Delegatable ABE with Full Security from Witness Encryption

Rishab Goyal
UW-Madison[*]

Saikumar Yadugiri
UW-Madison[†]

Abstract

Delegatable Attribute-Based Encryption (DABE) is a well-known generalization of ABE, proposed to mirror organizational hierarchies. In this work, we design a *fully-secure* DABE scheme from witness encryption and other simple assumptions. Our construction does not rely on Random Oracles, and we provide a black-box reduction to polynomial hardness of underlying assumptions. To the best of our knowledge, this is the first DABE construction (beyond hierarchical identity-based encryption) that achieves full security without relying on complexity leveraging. Our DABE supports an unbounded number of key delegations, and the secret key size grows just linearly with each key delegation operation.

1 Introduction

Since its inception, Attribute-Based Encryption (ABE) [SW05, GPSW06] has significantly revolutionized data encryption. It supports fine-grained access over encrypted data. Over the past two decades, ABE has received tremendous attention from the research community, leading to innumerable designs with varying efficiency, functionalities, security guarantees, and diverse security assumptions [SW05, GPSW06, BW07, BSW07, KSW08, Wat09, LW10, LOS+10, GVW13, GGSW13, BGG+14, GVW15]. ABE is extremely useful for many practical applications [PRV12, GKP+13, GM15, SRGS12, CDEN12, APG+11, TBEM08, BBS+09], and many prominent ABE schemes are also practically efficient [GPSW06, BSW07, Wat11, CGW15, AC17].

In ABE, a central authority sets up the system where any user can encrypt data under their choice of attribute, and a user can only decrypt the resulting ciphertext if and only if they can obtain a secret key for an accepting predicate. In particular, a ciphertext ct encrypting a payload message m, under attribute x, can be decrypted using a secret key sk_ϕ, associated with predicate ϕ, if $\phi(x) = 1$. For example, consider an academic institution that has deployed an ABE system for internal communication. Using ABE, a student can encrypt their homework solutions under the attribute *'course:CS101'*. Clearly, this can be accessed by any of the course staff (i.e., instructors, TAs, etc), but not by other students.

Delegatable Attribute-Based Encryption (DABE) is a well-known generalization of ABE [GVW13, BGG+14], that was proposed to mirror organizational hierarchies in ABE systems. In a few words, DABE enables full key delegation capabilities (i.e., sk_f can be delegated to another $\mathsf{sk}_{f \wedge g}$). Connecting to the earlier scenario, an instructor for *'course:CS101'* can delegate their secret keys to

[*]Email: `rishab@cs.wisc.edu`. Support for this research was provided by OVCRGE at UW–Madison with funding from the Wisconsin Alumni Research Foundation.

[†]Email: `saikumar@cs.wisc.edu`.

generate new keys for the TAs and other instructional staff, where the access policy for the staff can even be further restricted. Thus, this reduces the burden on the central authority as it does not need to generate secret keys for all receivers. A very popular specialization of DABE is the notion of hierarchical identity-based encryption (HIBE) [HL02, GS02]. HIBE is a very simple DABE, where the class of supported predicates are just prefix matching predicates (i.e., sk_f decrypts ct_x if f is a prefix of x). There are numerous constructions for DABE (and its specializations) in the literature [HL02, GS02, Wat05, GH09, Wat09, LOS+10, LW10, DG17b, DG17a, GVW13, BGG+14, ABG+13, BCG+17].

Due to the hierarchical nature of DABE, the notion of security has to be very carefully defined. In a few words, unlike vanilla ABE where each secret key can only be generated by the master key authority, secret keys in DABE can be computed by any honest user *and not just the master key authority*. Thus, an attacker can now not only corrupt secret keys generated by the master key authority, but also the secret keys generated by different users as part of key delegations. This significantly increases the scope of attackers, and makes the task of defining and proving full security of DABE significantly more challenging than vanilla ABE.

