Brodsky, Yael Tauman Kalai, Alex Lombardi, and Omer Paneth. SNARGs for monotone policy batch NP. In *CRYPTO*, 2023.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BGI+17] Saikrishna Badrinarayanan, Sanjam Garg, Yuval Ishai, Amit Sahai, and Akshay Wadia. Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In *ASIACRYPT*, 2017.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

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

[BIOW20] Ohad Barta, Yuval Ishai, Rafail Ostrovsky, and David J. Wu. On succinct arguments and witness encryption from groups. In *CRYPTO*, 2020.

[BISW18] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal SNARGs via linear multi-prover interactive proofs. In *EUROCRYPT*, 2018.

[BJK+18] Zvika Brakerski, Aayush Jain, Ilan Komargodski, Alain Passelègue, and Daniel Wichs. Non-trivial witness encryption and null-iO from standard assumptions. In *SCN*, 2018.

[BPR15] Nir Bitansky, Omer Paneth, and Alon Rosen. On the cryptographic hardness of finding a Nash equilibrium. In *FOCS*, 2015.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, 2012.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, 2011.

[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.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[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.

[CES21] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In *IMACC*, 2021.

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

[CH85] Stephen A. Cook and H. James Hoover. A depth-universal circuit. *SIAM J. Comput.*, 14(4), 1985.

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

[CLP15] Kai-Min Chung, Huijia Lin, and Rafael Pass. Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In *CRYPTO*, 2015.

[CPP20] Ran Canetti, Sunoo Park, and Oxana Poburinnaya. Fully deniable interactive encryption. In *CRYPTO*, 2020.

[CVW18] Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In *CRYPTO*, 2018.

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

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Sparks: Succinct parallelizable arguments of knowledge. In *EUROCRYPT*, 2020.

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

[FNV17] Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Predictable arguments of knowledge. In *PKC*, 2017.

[Gam84] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1984.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, USA, 2009.

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, 2014.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *PKC*, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC*, 2018.

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In *ACM CCS*, 2023.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO*, 2013.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *FOCS*, 2017.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In *CRYPTO*, 2014.

[GMM17] Sanjam Garg, Mohammad Mahmoody, and Ameer Mohammed. Lower bounds on obfuscation from all-or-nothing encryption primitives. In *CRYPTO*, 2017.

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

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, 2006.

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

[GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In *CRYPTO*, 2020.

[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.

[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.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, 2015.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, 2021.

[KRR17] Yael Tauman Kalai, Guy N. Rothblum, and Ron D. Rothblum. From obfuscation to the security of fiat-shamir for proofs. In *CRYPTO*, 2017.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, 1989.

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

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

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

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

[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.

[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.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *FOCS*, 2017.

A Registered Attributed-Based Encryption

In this section, we recall the full notion of registered attribute-based encryption (ABE) from [HLWW23]. For simplicity of notation, we describe our notion for the setting where the attribute is an arbitrary bit-string.

Definition A.1 (Registered Attributed-Based Encryption Syntax [HLWW23, adapted])**.** Let $\ell = \ell(\lambda)$ be an attribute length and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies on ℓ-bit inputs (i.e., $\mathcal{P}_\lambda$ is a set of functions $P\colon \{0,1\}^{\ell(\lambda)} \to \{0,1\}$). Let $m = m(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space with message length m. A *registered attributed-based encryption scheme* for attributes of length ℓ, policy space $\mathcal{P}$, and message space $\mathcal{M}$ consists of polynomial-time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Enc}, \mathsf{Update}, \mathsf{Dec})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: A probabilistic algorithm that on input the security parameter λ, outputs the public parameters pp. We implicitly assume that pp contains 1^λ.
- $\mathsf{KeyGen}(\mathsf{pp}, \mathsf{aux}) \to (\mathsf{pk}, \mathsf{sk})$: A probabilistic algorithm that on input the public parameters pp and a (possibly empty) auxiliary state aux, outputs a public key pk and a secret key sk.
- $\mathsf{RegPK}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}, x) \to (\mathsf{mpk}, \mathsf{aux}')$: A deterministic algorithm that on input the public parameters pp, a (possibly empty) auxiliary state aux, a public key pk, and an attribute $x \in \{0,1\}^{\ell(\lambda)}$, outputs a master public key mpk and an updated state aux'.
- $\mathsf{Enc}(\mathsf{mpk}, P, \mathsf{msg}) \to \mathsf{ct}$: A probabilistic algorithm that on input a master public key mpk, a policy $P \in \mathcal{P}_\lambda$, and a message $\mathsf{msg} \in \mathcal{M}_\lambda$, outputs a ciphertext ct.
- $\mathsf{Update}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}) \to \mathsf{hsk}$: A deterministic algorithm that on input the public parameters pp, a (possibly empty) auxiliary state aux, and a public key pk, outputs a helper decryption key hsk.
- $\mathsf{Dec}(\mathsf{mpk}, \mathsf{hsk}, \mathsf{sk}, \mathsf{ct}) \to \mathsf{msg}$: A deterministic algorithm that on input a master public key mpk, a helper decryption key hsk, a secret key sk, and a ciphertext ct, outputs either (1) a message $\mathsf{msg} \in \mathcal{M}_\lambda$; (2) a special symbol $\perp$ indicating decryption failure; or (3) a special flag GetUpdate that indicates an updated helper decryption key is needed to decrypt.

Definition A.2 (Registered ABE Correctness and Efficiency)**.** Let $\mathsf{rABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Enc}, \mathsf{Update}, \mathsf{Dec})$ be a registered ABE scheme for attributes of length $\ell = \ell(\lambda)$, policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$, and message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$. For a security parameter λ and a stateful adversary A, we define the following experiment between A and a challenger on common input 1^λ:

- **Setup phase:** The challenger samples public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. It initializes its auxiliary state $\mathsf{aux} := \perp$ and a master public key $\mathsf{mpk} := \perp$. It also initializes a counter $\mathsf{ctr}[\mathsf{reg}] := 0$ to keep track of the number of registration queries and another counter $\mathsf{ctr}[\mathsf{enc}] := 0$ to keep track of the number of encryption queries. Finally, it initializes $\mathsf{ctr}[\mathsf{reg}]^\star := \infty$ as the index for the target key (which will be updated during the course of the experiment). The challenger sends pp to A.

- **Query phase:** During the query phase, the adversary A is able to make the following queries:
  - **Register non-target key query:** In a non-target key-registration query, A specifies a public key pk and an attribute $x \in \{0,1\}^\ell$. The challenger increments $\mathsf{ctr}[\mathsf{reg}] := \mathsf{ctr}[\mathsf{reg}] + 1$ and registers the key by computing $(\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}, x)$. The challenger updates its auxiliary data by setting $\mathsf{aux} := \mathsf{aux}'$ and replies to A with $(\mathsf{ctr}[\mathsf{reg}], \mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux})$.
  - **Register target key query:** In a target-key registration query, the adversary specifies an attribute $x^\star \in \{0,1\}^\ell$. If the adversary has previously made a target-key registration query, then the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr}[\mathsf{reg}] := \mathsf{ctr}[\mathsf{reg}] + 1$, samples $(\mathsf{pk}^\star, \mathsf{sk}^\star) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{aux})$, and registers $(\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}^\star, x^\star)$. It computes the helper decryption key $\mathsf{hsk}^\star \leftarrow \mathsf{Update}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}^\star)$. The challenger updates its auxiliary data by setting $\mathsf{aux} := \mathsf{aux}'$, stores the index of the target identity $\mathsf{ctr}[\mathsf{reg}]^\star := \mathsf{ctr}[\mathsf{reg}]$, and replies to A with $(\mathsf{ctr}[\mathsf{reg}], \mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}, \mathsf{pk}^\star, \mathsf{hsk}^\star, \mathsf{sk}^\star)$.
  - **Encryption query:** In an encryption query, A submits an index $i \in [\mathsf{ctr}[\mathsf{reg}]^\star, \mathsf{ctr}[\mathsf{reg}]]$ for a public key, a message $\mathsf{msg}_{\mathsf{ctr}[\mathsf{enc}]} \in \mathcal{M}_\lambda$, and a policy $P_{\mathsf{ctr}[\mathsf{enc}]} \in \mathcal{P}_\lambda$. If the adversary has not yet registered a target key, or if the target set of attributes $x^\star$ do not satisfy the policy $P_{\mathsf{ctr}[\mathsf{enc}]}$, the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr}[\mathsf{enc}] := \mathsf{ctr}[\mathsf{enc}] + 1$ and computes $\mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}]} \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, P_{\mathsf{ctr}[\mathsf{enc}]}, \mathsf{msg}_{\mathsf{ctr}[\mathsf{enc}]})$. The challenger replies to A with $(\mathsf{ctr}[\mathsf{enc}], \mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}]})$.
  - **Decryption query:** In a decryption query, the adversary submits a ciphertext index $j \in [\mathsf{ctr}[\mathsf{enc}]]$. The challenger computes $\mathsf{msg}'_j \leftarrow \mathsf{Dec}(\mathsf{mpk}, \mathsf{sk}^\star, \mathsf{hsk}^\star, \mathsf{ct}_j)$. If $\mathsf{msg}'_j = \mathsf{GetUpdate}$, then the challenger computes an updated helper decryption key $\mathsf{hsk}^\star \leftarrow \mathsf{Update}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}^\star)$ and recomputes $\mathsf{msg}'_j \leftarrow \mathsf{Dec}(\mathsf{mpk}, \mathsf{hsk}^\star, \mathsf{ct}_j)$. If $\mathsf{msg}'_j \neq \mathsf{msg}_j$, the experiment halts and outputs 0.