This gap between ABE and DABE security was first pointed out by Shi and Waters [SW08], who studied these definitional issues in the context of HIBE. Informally, Shi and Waters noted that, in delegatable encryption systems, the right approach to capture general adversaries is to let them initialize an arbitrary number of honest users and adaptively decide which users must be corrupted. Moreover, this process of honest user initialization and corrupting users can be arbitrarily interleaved. As an example, any reasonable security definition should capture the following attacker. Attacker $\mathcal{A}$ asks the challenger to initialize key sk_{f_1} for predicate f_1, then it asks it to delegate it to $\mathsf{sk}_{f_1 \wedge f_2}$, and later to $\mathsf{sk}_{f_1 \wedge f_2 \wedge f_3}$ or to $\mathsf{sk}_{f_1 \wedge f_4}$ (and so on), and meanwhile it can ask to corrupt any subset of these initialized keys (e.g., corrupt $\mathsf{sk}_{f_1 \wedge f_2}$).

While a real-world adversary should definitely be allowed to corrupt users as above, it seems unclear as to why proving full security of DABE is significantly more challenging than ABE. Briefly, the reason is that, in (vanilla) ABE, an attacker never initializes user keys but just directly corrupts them. Thus, whenever a challenger has to create a secret key for some predicate f, *in the pre-challenge query phase*, then it knows for a fact that $f(x^*) = 0$ (i.e., f will not satisfy the challenge attribute x^*). However, this guarantee is **not** available for $\underline{\text{D}}$ABE. Because an attacker can ask the challenger to initialize a secret key sk_f for any predicate f, even ask the challenger to use sk_f to generate delegated keys, moreover it can even corrupt those delegated keys, but the challenger still has no clue as to whether $f(x^*) = 0$ or 1. This uncertainty does not appear in vanilla ABE, thus a reduction algorithm can always safely set up the system parameters in a trapdoor way such that it can answer every secret key query. But in DABE, due to this additional power given to the adversary (in the form of asking for delegated keys computed from a *distinguishing* key), it is very difficult to generalize ABE proof techniques for proving full security for DABE.

In the case of HIBE, Lewko-Waters [LW14] also proved that full security cannot be proven (via black-box reductions) for schemes with certain checkability properties. Given DABE is much more expressive than HIBE, thus it is safe to say that proving full security of DABE faces even stronger barriers.

Our results: Fully Secure Delegatable ABE from Witness Encryption. In this work, we construct fully-secure DABE for all polynomial sized predicates from witness encryption [GGSW13] along with statistically-sound NIZKs. To the best of our knowledge, this is the first DABE construction (beyond HIBE) that achieves full security without relying on complexity leveraging, or an exponential number of hybrids. Our construction does not rely on Random Oracle [BR93] heuristics, and we provide a black-box reduction to polynomial hardness of underlying cryptographic primitives.

Although witness encryption does not yet have as many diverse constructions, recent works from evasive LWE assumption [Tsa22, VWW22] and new directions from pairing free groups [BIOW20] suggest this could change quickly. Furthermore, we view our work as opening a new direction for designing fully secure DABE, a problem for which we did not have any solutions outside of using complexity leveraging (i.e., sub-exponential security loss) before this work.

Our DABE supports an unbounded number of key delegations, and the secret key size grows just linearly with each key delegation operation. However, the encryptor must specify the maximum hierarchy depth for secret keys that can decrypt the resulting ciphertext. We leave the problem of designing fully unbounded DABE as an interesting open problem. We note that the only other prior work that could support an unbounded number of key delegations (beyond HIBE) was by Brakerski et al. [BCG$^+$17], who required collusion resistant general-purpose functional encryption [SW08, BSW11], and proved selective security[1]. Another prior work by Boneh et al. [BGG$^+$14] designed DABE that could support an-priori bounded number of delegations, and their secret key size grew quadratically with the number of delegations.

At a high level, our approach is based on a recent work by Waters and Wichs [WW24], who generalized the classic *dual-systems* methodology introduced by Waters [Wat09] for designing fully-secure ABE from bilinear pairings. We develop new techniques to generalize Waters' *dual-systems* methodology to delegatable ABE systems. At its core, we go beyond the classic concept of *semi-functional* keys and ciphertexts, as we show that for designing fully-secure DABE it is more appropriate to design "splittable" semi-functional keys. Abstractly, by *splittable* semi-functional keys we mean that secret keys can be split up such that only a portion of them can be indistinguishably made semi-functional, while the rest of it is not needed for answering key delegation queries. As we elaborate in the overview, splitting semi-functional keys is an important technical tool that lets us handle fully adaptive attackers in DABE.

Related Work. Since ABE was proposed in [SW05, GPSW06], it grasped significant attention from the community. It has been a versatile tool in constructing encrypted access control systems [PRV12, GKP$^+$13]. Numerous works constructed ABE with varying levels of security and functionality from several standard and post-quantum assumptions. A non-exhaustive list is as follows: [LOS$^+$10, GVW13, BGG$^+$14, Tsa19, Wee22, LLL22, HLL23, AKY24, HLL24].

Delegatable ABE is a "hierarchical" variant of ABE and seen limited success through the decades with [GVW13, BGG$^+$14] constructing selectively secure variants. A stronger version, delegatable FE is nearly equivalent to FE [BCG$^+$17, BS15]. A weaker version, selective hierarchical IBE is known to be equivalent to selective IBE [DG17b, DG17a].

Adaptive security is very challenging problem in ABE with only few approaches. These include those based on general-purposed functional encryption [GGH$^+$13, Wat15, ABSV15, AJS15, AJ15, BV15], pairing-based dual-systems methodology [Wat09, LOS$^+$10], and subset functionalities [Tsa19, GLW21]. Recently, [WW24] constructed adaptive ABE for any polynomial-size policy class from witness encryption which is weaker than functional encryption/ obfuscation.

[1]We point out that Brakerski et al. [BCG$^+$17] designed hierarchical/delegatable functional encryption, which is more general than DABE.

2 Technical Overview

In this section, we provide a high-level overview of techniques we used in constructing a delegatable ABE scheme for polynomial-size policies. We start by recalling the notion of delegatable ABE [HL02, GS02, GVW13, BGG+14] as an extension of (key-policy) ABE.

Delegatable ABE. An ABE scheme consists of four algorithms — $\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}$. Setup generates public parameters PP and master secret key MSK. KGen is used to *privately* delegate keys for policy f to create secret key SK_f. That is, KGen uses MSK and description of f to create SK_f. Enc encrypts a message μ and an attribute x to create a ciphertext CT_x. Dec using SK_f and CT_x outputs μ if and only if $f(x) = 1$.

A delegatable ABE (DABE) scheme facilitates for *public* delegation on top of private delegation facility of KGen. This is done using $\mathsf{Delegate}$ algorithm that uses secret key for policy f (SK_f) and description of g to generate $\mathsf{SK}_{f\wedge g}$. Note that MSK is not used for this process. Naturally, Dec now reveals μ if and only if for any $f = f_1 \wedge \ldots \wedge f_\ell$, $f(x) = 1 = (f_1(x) = 1) \wedge \ldots \wedge (f_\ell(x) = 1)$. We now define the adaptive security definition that we consider for DABE [SW08, BGG+14, BCG+17].

Adaptive security for DABE. The main idea in adaptive security for DABE is to protect the ciphertexts of users from active adversaries who can create and delegate secret keys for arbitrary policies but can only corrupt unsatisfying secret keys. That is, for a ciphertext CT_x, an attacker can create and delegate keys for arbitrary policies f. However, attacker will only corrupt SK_g if $g(x) = 0$.

To formulate this, we consider a game between a challenger and an attacker where the attacker asks the challenger to generate and "Store" secret key for policy f. The attacker receives a token h from challenger in response. Attacker can use h and g to "$\mathsf{Delegate}$" secret key SK_f with g to create $\mathsf{SK}_{f\wedge g}$ and receive a token to access this secret key. The attacker can also "$\mathsf{Corrupt}$" secret key for any policy p using the corresponding token if $p(x^*) = 0$ and receive SK_p. The attacker in addition will make a challenge query with (x^*, μ_0, μ_1) *adaptively*, i.e, at any point in the game. Note that there is no admissibility criterion on $f, f \wedge g$. They could be satisfying queries, i.e $f(x) = 1$ or $f(x) = 1 \wedge g(x) = 1$ or both. Challenger encrypts (x^*, μ_b) for randomly chosen b and attacker wins if it can guess b with non-negligible probability. As mentioned in introduction, this behavior of attacker to send x^* after making any of the previous queries is why achieving adaptive security is hard.