  If the adversary has finished making queries and the experiment has not halted (as a result of a decryption query), then the experiment outputs the bit $b = 1$. We require the following properties for all (potentially unbounded) adversaries A making at most a polynomial number of queries:
  - **Correctness:** There exists a negligible function negl such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] \geq 1 - \mathsf{negl}(\lambda)$ in the above game. We say that the scheme satisfies perfect correctness if $\Pr[b = 1] = 1$.
  - **Compactness:** Let N be the number of registration queries the adversary makes in the above game. There exists a university polynomial $p(\cdot,\cdot,\cdot)$ such that for all $i \in [N]$, $|\mathsf{mpk}_i| \leq p(\lambda, \ell(\lambda), \log i)$. We also require that the size of the helper decryption key $\mathsf{hsk}^\star$ satisfy $|\mathsf{hsk}^\star| \leq p(\lambda, \ell(\lambda), \log N)$ at all times during the experiment.
  - **Update efficiency:** Let N be the number of registration queries the adversary makes in the above game. Then, in the course of the above game, the challenger invokes the update algorithm Update at most $O(\log N)$ times, where each invocation runs in time at most $\mathrm{poly}(\log N)$ in the RAM model of computation. Specifically, we model Update as a RAM program that has random access to its input, so the running time of Update may be smaller than the input length.

Definition A.3 (Registered ABE Security)**.** Let $\mathsf{rABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Enc}, \mathsf{Update}, \mathsf{Dec})$ be a registered ABE scheme for attributes of length $\ell = \ell(\lambda)$, policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$, and message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$. We say that rABE satisfies *adaptive security* if for all stateful PPT adversaries A, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}$

$$\Pr\left[\mathsf{Expt}_A^{\mathsf{rABE}}(\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda),$$

where $\mathsf{Expt}_A^{\mathsf{rABE}}(\lambda)$ is defined via the following security game between the adversary A and a challenger on common input 1^λ:

- **Setup phase:** The challenger samples the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$. In then initializes the auxiliary input $\mathsf{aux} := \perp$, a master public key $\mathsf{mpk} := \perp$, a counter $\mathsf{ctr} := 0$ for the number of honest key-registration queries the adversary has made, an empty set of keys $C = \varnothing$ (to keep track of corrupted public keys), and an empty dictionary mapping public keys to registered attribute sets D. The challenger sends pp to A.

- **Query phase:** The adversary A can now issue the following queries:
  - **Register honest key query:** In an honest key-generation query, the adversary specifies an attribute $x \in \{0,1\}^\ell$. The challenger increments $\mathsf{ctr} := \mathsf{ctr} + 1$, samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, \mathsf{aux})$, and registers $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}, x)$. The challenger updates its public key $\mathsf{mpk} := \mathsf{mpk}'$, its auxiliary data $\mathsf{aux} := \mathsf{aux}'$, and $\mathsf{D}[\mathsf{pk}] := \mathsf{D}[\mathsf{pk}] \cup \{x\}$. The challenger replies to A with $(\mathsf{ctr}, \mathsf{mpk}, \mathsf{aux}, \mathsf{pk})$.
  - **Corrupt honest key query:** In a corrupt-honest-key query, the adversary specifies an index $i \in [\mathsf{ctr}]$. Let $(\mathsf{pk}_i, \mathsf{sk}_i)$ be the i^{th} public/secret key query the challenger samples when responding to the i^{th} honest key-registration query. The challenger adds pk_i to C and replies to A with sk_i.
  - **Register corrupted key query:** In a corrupted key-registration query, the adversary A specifies a public key pk and an attribute $x \in \{0,1\}^\ell$. The challenger registers the key by computing $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{pp}, \mathsf{aux}, \mathsf{pk}, x)$. The challenger updates its copy of the public key $\mathsf{mpk} := \mathsf{mpk}'$, its auxiliary data $\mathsf{aux} := \mathsf{aux}'$, adds pk to C, and updates $\mathsf{D}[\mathsf{pk}] \leftarrow \mathsf{D}[\mathsf{pk}] \cup \{x\}$. It replies to A with $(\mathsf{mpk}, \mathsf{aux})$.
- **Challenge phase:** The adversary A sends two messages $\mathsf{msg}_0, \mathsf{msg}_1 \in \mathcal{M}_\lambda$ and an access policy $P^\star \in \mathcal{P}_\lambda$. The challenger samples a bit $b \xleftarrow{R} \{0,1\}$, known as the challenge bit, and replies with the challenge ciphertext $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{mpk}, P^\star, \mathsf{msg}_b)$.
- **Output phase:** Let $\mathcal{X} = \{x \in \mathsf{D}[\mathsf{pk}] : \mathsf{pk} \in C\}$ be the set of corrupted attributes. We say that an adversary A is admissible if the challenger policy $P^\star(x) = 0$ for all $x \in \mathcal{X}$. The experiment outputs 0 if A is not admissible. Finally, A outputs a bit b' and the experiment outputs 1 if $b' = b$ and 0 otherwise.

Selective security. Similar to the setting of slotted registered ABE (Definitions 6.2 and 6.3), we will also the following selective notions of security:

- **Policy-selective security:** This is the analog of Definition 6.2. In particular, in the policy-selective security game, $\mathsf{Expt}_A^{\mathsf{rABE}}(\lambda)$ is modified so that the adversary A commits to its challenge policy $P^\star$ during the setup phase before it receives the public parameters pp.
- **Policy-selective security without corruptions:** This is the analog of Definition 6.3, where in addition to committing to the challenge policy $P^\star$ at the beginning of the security game, the adversary is also *not* allowed to make any corrupt-honest-key queries.

From slotted registered ABE to registered ABE. Previously [HLWW23] showed how to construct a full (non-slotted) registered ABE from any slotted registered ABE scheme. The transformation preserves the policy family of the underlying scheme and only incurs overhead that is logarithmic in the number of registered users. We state the theorem below:

Theorem A.4 (Registered ABE from Slotted Registered ABE [HLWW23, §6])**.** *Suppose there exists a slotted registered ABE scheme* srABE *for attributes of length $\ell = \ell(\lambda)$, policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$, and message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$. Then, there exists a registered ABE scheme* rABE *for attributes of the same length $\ell = \ell(\lambda)$, policy space $\mathcal{P}$, and message space $\mathcal{M}$. If* srABE *satisfies adaptive security (resp., policy-selective security or policy-selective security without corruptions), then* rABE *satisfies adaptive security (resp., policy-selective security or policy-selective security without corruptions).*

Remark A.5 (Stateless vs. Stateful Key Generation)**.** The key-generation algorithm Definition A.1 is allowed to depend on the auxiliary input aux. We can define a stronger notion of registered ABE where the key-generation algorithm is *independent* of the auxiliary data. We refer to such schemes as supporting *stateless* key generation; namely, users in such schemes can generate their keys *independently* of existing users. Earlier works on registration-based encryption (i.e., registered identity-based encryption) all considered stateless key generation [GHMR18, GHM+19].