The major tool we use in our construction is witness encryption (WE). A WE scheme for an NP language ($\mathcal{L}$) can be thought of as a worst-case public-key encryption where the relation between public and secret key is determined by an NP relation (R). In particular, in a WE scheme, we encrypt a message (μ) using an instance $\mathtt{inst}$ that may or may not be part of $\mathcal{L}$. The resulting ciphertext can only be decrypted by using a witness $\mathtt{wit}$ such that $\mathsf{R}(\mathtt{inst}, \mathtt{wit}) = 1$, i.e, if and only if $\mathtt{wit}$ is a valid witness for the membership of $\mathtt{inst}$ in $\mathcal{L}$. Semantic security of WE can be argued as long as $\mathtt{inst} \notin \mathcal{L}$. With these definitions and goals, let's now look at construction of ABE from WE which we use to construct our fully secure DABE scheme.

Reviewing [GGSW13]. Here, we provide an overview of the *selectively-secure* ABE[2] from WE of Garg-Gentry-Sahai-Waters (GGSW) [GGSW13]. The public and master secret key of their system are a verification and signing key of a special dual-mode constrained signature scheme. Secret-key for f, is a signature (f, σ_f) generated in the normal mode. In order to encrypt (x^*, μ), GGSW uses

[2]In selectively-secure ABE, x^* is declared by attacker before receiving PP.

a WE scheme for the language that requires a policy g and signature σ_g such that (g, σ_g) is a valid signature pair *and* $g(x) = 1$.

In order to argue security, we constrain each signature to condition $f(x^*) = 0$ by moving to a "trapdoor" mode of generating signatures. That is, the signatures now verify only if $f(x^*) = 0$. Hence, the WE language will not be satisfiable anymore as it requires a valid signature and $f(x^*) = 1$. However, in GGSW x^* needs to be declared a-priori in order to constrain the signatures. Thus, only selective security is achievable. The signature scheme is constructed using commitment schemes (COM) and non-interactive zero-knowledge (NIZK) proof system for the following language.

$$\mathsf{PP} := \Big(\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(0; r^{(1)})\Big)$$

$$\mathsf{NIZK}.\mathcal{L} = \left\{ \mathtt{inst} := (\mathsf{PP}, f) \ : \begin{array}{c} \mathtt{wit} := r^{(0)} \text{ such that } \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}) \\ (\text{OR}) \\ \mathtt{wit} := (r^{(1)}, x^*) \text{ such that } \mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)}) \wedge \ f(x^*) = 0 \end{array} \right\}$$

In the normal mode, we use $r^{(0)}$ to generate proofs for policies which are used as signatures, i.e, $\sigma_f := \pi_f$. We switch to trapdoor mode by setting $\mathsf{com}^{(1)} = \mathsf{Com}(x^*)$ and $\mathsf{com}^{(0)} = \mathsf{Com}(1)$. If NIZK and Com are perfectly sound and binding, in the trapdoor mode, instance used in WE encryption will not be satisfiable.

Now, we will discuss our initial approach in making GGSW scheme delegatable and fully secure. We want to point out that this is *not* our final construction. Nevertheless, this discussion will be helpful as parts of this approach overlap with the proof techniques of our main construction. Towards the end of this discussion, we will point out a major issue in this approach later and provide additional methods we used to remedy this. At a high level, in this approach we compose proofs on top of π_f to construct a delegation chain which will render the scheme too inefficient. We expand on this below.

Adding delegation capacity to GGSW. In order to make the GGSW ABE delegatable, we need to first design a public delegation algorithm Delegate that uses (SK_f, g) to compute delegated $\mathsf{SK}_{f\wedge g}$. To the best of our knowledge, prior to this work it was not known how to design delegatable ABE using witness encryption. Our idea is to generate a metaproof [DSY91] for the NIZK verification language using $(f, \mathsf{SK}_f := \pi_f)$ as witness[3]. Let us elaborate.