On the contrary, we refer to schemes where the key-generation algorithm can depend on the auxiliary input aux (as in Definition A.1) as supporting *stateful* key generation. In the generic compilation from slotted registered ABE to registered ABE from [HLWW23], users have to know the current number of registered users when generating their keys; as such, the compiler produces a registered ABE scheme with a *stateful* key-generation procedure. However, we

observe that if the underlying slotted registered ABE scheme srABE has a slot-independent key-generation algorithm (i.e., KeyGen in srABE does not depend on the slot index i), then the [HLWW23] compiler (i.e., Theorem A.4) yields a registered ABE scheme with stateless key generation. Stateless key generation is necessary for using a registered ABE scheme to obtain a flexible broadcast encryption (Section 7).

B Flexible Broadcast Encryption with Adaptive Security

In this section, we show how to apply the Gentry-Waters transformation [GW09] to upgrade a flexible broadcast encryption scheme with semi-static security into one that satisfies full adaptive security in the random oracle model. The transformation from [GW09] applied to traditional broadcast encryption, but the underlying two-key technique directly applies to the setting of flexible broadcast encryption.

Construction B.1 (Adaptively-Secure Flexible Broadcast)**.** Let $\lambda \in \mathbb{N}$ be a security parameter. Let $m = m(\lambda)$ be any function and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space with messages of length m. Let n be the number of users the scheme supports. Our construction relies on the following primitives:

- Let $\mathsf{FBE}_{\mathsf{SS}} = (\mathsf{FBE}_{\mathsf{SS}}.\mathsf{Setup}, \mathsf{FBE}_{\mathsf{SS}}.\mathsf{KeyGen}, \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Enc}, \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Dec})$ be a flexible broadcast encryption scheme with message space $\mathcal{M}$ that satisfies semi-static security.
- Let $O\colon \{0,1\}^\lambda \to \{0,1\}^n$ be a hash function (that we will model as a random oracle in the security analysis).

We construct an (adaptively-secure) flexible broadcast encryption scheme $\mathsf{FBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda, n)$: On input the security parameter λ and the number of users n, output $\mathsf{pp} \leftarrow \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Setup}(1^\lambda, n)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters pp, sample two keys $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{FBE}_{\mathsf{SS}}.\mathsf{KeyGen}(\mathsf{pp})$ and $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{FBE}_{\mathsf{SS}}.\mathsf{KeyGen}(\mathsf{pp})$ for the underlying flexible broadcast encryption scheme. Sample a random bit $a \xleftarrow{\mathrm{R}} \{0,1\}$ and output $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1)$ and $\mathsf{sk} = (a, \mathsf{sk}_a)$.
- $\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$: On input the public parameters pp, the message $\mathsf{msg} \in \mathcal{M}_\lambda$, and a tuple of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$, where each $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$, the encryption algorithm samples a random $r \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ and computes $(t_1, \ldots, t_n) = O(r) \in \{0,1\}^n$. Next, it constructs two ciphertexts

$$\mathsf{ct}_0 \leftarrow \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Enc}\big(\mathsf{pp}, \mathsf{msg}, \big(\mathsf{pk}_{1,t_1}, \ldots, \mathsf{pk}_{k,t_k}\big)\big)$$
$$\mathsf{ct}_1 \leftarrow \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Enc}\big(\mathsf{pp}, \mathsf{msg}, \big(\mathsf{pk}_{1,1-t_1}, \ldots, \mathsf{pk}_{k,1-t_k}\big)\big),$$

  and outputs the ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, r)$.
- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$: On input the public parameters pp, the ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, r)$, the index j, the secret key $\mathsf{sk}_j = (a, \mathsf{sk})$, and a tuple of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_k)$ where each $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$, the decryption algorithm starts by computing $(t_1, \ldots, t_n) = O(r) \in \{0,1\}^n$.
  - If $a = t_j$, the decryption algorithm outputs $\mathsf{msg} = \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Dec}\big(\mathsf{pp}, \mathsf{ct}_0, (j, \mathsf{sk}), \big(\mathsf{pk}_{1,t_1}, \ldots, \mathsf{pk}_{k,t_k}\big)\big)$.
  - If $a = 1 - t_j$, the decryption algorithm outputs $\mathsf{msg} = \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Dec}\big(\mathsf{pp}, \mathsf{ct}_1, (j, \mathsf{sk}), \big(\mathsf{pk}_{1,1-t_1}, \ldots, \mathsf{pk}_{k,1-t_k}\big)\big)$.

Correctness and security analysis. We now show that FBE of Construction B.1 is a correct flexible broadcast encryption scheme that satisfies adaptive security.

Theorem B.2 (Correctness and Efficiency)**.** *Assuming* $\mathsf{FBE}_{\mathsf{SS}}$ *has succinct ciphertexts and is correct, then Construction B.1 has succinct ciphertexts and is correct.*

Proof. We consider each property separately:

- **Succinct ciphertexts:** Consider a ciphertext ct output by the encryption algorithm. By construction, $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, r)$ where $\mathsf{ct}_0, \mathsf{ct}_1$ are ciphertexts for $\mathsf{FBE}_{\mathsf{SS}}$ and $|r| = \lambda$ by construction. Thus, if the underlying flexible broadcast encryption scheme $\mathsf{FBE}_{\mathsf{SS}}$ has succinct ciphertexts, then so does FBE.

- **Correctness:** Take any $\lambda \in \mathbb{N}$, $n \in \mathsf{poly}(\lambda)$, $k \le n(\lambda)$, $j \in [k]$, and $\mathsf{msg} \in \mathcal{M}_\lambda$. Then, we have the following:
  - Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n(\lambda))$, and for each $i \in [k]$, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. Note that pp is simply the public parameters for $\mathsf{FBE}_{\mathsf{SS}}$. Each public key pk_i consists of two public keys $(\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$ for $\mathsf{FBE}_{\mathsf{SS}}$, and $\mathsf{sk}_i = (a, \mathsf{sk}_{i,a})$ is the corresponding secret key for one of the two public keys.
  - Let $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pp}, \mathsf{msg}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_k))$. Note that $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, r)$ consists of two ciphertexts for $\mathsf{FBE}_{\mathsf{SS}}$. The first ciphertext ct_0 is encrypted with respect to the public keys $\left(\mathsf{pk}_{1,t_1}, \ldots, \mathsf{pk}_{k,t_k}\right)$ while the second ct_1 is encrypted to the public keys $\left(\mathsf{pk}_{1,1-t_1}, \ldots, \mathsf{pk}_{k,1-t_k}\right)$, and $(t_1, \ldots, t_n) = O(r)$.
  - We argue that $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, (j, \mathsf{sk}_j), (\mathsf{pk}_1, \ldots, \mathsf{pk}_k)) = \mathsf{msg}$. Since $\mathsf{sk}_j = (a, \mathsf{sk}_{j,a})$ and $\mathsf{sk}_{j,a}$ is the secret key associated with public key $\mathsf{pk}_{j,a}$, we can appeal to correctness of $\mathsf{FBE}_{\mathsf{SS}}$ to conclude that $\mathsf{sk}_{j,a}$ can be used to decrypt ciphertext ct_0 if $a = t_j$ and ct_1 if $a = 1 - t_j$. □

Theorem B.3 (Adaptive Security). *Assuming* $\mathsf{FBE}_{\mathsf{SS}}$ *satisfies semi-static security and* O *is modeled as a random oracle, then Construction B.1 satisfies adaptive security.*

Proof. Suppose by way of contradiction that adaptive security does not hold. Namely, there exists a PPT adversary A, an efficiently computable function $n \in \mathsf{poly}(\lambda)$, a polynomial q, and an infinite set $\Lambda \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$, it holds that

$$\Pr\left[\mathsf{Expt}^{\mathsf{FBE}}_{A,n(\lambda)}(\lambda) = 1\right] > 1/2 + 1/q(\lambda).$$

Towards reaching a contradiction, we consider a series of hybrid experiments as follows:

- $H_{b,0}(\lambda)$: This experiment is identical to $\mathsf{Expt}^{\mathsf{FBE}}_{A,n(\lambda)}(\lambda)$ where the challenge bit is fixed to b.
- $H_{b,1}(\lambda)$: This experiment is identical to $H_{b,0}(\lambda)$ except the challenger encrypts the lexicographically first message $\mathsf{msg}^\star \in \mathcal{M}_\lambda$ instead of msg_b for $\mathsf{ct}^\star_0$ in the challenge ciphertext $\mathsf{ct}^\star = (\mathsf{ct}^\star_0, \mathsf{ct}^\star_1)$. Specifically, the challenger computes $\mathsf{ct}^\star_0 \leftarrow \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}^\star, (\mathsf{pk}_{1,t_1}, \ldots, \mathsf{pk}_{k,t_k}))$. The ciphertext $\mathsf{ct}^\star_1$ is computed as in the real scheme (i.e., as in $H_{b,0}$).
- $H_{b,2}(\lambda)$: In this experiment, the challenger encrypts the lexicographically first message $\mathsf{msg}^\star \in \mathcal{M}_\lambda$ instead of msg_b for $\mathsf{ct}^\star_1$ in the challenge phase. Specifically, when constructing the challenge ciphertext $\mathsf{ct}^\star = (\mathsf{ct}^\star_0, \mathsf{ct}^\star_1)$, the challenger computes $\mathsf{ct}^\star_1 \leftarrow \mathsf{FBE}_{\mathsf{SS}}.\mathsf{Enc}(\mathsf{pp}, \mathsf{msg}^\star, (\mathsf{pk}_{1,1-t_1}, \ldots, \mathsf{pk}_{k,1-t_k}))$. The ciphertext $\mathsf{ct}^\star_0$ is constructed as in $H_{b,1}$.

Since the challenge bit is sampled uniformly (i.e., $b \xleftarrow{R} \{0,1\}$), the success probability of A on any security parameter λ is equal to

$$\Pr[A \text{ wins}] = \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right).$$

For each $b \in \{0,1\}$ and $i \in \{1,2\}$, we define

$$\delta_{b,i}(\lambda) = \Pr[H_{b,i-1}(\lambda) = 1] - \Pr[H_{b,i}(\lambda) = 1].$$

It follows by our assumption that for all $\lambda \in \Lambda$,

$$\begin{aligned}\frac{1}{2} + \frac{1}{q(\lambda)} &\le \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right) \\ &\le \frac{1}{2} \cdot \big(\Pr[H_{0,2}(\lambda) = 1] + \delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) \\ &\qquad + \Pr[H_{1,2}(\lambda) = 1] + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda)\big).\end{aligned}$$

As $H_{b,2}(\lambda)$ is independent of b for each $b \in \{0,1\}$, it holds that $\Pr[H_{b,2}(\lambda) = 1] \le 1/2$. Thus, for all $\lambda \in \Lambda$,

$$\frac{1}{q(\lambda)} \le \frac{1}{2} \cdot \left(\delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda)\right).$$

As Λ is an infinitely large subset of $\mathbb{N}$, it must be the case that for some $b \in \{0,1\}$ and $i \in \{1,2\}$, $\delta_{b,i}(\lambda) \ge 1/(2q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. In the subsequent claims, we show that this contradicts semi-static security of $\mathsf{FBE}_{\mathsf{SS}}$. We prove the claims for $b = 0$ without loss of generality.

Claim B.4. *If $\delta_{0,1}(\lambda) \geq 1/(2q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then $\mathsf{FBE}_{\mathsf{SS}}$ does not satisfy semi-static security.*

Proof. Given an efficient adversary A that breaks adaptive security of FBE, we construct an efficient adversary B that breaks semi-static security of $\mathsf{FBE}_{\mathsf{SS}}$ as follows. In the security reduction, we model O as a random oracle (which algorithm B simulates for A):

- **Random oracle queries:** Whenever A makes an oracle query to O, algorithm B responds with a random string of $n(\lambda)$ bits and stores the response so it can respond consistently to any future queries.
- **Setup phase:** In the setup phase, algorithm B receives the public parameters pp from the $\mathsf{FBE}_{\mathsf{SS}}$ challenger, which it forwards to A. Algorithm B initializes a counter $\mathsf{ctr} \coloneqq 0$, a dictionary D, and a set $C = \varnothing$.
- **Pre-challenge query phase:** In the pre-challenge query phase, algorithm B simulates the queries for A as follows:
  - **Key-generation query:** Whenever A makes a key-generation query, algorithm B increments $\mathsf{ctr} \coloneqq \mathsf{ctr}+1$ and samples a bit $a \xleftarrow{\mathrm{R}} \{0,1\}$. Algorithm B then samples $(\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{FBE}_{\mathsf{SS}}.\mathsf{KeyGen}(\mathsf{pp})$. In addition, it makes a key-generation query to its challenger to obtain a public key pk_{1-a}. Algorithm B sends A the key $(\mathsf{pk}_0, \mathsf{pk}_1)$ and adds the mapping $\mathsf{ctr} \mapsto (a, (\mathsf{pk}_0, \mathsf{pk}_1), \mathsf{sk}_a)$ to D.
  - **Corruption query:** Whenever A makes a corruption query for counter value $c \in [\mathsf{ctr}]$, algorithm B retrieves $(a, (\mathsf{pk}_0, \mathsf{pk}_1), \mathsf{sk}_a) \coloneqq \mathsf{D}[c]$ and returns (a, sk_a). Algorithm B adds c to C.
- **Challenge phase:** In the challenge phase, A sends two messages $\mathsf{msg}'_0, \mathsf{msg}'_1 \in \mathcal{M}_\lambda$ and an ordered list $S^\star = (i_1, \ldots, i_{k^\star}) \subseteq [\mathsf{ctr}]$ to B. Algorithm B proceeds as follows:
  - For each index $i_j \in S^\star$, algorithm B retrieves $(a, (\mathsf{pk}_0, \mathsf{pk}_1), \mathsf{sk}_a) \coloneqq \mathsf{D}[i_j]$, and sets $t_j = 1 - a$. For indices $j \in \{k^\star + 1, \ldots, n\}$, algorithm B samples $t_j \xleftarrow{\mathrm{R}} \{0,1\}$.
  - Algorithm B samples a random value $r^\star \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ and programs the random oracle so that on input r, the value of $O(r^\star) = (t_1, \ldots, t_{k^\star}) \in \{0,1\}^n$. If adversary A has previously queried O on $r^\star$, then algorithm B aborts with output $\perp$.
  - Algorithm B sends the same set $S^\star$ and messages $\mathsf{msg}_0 = \mathsf{msg}'_0$ and $\mathsf{msg}_1 = \mathsf{msg}^\star$, which is the lexicographically first message in $\mathcal{M}_\lambda$, to the $\mathsf{FBE}_{\mathsf{SS}}$ challenger. Let $\mathsf{ct}^\star_0$ be the challenge ciphertext constructed by the challenger. By construction, the challenger samples a bit $\hat{b} \xleftarrow{\mathrm{R}} \{0,1\}$ and replies with an encryption $\mathsf{msg}_{\hat{b}}$ under the honestly-generated public keys for $S^\star$.
  - Algorithm B computes a ciphertext for the message msg_0 using the public keys for $S^\star$ that it generated itself. Algorithm B sends $\mathsf{ct}^\star = (\mathsf{ct}^\star_0, \mathsf{ct}^\star_1, r^\star)$ to A.
- **Output phase:** At the end of the game, algorithm A outputs a bit $b' \in \{0,1\}$, which B also outputs.