In the Delegate algorithm, we will use (f, π_f) as witness to a NIZK language that also checks if $\mathsf{NIZK.Verify}(f, \pi_f) = 1$ in the normal mode. The resulting proof in conjunction with f, g will be $\mathsf{SK}_{f\wedge g}$ (we will *not* include π_f as part of $\mathsf{SK}_{f\wedge g}$). We also modify the WE language appropriately.

In order to argue *selective* security, the same ideas from GGSW flow naturally. If we rely on the recursive simulation property of metaproofs (that is, to simulate a metaproof, we start by simulating the base layer proof and then the first composed proof, and so on), we can simulate all the secret keys (that were stored either by performing KGen or Delegate). This yields two major advantages – (1) we will not be using $r^{(0)}$, (2) we will also not use π_f to generate $\pi_{f\wedge g}$ for Delegate queries. In addition, we also know the policies (both queried to Store and Delegate) that are unsatisfying as we know x^* a-priori. Thus, we can generate proofs in the trapdoor mode only for these policies. As NIZK and COM are perfectly sound and binding, we can argue that the instance in WE is not satisfiable similar to GGSW.

[3]We abuse the notation with $\mathsf{SK}_f = \pi_f$ and $\mathsf{SK}_f = (f, \pi_f)$ when the context is clear.

Adaptive DABE from GGSW. Making this scheme adaptive requires some additional technical tools. To this end, we look at the recent work of Waters-Wichs [WW24] that relied on a special type of functional encryption (FE) system, called Mixed-FE[4] [GKW18] which played an important part in executing the dual systems paradigm. Below, we briefly recall the definition of Mixed-FE before circling back to using it in our DABE scheme to realize adaptive security.

Reviewing Mixed-FE. A mixed functional encryption scheme (Mixed-FE) [GKW18] is a bounded-collusion[5] secure secret-key FE scheme. Although we know how to construct this FE from one-way functions [SS10, GVW12, AV19], a Mixed-FE scheme has one interesting feature that makes it non-trivial. The ciphertexts for any policy f can be generated in two modes – a private mode using $\mathsf{skEnc}(\mathsf{msk}, f)$ and a public mode using $\mathsf{pkEnc}(f)$. $(\mathsf{skEnc}, \mathsf{KGen})$ of Mixed-FE work like FE and reveal $f(x)$. $(\mathsf{pkEnc}, \mathsf{KGen})$ always outputs 0. Security is defined as an indistinguishability game where an attacker makes unbounded queries for ciphertexts of policies $\{f_i\}_i$ and a *single* secret key for input x adaptively[6]. This attacker cannot distinguish between the modes in which ciphertexts are generated for any policy as long as $\forall i, f_i(x) = 0$. Full definition of Mixed-FE is provided in Definition 3.5. Now, let's look at how we can use Mixed-FE to boost security of our DABE construction.

Adaptive DABE via Mixed-FE. Using Mixed-FE similar to [WW24], we can more or less realize adaptive security readily. Essentially, $(\mathsf{Mix\text{-}FE.KGen}, \mathsf{Mix\text{-}FE.pkEnc})$ act as a functional system and $(\mathsf{Mix\text{-}FE.KGen}, \mathsf{Mix\text{-}FE.skEnc})$ act as a semi-functional system. The modes are indistinguishable readily from $\mathsf{Mix\text{-}FE}$ security.

More precisely the changes are as follows — in KGen and $\mathsf{Delegate}$, we sample $\mathsf{ct}_f \leftarrow \mathsf{pkEnc}(f)$ and use these as part of SK_f. Similarly, in Enc, we sample msk and use $\mathsf{sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{msk}, x)$ as part of CT_x. We also modify the WE language to check $0 \overset{?}{=} \mathsf{Mix\text{-}FE.Dec}(\cdot,\cdot) \wedge f(x) \overset{?}{=} 1$. As values from $\mathsf{pkEnc}, \mathsf{Mix\text{-}FE.KGen}$ always output 0, correctness is unaltered. But in order to argue adaptive security, we shouldn't be committing x^* and checking $f(x^*) \overset{?}{=} 0$ in the NIZK trapdoor mode. Instead, we should commit $\mathsf{Mix\text{-}FE.msk}$ to check if we are in the semi-functional mode, i.e, $\exists\, r$ such that $\mathsf{ct}_f = \mathsf{Mix\text{-}FE.skEnc}(\mathsf{Mix\text{-}FE.msk}, f; r)$.