Algorithm B is efficient as long as A is efficient, so it suffices to analyze its success probability. First, we define $\mathsf{QE}(\lambda)$ to be the event where A queries $O(r^\star)$ before the challenge phase (in which case algorithm B aborts). As A runs in polynomial-time, there exists some polynomial $Q(\lambda)$ bounding the number of queries that A makes to the random oracle before the challenge phase. Since algorithm B samples $r^\star$ uniformly at random from $\{0,1\}^\lambda$, it follows that

$$\Pr[\mathsf{QE}(\lambda)] \leq Q(\lambda)/2^\lambda.$$

Conditioned on $\mathsf{QE}(\lambda)$ not occurring, we observe that $t_1, \ldots, t_{k^\star}$ are uniformly distributed given the view of algorithm A in the security game (this is because algorithm B samples $a \xleftarrow{\mathrm{R}} \{0,1\}$ when responding to each key-generation query). We proceed to analyze the probability that B outputs the correct guess b' for the challenge bit $\hat{b}$, conditioned on $\mathsf{QE}(\lambda)$ not holding. If the challenge bit $\hat{b} = 0$, then the view of A is distributed according to $H_{0,0}(\lambda)$. This means that B outputs the correct bit whenever A wins since $b' = b = 0$. If the challenge bit $\hat{b} = 1$, then the view of A is

distributed according to $H_{0,1}(\lambda)$. So, B outputs the correct bit whenever A loses since $b' \neq b = 0$. It follows that the success probability of B is at least

$$\begin{aligned}\Pr[B \text{ wins}] &\geq \Pr[\neg\mathsf{QE}(\lambda)] \cdot \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1 \mid \neg\mathsf{QE}(\lambda)] + \Pr[H_{0,1}(\lambda) = 0 \mid \neg\mathsf{QE}(\lambda)]\right) \\ &\geq \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] - \Pr[H_{0,1}(\lambda) = 1]\right) - \Pr[\mathsf{QE}(\lambda)].\end{aligned}$$

As $\delta_{0,1}(\lambda) > 1/(2q(\lambda))$ by assumption and $\Pr[\mathsf{QE}(\lambda)] \leq Q(\lambda)/2^\lambda$ is negligible, this violates the semi-static security of $\mathsf{FBE}_{\mathsf{SS}}$. □

Claim B.5. *If $\delta_{0,2}(\lambda) \geq 1/(2q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then $\mathsf{FBE}_{\mathsf{SS}}$ does not satisfy semi-static security.*

Proof. The proof of this claim is essentially identical to the proof of Claim B.4, except the reduction algorithm B programs the random oracle so that $t_j = a$ instead of $1 - a$. It then follows that the honestly-generated keys correspond to $\mathsf{ct}_1^\star$ instead of $\mathsf{ct}_0^\star$. In this case, the challenge ciphertext is used to simulate $\mathsf{ct}_1^\star$ instead of $\mathsf{ct}_0^\star$ in the reduction. Algorithm B constructs $\mathsf{ct}_0^\star$ itself (according to the specification of $H_{0,1}$). The rest of the proof then follows identically to the previous proof. □

The claim now follows via Claims B.4 and B.5. □

C Slotted Registered ABE with Policy-Selective Security

In this section, we show how to transform a slotted registered ABE scheme that satisfies policy-selective security *without* corruptions into one that satisfies policy-selective security *with* corruptions in the random oracle model. As was the case with flexible broadcast encryption (Appendix B), our transformation is a simple adaptation of the two-key approach from [GW09]. We note that the random oracle in our transformation is only used for key aggregation. We could alternatively consider a notion of slotted registered ABE where we allow for a randomized key aggregation procedure. This would yield a registered ABE scheme that satisfies policy-selective security in the plain model, albeit with a randomized (though public-coin) aggregation procedure. We discuss this in more detail in Remark C.6.

Construction C.1 (Slotted Registered ABE with Policy-Selective Security)**.** Let $\lambda \in \mathbb{N}$ be a security parameter and L be the number of slots. Let $\ell = \ell(\lambda)$ be an attribute length and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a policy space on ℓ-bit inputs (i.e., $\mathcal{P}_\lambda$ is a set of functions $P\colon \{0,1\}^{\ell(\lambda)} \to \{0,1\}$). Let $m = m(\lambda)$ and $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space with message length m. Our construction relies on the following primitives:

- First, let $\mathsf{srABE}_{\mathsf{PSWC}} = (\mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Setup}, \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{KeyGen}, \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Aggregate}, \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Enc}, \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Dec})$ be a slotted registered ABE scheme for attributes of length ℓ, policy space $\mathcal{P}$, and message space $\mathcal{M}$. We require that $\mathsf{srABE}_{\mathsf{PSWC}}$ satisfies policy-selective security without corruptions.
- Let $O\colon \{0,1\}^* \to \{0,1\}^L$ be a hash function (that we will model as a random oracle in the security analysis). For simplicity of notation, we assume O can take an arbitrary bit-string (of unbounded length) as input; however, it suffices to have a hash function on bounded-length inputs of the form $(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$ where pp is the public parameters for $\mathsf{srABE}_{\mathsf{PSWC}}$, and each pk_i consists of a two public keys for $\mathsf{srABE}_{\mathsf{PSWC}}$ along with a λ-bit string, and each $x_i \in \{0,1\}^\ell$ is an attribute.

We construct a slotted registered ABE scheme $\mathsf{srABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Enc}, \mathsf{Dec})$ as follows:

- $\mathsf{Setup}(1^\lambda, L)$: On input the security parameter λ and the number of slots L, the setup algorithm outputs $\mathsf{pp} \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Setup}(1^\lambda, L)$.
- $\mathsf{KeyGen}(\mathsf{pp}, i)$: On input the public parameters pp and an index $i \in [L]$, the key-generation algorithm samples two public keys $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{KeyGen}(\mathsf{pp}, i)$ and $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{KeyGen}(\mathsf{pp}, i)$. It also samples a random bit $a \xleftarrow{R} \{0,1\}$ and $r \xleftarrow{R} \{0,1\}^\lambda$ and outputs $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1, r)$ and $\mathsf{sk} = (a, \mathsf{sk}_a)$.

- $\mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$: On input the public parameters pp, a tuple of public keys $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$ and associated attributes $x_i \in \{0,1\}^\ell$, the aggregation algorithm first computes $(t_1, \ldots, t_L) = O(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$. Then, it prepares two sets of public keys and secret keys as follows:

$$(\mathsf{mpk}_0, \mathsf{hsk}_{1,0}, \ldots, \mathsf{hsk}_{L,0}) = \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Aggregate}\big(\mathsf{pp}, (\mathsf{pk}_{1,t_1}, x_1), \ldots, \big(\mathsf{pk}_{L,t_L}, x_L\big)\big)$$
$$(\mathsf{mpk}_1, \mathsf{hsk}_{1,1}, \ldots, \mathsf{hsk}_{L,1}) = \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Aggregate}\big(\mathsf{pp}, (\mathsf{pk}_{1,1-t_1}, x_1), \ldots, \big(\mathsf{pk}_{L,1-t_L}, x_L\big)\big).$$

  It outputs the master public key $\mathsf{mpk} = (\mathsf{mpk}_0, \mathsf{mpk}_1)$ and the helper decryption keys $\mathsf{hsk}_i = (t_i, \mathsf{hsk}_{i,0}, \mathsf{hsk}_{i,1})$ for each $i \in [L]$.

- $\mathsf{Enc}(\mathsf{mpk}, P, \mathsf{msg})$: On input the master public key $\mathsf{mpk} = (\mathsf{mpk}_0, \mathsf{mpk}_1)$, the policy $P \in \mathcal{P}_\lambda$, and the message $\mathsf{msg} \in \mathcal{M}_\lambda$, the encryption algorithm constructs two ciphertexts $\mathsf{ct}_0 \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Enc}(\mathsf{mpk}_0, P, \mathsf{msg})$ and $\mathsf{ct}_1 \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Enc}(\mathsf{mpk}_1, P, \mathsf{msg})$. It outputs $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$.

- $\mathsf{Dec}(\mathsf{mpk}, \mathsf{hsk}, \mathsf{sk}, \mathsf{ct})$: On input the master public key $\mathsf{mpk} = (\mathsf{msk}_0, \mathsf{msk}_1)$, the helper decryption key $\mathsf{hsk} = (t, \mathsf{hsk}_0, \mathsf{hsk}_1)$, the secret key $\mathsf{sk} = (a, \mathsf{sk}_a)$, and the ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$, the decryption algorithm proceeds as follows:
  - If $a = t$, output $\mathsf{msg} = \mathsf{Dec}(\mathsf{mpk}_0, \mathsf{hsk}_0, \mathsf{sk}_a, \mathsf{ct}_0)$.
  - If $a = 1 - t$, output $\mathsf{msg} = \mathsf{Dec}(\mathsf{mpk}_1, \mathsf{hsk}_1, \mathsf{sk}_a, \mathsf{ct}_1)$.

Correctness and security analysis. We now show that srABE of Construction C.1 is a correct slotted registered ABE scheme that satisfies policy-selective security (with corruptions).

Theorem C.2 (Correctness and Efficiency)**.** *Assuming* $\mathsf{srABE}_{\mathsf{PSWC}}$ *is correct and compact, then Construction C.1 is correct and compact.*

Proof. We consider each property separately:

- **Compactness:** The master public key mpk consists of two master public keys $(\mathsf{mpk}_0, \mathsf{mpk}_1)$ for $\mathsf{srABE}_{\mathsf{PSWC}}$. By compactness of $\mathsf{srABE}_{\mathsf{PSWC}}$, $|\mathsf{mpk}_0|, |\mathsf{mpk}_1| \in \mathsf{poly}(\lambda, \ell, \log L)$ so the same holds for $|\mathsf{mpk}|$. Similarly, each helper decryption key hsk_i consists of two helper secret keys $(\mathsf{hsk}_{i,0}, \mathsf{hsk}_{i,1})$ for $\mathsf{srABE}_{\mathsf{PSWC}}$ along with a single additional bit. Again by compactness of $\mathsf{srABE}_{\mathsf{PSWC}}$, it follows that $|\mathsf{hsk}_i| \in \mathsf{poly}(\lambda, \ell, \log L)$.

- **Correctness:** Take any $\lambda \in \mathbb{N}$, $L \in \mathbb{N}$, $i \in [L]$, message $\mathsf{msg} \in \mathcal{M}_\lambda$, attributes $x_1, \ldots, x_L \in \{0,1\}^\ell$, policy $P \in \mathcal{P}_\lambda$ where $P(x_i) = 1$. Then, we have the following:
  - Let $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, L)$, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$, and $\{\mathsf{pk}_j\}_{j \neq i}$ be any collection of public keys (which may be arbitrarily correlated with pk_i). By construction, pp is simply the public parameters for the underlying scheme $\mathsf{srABE}_{\mathsf{PSWC}}$ and the public key $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_1, r)$ consists of a pair of public keys $(\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$ for $\mathsf{srABE}_{\mathsf{PSWC}}$ along with a random string $r_i \in \{0,1\}^\lambda$. The associated secret key $\mathsf{sk}_i = (a, \mathsf{sk}_{i,a})$ is the corresponding secret key for $\mathsf{pk}_{i,a}$.
  - Let $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) = \mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$ and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, P, \mathsf{msg})$. By construction, $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1)$ consists of two ciphertexts for $\mathsf{srABE}_{\mathsf{PSWC}}$. The first ciphertext ct_0 is with respect to mpk_0 which was obtained by aggregating the public keys $(\mathsf{pk}_{1,t_1}, \ldots, \mathsf{pk}_{L,t_L})$ while the second ciphertext ct_1 is with respect to mpk_1 which was obtained by aggregating the public keys $(\mathsf{pk}_{1,1-t_1}, \ldots, \mathsf{pk}_{L,1-t_L})$. Here, the bits $t_1, \ldots, t_L$ are derived by evaluating $O(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$. In particular, we note that $\mathsf{pk}_{i,a}$ is aggregated in mpk_0 if $a = t_i$ and mpk_1 if $a = 1 - t_i$.
  - We argue that $\mathsf{Dec}(\mathsf{pp}, \mathsf{hsk}_i, \mathsf{sk}_i, \mathsf{ct}) = \mathsf{msg}$. Note that $\mathsf{sk}_i = (a, \mathsf{sk}_{i,a})$. By construction, the helper decryption key hsk_i contains the bit t_i that specifies which of $\mathsf{pk}_{i,0}$ and $\mathsf{pk}_{i,1}$ was aggregated in mpk_0 and which was aggregated in mpk_1. Since $P(x_i) = 1$, by correctness of $\mathsf{srABE}_{\mathsf{PSWC}}$, either $a = t_i$ in which case $\mathsf{sk}_{i,a}$ correctly decrypts ct_0, or $a = 1 - t_i$, in which case $\mathsf{sk}_{i,a}$ correctly decrypts ct_1. □

Theorem C.3 (Policy-Selective Security). *Assuming* $\mathsf{srABE}_{\mathsf{PSWC}}$ *satisfies policy-selective security without corruptions and* O *is modeled as a random oracle, then Construction C.1 satisfies policy-selective security.*

Proof. Suppose there exists a PPT adversary A, an efficiently computable function $L \in \mathsf{poly}(\lambda)$, a polynomial q, and an infinite set $\Lambda \subseteq \mathbb{N}$ such that for all $\lambda \in \Lambda$, it holds that

$$\Pr\left[\mathsf{Expt}^{\mathsf{srABE,PS}}_{A,L(\lambda)}(\lambda) = 1\right] > 1/2 + 1/q(\lambda).$$

Towards reaching a contradiction, we consider a series of hybrid experiments as follows:

- $H_{b,0}(\lambda)$: This experiment is identical to $\mathsf{Expt}^{\mathsf{srABE,PS}}_{A,L(\lambda)}(\lambda)$ where the challenge bit is fixed to b.
- $H_{b,1}(\lambda)$: This experiment is identical to $H_{b,0}(\lambda)$ except the challenger encrypts the lexicographically first message $\mathsf{msg}^\star \in \mathcal{M}_\lambda$ instead of msg_b for $\mathsf{ct}^\star_0$ when constructing the challenge ciphertext $\mathsf{ct}^\star = (\mathsf{ct}^\star_0, \mathsf{ct}^\star_1)$. Specifically, the challenger computes $\mathsf{ct}^\star_0 \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Enc}(\mathsf{mpk}_0, P^\star, \mathsf{msg}^\star)$. The ciphertext $\mathsf{ct}^\star_1$ is computed as in the real scheme (i.e., as in $H_{b,0}$).
- $H_{b,2}(\lambda)$: In this experiment, the challenger encrypts the lexicographically first message $\mathsf{msg}^\star \in \mathcal{M}_\lambda$ instead of msg_b for $\mathsf{ct}^\star_1$ in the challenge phase. Specifically, when constructing the challenge ciphertext $\mathsf{ct}^\star = (\mathsf{ct}^\star_0, \mathsf{ct}^\star_1)$, the challenger computes $\mathsf{ct}^\star_1 \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Enc}(\mathsf{mpk}_1, P^\star, \mathsf{msg}^\star)$. The ciphertext $\mathsf{ct}^\star_1$ is constructed as in $H_{b,1}$.

Since the challenge bit $b \xleftarrow{R} \{0,1\}$ is sampled uniformly, the success probability of A on any security parameter λ is equal to

$$\Pr[A \text{ wins}] = \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right).$$

For each $b \in \{0,1\}$ and $i \in \{1,2\}$, we define

$$\delta_{b,i}(\lambda) = \Pr[H_{b,i-1}(\lambda) = 1] - \Pr[H_{b,i}(\lambda) = 1].$$

It follows by our assumption that for all $\lambda \in \Lambda$,

$$\begin{aligned}\frac{1}{2} + \frac{1}{q(\lambda)} &\leq \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{1,0}(\lambda) = 1]\right) \\ &\leq \frac{1}{2} \cdot \big(\Pr[H_{0,2}(\lambda) = 1] + \delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) \\ &\qquad + \Pr[H_{1,2}(\lambda) = 1] + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda)\big).\end{aligned}$$

As $H_{b,2}(\lambda)$ is independent of b for each $b \in \{0,1\}$, it holds that $\Pr[H_{b,2}(\lambda) = 1] \leq 1/2$. Thus, for all $\lambda \in \Lambda$,

$$\frac{1}{q(\lambda)} \leq \frac{1}{2} \cdot \left(\delta_{0,1}(\lambda) + \delta_{0,2}(\lambda) + \delta_{1,1}(\lambda) + \delta_{1,2}(\lambda)\right).$$

As Λ is an infinitely large subset of $\mathbb{N}$, it must be the case that for some $b \in \{0,1\}$ and $i \in \{1,2\}$, $\delta_{b,i}(\lambda) \geq 1/(2q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$. In the subsequent claims, we show that this contradicts the policy-selective security of $\mathsf{srABE}_{\mathsf{PSWC}}$ without corruption queries. We prove the claims for $b = 0$ without loss of generality.