To argue adaptive security of DABE, first observe that once we are in trapdoor mode where we use semi-functional system, if π_f verifies, it means $0 = f(x) = \mathsf{Mix\text{-}FE.Dec}(\mathsf{sk}_x, \mathsf{ct}_f)$. Thus WE instance is always unsatisfiable! However, in order generate proofs in the trapdoor mode, we need to set $\mathsf{com}^{(0)} = \mathsf{Com}(1)$ and generate all ct_f using $\mathsf{Mix\text{-}FE.skEnc}$ as otherwise we will generating proofs for invalid instances. Thus, we need to switch to the semi-functional system before invoking the trapdoor mode. Switching to semi-functional system in adaptive game for DABE poses additional challenges. Specifically, an attacker can ask the challenger to generate keys for satisfying policies $(f(x^*) = 1)$ as well. In that case, we will be generating ct_f for satisfying policies and we cannot rely on Mixed-FE security readily.

[4]We point that [WW24] introduced a new notion that they refer to as functional tags, constructed them from one-way functions, and used functional tags instead of Mixed-FE. However, we observe that functional tags are merely an alternate approach to define a 1-query bounded version of Mixed-FE [GKW18]. We find it a bit easier to explain our approach using the Mixed-FE framework. Moreover, we believe that viewing this abstract dual-systems technique from the lens of Mixed-FE could be more meaningful for future work in adaptive security.

[5]By bounded-collusion, we mean that security is only going to hold against attackers that corrupt an a-priori bounded number of secret keys.

[6]Note that [GKW18, CVW$^+$18] proposed and constructed the "dual" notion of our Mixed-FE where indistinguishability holds against an attacker that possesses unbounded number of secret keys and a-priori bounded (not necessarily one) ciphertexts. In addition, their ciphertexts generated by pkEnc always output 1.

In order to fix this issue, we observe that we do not need to generate keys for all policies when the attacker queries for it. We can delay this generation to $\mathsf{Corrupt}$ phase when the attacker actually requires a secret key. At this point, we know that this policy for which we need to send a secret key to attacker is unsatisfying. Hence, we can readily generate ct_f and rely on Mixed-FE security. In summary based on the changes mentioned to the construction above, the flow of hybrids roughly look as follows — (1) use recursive simulation to simulate all NIZK proofs (2) delay secret key generation (for KGen or $\mathsf{Delegate}$) and wait for $\mathsf{Corrupt}$ query (3) switch to the semi-functional system (4) enter trapdoor mode in NIZK for all corrupted keys (5) rely on WE security.

Limitations of this approach. It looks like by generating metaproofs and relying on Mixed-FE, we can construct a DABE scheme from the GGSW construction. However, there is a strong technical hurdle that makes the above design for delegatable ABE very weak. The issue is that if size of NIZK proof $|\pi| = \mathsf{poly}(\lambda, |\mathtt{inst}|, |\mathtt{wit}|)$, then running time of $\mathsf{NIZK.Verify}$ is at least $\mathsf{poly}(\lambda, |\mathtt{inst}|, |\mathtt{wit}|)$. When we use another proof on top of it, size and running time of prover will grow proportional to $|\pi|$. After d many layers of delegation, size of proof will be some $\mathsf{poly}(\lambda)^d$. Thus, we cannot delegate more that $O(1)$ times with this strategy.

One straight-forward way of fixing our issues is to use rate-1 NIZK schemes. If $|\pi| = |\mathtt{wit}| + \mathsf{poly}(\lambda)$, composing d proofs will result in a proof of size at most $d \cdot |\mathtt{wit}| + \mathsf{poly}(\lambda)$. But we need additional requirements for NIZK – (a) perfect (or statistical) soundness (b) $\mathsf{NIZK.Prove}$ should run in at most $\mathsf{poly}(\lambda, |\mathtt{wit}|, \log |\mathtt{inst}|)$ time. We need (a) so that we could rely on witness encryption security, and (b) so that $\mathsf{Delegate}$ remains efficient. Constructions of rate-1 NIZKs are known from fully homomorphic encryption [GGI+15] or from batch arguments [ACG+24, BDS24]. However, none of these constructions meet requirements (a + b) simultaneously. Hence, it looks like using our basic template, we will be stuck at $d = O(1)$ number of delegations. This is far from the desirable goal of arbitrary key delegation capabilities.