Claim C.4. *If* $\delta_{0,1}(\lambda) \geq 1/(2q(\lambda))$ *for infinitely many* $\lambda \in \mathbb{N}$, *then* $\mathsf{srABE}_{\mathsf{PSWC}}$ *does not satisfy policy-selective security without corruption queries.*

Proof. Let $Q = Q(\lambda)$ be a polynomial that upper bounds the number of pre-challenge random oracle queries the adversary A makes. Additionally, we assume without loss of generality that the following conditions holds:

- Algorithm A always queries O on the input to the Aggregate algorithm prior to the challenge phase.
- Algorithm A does not query the random oracle on the same input multiple times.

If A does not satisfy the first requirement, we can use A to construct a new adversary that makes one additional random oracle query and which succeeds with the same advantage. Similarly, if A is an algorithm that queries O on the same input multiple times, we can transform A into an adversary that makes unique queries to O (and remembers the responses to each oracle query so as to simulate future queries on a repeated input). This change also does not affect the advantage of the adversary. We now use A to construct an adversary B for the policy-selective security game for $\mathsf{srABE}_{\mathsf{PSWC}}$ *without* corruption queries:

- **Setup phase:** Algorithm B starts running algorithm A who starts by committing to a challenge policy $P^\star$. Algorithm B forwards the policy $P^\star$ to the challenger and receives the public parameters pp from the challenger, which it forwards to A. Algorithm B then initializes a counter $\mathsf{ctr} := 0$, a dictionary D, and a set $C = \emptyset$. Algorithm B also samples a random index $i^\star \xleftarrow{R} [Q]$.
- **Random oracle queries:** When A makes a random oracle query on a tuple $\big(\mathsf{pp}', (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L)\big)$, algorithm B proceeds as follows. If this is not the $(i^\star)^{\text{th}}$ pre-challenge random oracle made by A, then algorithm B responds with a uniform random string of length L. If this is the $(i^\star)^{\text{th}}$ pre-challenge random oracle made by A, then algorithm B proceeds as follows:
  - If $\mathsf{pp}' \neq \mathsf{pp}$, algorithm B outputs a uniform random bit $b' \xleftarrow{R} \{0,1\}$.
  - For each $i \in [L]$, algorithm B computes $t_i \in \{0,1\}$ as follows:
    - If $P^\star(x_i) = 0$, algorithm B samples $t_i \xleftarrow{R} \{0,1\}$.
    - If $P^\star(x_i) = 1$, B checks if there is a tuple $(i, a_i, \mathsf{pk}_i, \mathsf{sk}_i)$ in D associated with some counter value c_i. If no such tuple exists, B samples $t_i \xleftarrow{R} \{0,1\}$. Otherwise, B sets $t_i = 1 - a_i$.
  - Algorithm B replies with $(t_1, \ldots, t_L) \in \{0,1\}^L$.
- **Pre-challenge query phase:** In the pre-challenge query phase, algorithm B responds to queries as follows:
  - **Key-generation query:** Whenever A makes a key generation query for slot $i \in [L]$, algorithm B increments $\mathsf{ctr} := \mathsf{ctr} + 1$ and samples a bit $a \xleftarrow{R} \{0,1\}$ and string $r \xleftarrow{R} \{0,1\}^\lambda$. Algorithm B samples a key $(\mathsf{pk}_a, \mathsf{sk}_a) \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{KeyGen}(\mathsf{pp}, i)$. It also makes a key-generation query to its own challenger to obtain a public key pk_{1-a}. Finally, algorithm B samples $r \xleftarrow{R} \{0,1\}^\lambda$ and replies to A with the public key $\mathsf{pk} = (\mathsf{pk}_0, \mathsf{pk}_1, r)$. It also adds the mapping $\mathsf{ctr} \mapsto (i, a, (\mathsf{pk}_0, \mathsf{pk}_1, r), \mathsf{sk}_a)$ to D.
  - **Corruption query:** Whenever A makes a corruption query on an index $c \in [\mathsf{ctr}]$, algorithm B retrieves $(i, a, (\mathsf{pk}_0, \mathsf{pk}_1, r), \mathsf{sk}_a) := \mathsf{D}[c]$, and returns (a, sk_a).
- **Challenge phase:** In the challenge phase, algorithm A chooses two messages $\mathsf{msg}_0', \mathsf{msg}_1' \in \mathcal{M}_\lambda$ and for each slot $i \in [L]$, it specifies a tuple $(c_i', x_i, \mathsf{pk}_i^\star)$. Algorithm B now constructs the public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_L)$ according to the specification of the adaptive security game:
  - If $c_i' \in \{1, \ldots, \mathsf{ctr}\}$, algorithm B sets $(i', a_i, (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1}, r_i), \mathsf{sk}_i) := \mathsf{D}[c_i']$. If $i \neq i'$, then algorithm B aborts and outputs a uniform random bit $b' \xleftarrow{R} \{0,1\}$. Otherwise, algorithm B sets $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1}, r_i)$. If the adversary previously issued a corruption query on counter c_i', then algorithm B also adds the slot index i to the corrupted set C.
  - If $c_i' = \perp$, then algorithm B sets $\mathsf{pk}_i := \mathsf{pk}_i^\star$ and adds the slot index i to the corrupted set C.

  Algorithm B now checks that for all $i \in C$, it holds that $P^\star(x_i) = 0$. If not, algorithm B aborts and outputs a uniform random bit $b' \xleftarrow{R} \{0,1\}$. Otherwise, if $P^\star(x_i) = 0$ for all $i \in C$, algorithm B constructs the challenge ciphertext as follows:
  - First, it checks whether algorithm A has made at least $i^\star$ queries to the random oracle. If not, then algorithm B halts with output $\perp$.
  - If algorithm A has made at least $i^\star$ queries to the random oracle and its $(i^\star)^{\text{th}}$ query was *not* on input $(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$, then algorithm B aborts and outputs a uniform random bit $b' \xleftarrow{R} \{0,1\}$. Otherwise let $(t_1, \ldots, t_L) := O(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$ be the value algorithm B used to respond to the $(i^\star)^{\text{th}}$ random oracle query.

- For each $i \in [L]$, algorithm B sets $c_i := c'_i$.
- Algorithm B outputs $\mathsf{msg}_0 = \mathsf{msg}'_0$ and $\mathsf{msg}_1 = \mathsf{msg}^\star$ as its challenge message and $(c_i, x_i, \mathsf{pk}_{i,t_i})$ for each $i \in [L]$ as its public keys. The challenger replies with a challenge ciphertext $\mathsf{ct}^\star_0$. Specifically, the challenger samples a bit $\hat{b} \xleftarrow{R} \{0,1\}$ and sends back an encryption of $\mathsf{msg}_{\hat{b}}$.
- Finally, algorithm B also computes $\mathsf{mpk}_1 = \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Aggregate}(\mathsf{pp}, (\mathsf{pk}_{1,1-t_1}, x_1), \ldots, (\mathsf{pk}_{L,1-t_L}, x_L))$ and $\mathsf{ct}^\star_1 \leftarrow \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Enc}(\mathsf{mpk}_1, P^\star, \mathsf{msg}_0)$.
- Algorithm B sends $\mathsf{ct}^\star = (\mathsf{ct}^\star_0, \mathsf{ct}^\star_1)$ to A.

- **Output phase:** Algorithm A outputs a bit $b' \in \{0,1\}$, which B also outputs.

If A is efficient, then algorithm B is also efficient by construction. It suffices to analyze the success probability of algorithm B. First, we note that if algorithm B aborts before the output phase, then it always outputs a uniform random bit $b' \in \{0,1\}$. In this case, the advantage of algorithm B is exactly $1/2$. Suppose algorithm B does not abort early. Then it must be the case that the following properties hold:

- Algorithm A makes at least $i^\star$ queries to the random oracle prior to the challenge phase, and moreover, the $(i^\star)^{\text{th}}$ query algorithm A makes to the random oracle is on the tuple $(\mathsf{pp}, (\mathsf{pk}_1, x_1), \ldots, (\mathsf{pk}_L, x_L))$ that algorithm B constructs in the challenge phase.
- Every tuple $(c'_i, x_i, \mathsf{pk}^\star_i)$ that algorithm A provides in the challenge phase has the property that if $c'_i \in \{1, \ldots, \mathsf{ctr}\}$, then $i = i'$ where $(i', a_i, (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1}, r_i)) := \mathsf{D}[c'_i]$. Similarly, for all $i \in C$, it holds that $P^\star(x_i) = 0$.

By construction, this means that algorithm B is admissible. Next, let $\mathsf{QE}(\lambda)$ be the event where the challenge query contains an honestly-generated public key pk_i, but algorithm A makes a challenge query before pk_i is actually generated and inserted into D. Since each public key pk_i contains a random string $r \leftarrow \{0,1\}^\lambda$ and A makes at most $Q(\lambda) = \mathsf{poly}(\lambda)$ queries to O, it follows that

$$\Pr[\mathsf{QE}(\lambda)] \leq Q(\lambda)/2^\lambda = \mathsf{negl}(\lambda).$$

We now show that conditioned on event QE *not* happening, algorithm B correctly simulates either hybrid $H_{0,0}$ or hybrid $H_{0,1}$. For this to be the case, $\mathsf{ct}^\star_0$ should be sampled according to $\mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Enc}(\mathsf{mpk}_0, P^\star, \mathsf{msg}_{\hat{b}})$ where $\mathsf{mpk}_0 = \mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Aggregate}\big(\mathsf{pp}, (\mathsf{pk}_{1,t_1}, x_1), \ldots, (\mathsf{pk}_{L,t_L}, x_L)\big)$ and each t_i is uniformly random given A's view. For each slot $i \in [L]$, we show that the challenger uses the key pk_{i,t_i} as input to $\mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Aggregate}$ and the corresponding bit t_i output by O is uniformly random conditioned on A's view. We consider two cases:

- Suppose $c'_i = \perp$. In this case, the challenger uses the public key pk_{i,t_i} provided by B as the i^{th} public key when computing $\mathsf{srABE}_{\mathsf{PSWC}}.\mathsf{Aggregate}$. This means algorithm A chose the public key itself. In this case, if A is admissible, then it must be the case that $P^\star(x_i) = 0$. By construction of algorithm B, it sampled $t_i \xleftarrow{R} \{0,1\}$ in this case.
- Suppose $c'_i \neq \perp$. Since $c'_i \neq \perp$, then $\mathsf{D}[c'_i]$ must contain a tuple $(i, a_i, \mathsf{pk}_i, \mathsf{sk}_i)$ where $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1}, r_i)$. In this case, algorithm B sets $\mathsf{pk}_{i,t_i} = \mathsf{pk}_{i,1-a_i}$. By construction, $\mathsf{pk}_{i,1-a_i}$ is the key chosen by the challenger in response to the $(c'_i)^{\text{th}}$ key-generation query. Correspondingly, the challenger in this case uses pk_{i,t_i} to construct the aggregated key. We again consider two cases:
  - If $P^\star(x_i) = 0$, then t_i is uniform by construction.
  - If $P^\star(x_i) = 1$, then conditioned on $\mathsf{QE}(\lambda)$ not occurring, it must be the case case that $t_i = 1 - a_i$. Since $P^\star(x_i) = 1$ and assuming A is admissible, then algorithm A does not make a corruption query on index $c_i = c'_i$. In this case, the value of a_i is information-theoretically hidden from the view of A. Since algorithm B samples $a_i \xleftarrow{R} \{0,1\}$ and sets $t_i = 1 - a_i$, the bit t_i is correctly distributed in this case.

Since t_i is appropriately distributed for each slot $i \in [L]$, it also follows that $\mathsf{ct}^\star_0$ is distributed according to $H_{0,0}$ when $\hat{b} = 0$ and according to $H_{0,1}$ when $\hat{b} = 1$. Moreover, algorithm B constructs $\mathsf{ct}^\star_1$ exactly according to the specification

of $H_{0,0}$ and $H_{0,1}$. Therefore, it follows that if $\hat{b} = 0$, algorithm B wins whenever $H_{0,0}(\lambda) = 1$ (since $b = 0$), and if $\hat{b} = 0$, algorithm B wins whenever $H_{0,1}(\lambda) = 0$. To complete the proof, we now compute the overall success probability of B. Let $\mathsf{E}(\lambda)$ be the event that A queries the challenge query on query $i^\star$. By assumption, algorithm A is required to query the random oracle on its challenge query at some point prior to the challenge phase, so if A makes exactly $Q = Q(\lambda)$ queries, then

$$\Pr[\mathsf{E}(\lambda)] = 1/|Q|.$$

Let $\overline{\mathsf{QE}}(\lambda)$ denote the complement of $\mathsf{QE}(\lambda)$. For notational convenience, in the following, we omit the explicit dependence on the security parameter when it is clear from context. From the above analysis, we have that

$$\begin{aligned}\Pr[B \text{ wins} \mid \mathsf{E}, \overline{\mathsf{QE}}] &= \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] + \Pr[H_{0,1}(\lambda) = 0]\right) \\ &= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[H_{0,0}(\lambda) = 1] - \Pr[H_{0,1}(\lambda) = 1]\right) \\ &= \frac{1}{2} + \frac{1}{2}\delta_{0,1}(\lambda).\end{aligned}$$

Moreover, conditioned on $\overline{\mathsf{E}}$, we have by construction that $\Pr[B \text{ wins} \mid \overline{\mathsf{E}}] = 1/2$. Putting everything together, we can now write

$$\begin{aligned}\Pr[B \text{ wins}] &\geq \Pr[B \text{ wins} \mid \mathsf{E}, \overline{\mathsf{QE}}] \cdot \Pr[\mathsf{E} \mid \overline{\mathsf{QE}}] \cdot \Pr[\overline{\mathsf{QE}}] + \Pr[B \text{ wins} \mid \overline{\mathsf{E}}] \cdot \Pr[\overline{\mathsf{E}}] \\ &\geq \left(\frac{1}{2} + \frac{1}{2}\delta_{0,1}(\lambda))\right) \cdot \frac{1}{Q} \cdot \left(1 - \frac{Q}{2^\lambda}\right) + \frac{1}{2} \cdot \left(1 - \frac{1}{Q}\right) \\ &\geq \frac{1}{2} + \frac{1}{2Q}\delta_{0,1}(\lambda) - \mathsf{negl}(\lambda),\end{aligned}$$

whenever $Q = Q(\lambda)$ is polynomially-bounded. Thus, algorithm B breaks policy-selective security of $\mathsf{srABE}_{\mathsf{PSWC}}$ without corruption queries. □

Claim C.5. *If $\delta_{0,2}(\lambda) \geq 1/(2q(\lambda))$ for infinitely many $\lambda \in \mathbb{N}$, then $\mathsf{srABE}_{\mathsf{PSWC}}$ does not satisfy policy-selective security without corruptions.*

Proof. The proof of this claim is essentially identical to the proof of Claim C.4 with the following minor modifications:

- The reduction algorithm B now programs $t_i := a_i$, instead of $t_i = 1 - a_i$ when programming the random oracle for the challenge query.
- When answering key-generation queries, after sampling the bit $a \xleftarrow{R} \{0,1\}$, algorithm B now samples pk_{1-a} itself (using the honest key-generation algorithm) and makes a key-generation query to its own challenger to obtain pk_a.
- When generating the challenge ciphertext, algorithm B constructs $\mathsf{ct}_0^\star$ itself (according to the specification of $H_{0,1}$ and $H_{0,2}$). It uses the challenger to construct $\mathsf{ct}_1^\star$ using the public keys $\mathsf{pk}_{i,1-t_i}$ for each slot $i \in [L]$ instead of pk_{i,t_i}.

The rest of the proof follows identically to the proof of Claim C.4. □

Combining Claims C.4 and C.5 completes the proof of the theorem. □

Remark C.6 (Registered ABE without Random Oracles)**.** We note that the random oracle in Construction C.1 is only invoked in the Aggregate algorithm. We could alternatively consider a notion of (slotted) registered ABE where we allow key aggregation to be randomized. This yields a scheme satisfying policy-selective security in the *plain* model. However, the security proof in this case would only hold for a *semi-honest* key curator that samples a uniform random string for key aggregation. Security does not necessarily hold if the key curator behaves maliciously, and as such, we find this notion significantly less compelling from a definitional perspective. A key motivation for considering notions like registration-based encryption and registered ABE is that we do *not* need to trust any central authority. As such, it is more natural to model the key curator as a public, deterministic entity whose behavior can be easily audited.