Going beyond $O(1)$ delegations. At this point, in order to go beyond constant number of delegations, we need to come up with a new design. The core issue in the previous construction is the metaproof strategy where we compose proofs. We did so to make sure that there is a connection between the secret key generated by KGen and any delegated key, yet a delegated key does not contain all information about its parent's key. The latter point is crucial in proving full security, since if a delegated key contains a lot of non-trivial information about its parent key, then it is unclear if the resulting scheme will still be secure.

In what follows, we look beyond basic proof composition techniques to capture key delegation capabilities. Our strategy is to rather chain proofs. That is, instead of composing proofs, we will generate new proofs based on the components of previous keys. Let us discuss this more thoroughly.

We view each secret key to be *splittable* into two components – a public part that can be given out as part of future delegated keys, and a hidden part that is removed during delegation and only needed when a user wants to decrypt a WE ciphertext. This way, we will still use the hidden part of SK_f to generate a (delegated) secret key for $f \wedge g$, but it won't get "leaked" at all from the delegated key. In particular, $\mathsf{SK}_{f \wedge g}$ will contain the public part SK_f along with some new public components, and a new hidden component. This way, anyone in possession of $\mathsf{SK}_{f \wedge g}$ can only delegate on top of $f \wedge g$. Importantly, the hidden part of SK_f will not appear in $\mathsf{SK}_{f \wedge g}$.

Expanding on this further, in order to achieve *selectively* secure DABE with unbounded collusions, our main idea is that we will add *two* new commitments of 0 to each secret key. One of these will be used to make the switch to trapdoor mode where the other will be used to delegate further. That is, $\mathsf{SK}_f = (f, \pi_f, \mathsf{com}_f, \underline{\mathsf{com}_f^{(0)}, r})$ where $\mathsf{com}_f \leftarrow \mathsf{Com}(0)$ and $\mathsf{com}_f^{(0)} = \mathsf{Com}(0; r)$. π_f now also verifies if com_f and $\mathsf{com}_f^{(0)}$ are commitments of 0 in the normal mode. In order to construct a delegated key for $f \wedge g$, we will sample two new commitments of 0, $\mathsf{com}_{f\wedge g}, \mathsf{com}_{f\wedge g}^{(0)}$ and sample a NIZK proof for the language that checks if $\mathsf{com}_{f\wedge g}$, $\mathsf{com}_{f\wedge g}^{(0)}$ are commitments of 0 *and* $\underline{\mathsf{com}_f^{(0)}} \stackrel{?}{=} \mathsf{Com}(0; \underline{r})$. Essentially, we will use $\mathsf{com}_f^{(0)}$ as the new PP and r as the new master secret key to delegate the key. Crucially, instance or witness for this NIZK proof *does not* include π_f. This will generate $\pi_{f\wedge g}$ and we will set $\mathsf{SK}_{f\wedge g} = (f, \pi_f, \mathsf{com}_f, g, \pi_{f\wedge g}, \mathsf{com}_{f\wedge g}, \underline{\mathsf{com}_{f\wedge g}^{(0)}, r'})$. We will also modify WE language that checks if each and every proof in $\mathsf{SK}_{f_1\wedge\ldots\wedge f_\ell}$ verifies, if the last commitment in the chain is valid, and $f_1 \wedge \ldots f_\ell(x) = 1$.

Note that, the size of $\pi_{f\wedge g}$ doesn't grow drastically at each link and size of $\mathsf{SK}_{f_1\wedge\ldots\wedge f_\ell}$ only grows linearly in ℓ. The link between SK_f and $\mathsf{SK}_{f\wedge g}$ maintained using $\mathsf{com}_f^{(0)}$. Thus, instead of metaproofs, a chaining of commitments and proofs bypasses the efficiency issue that we faced earlier. It turns out that we can argue *selective* security similarly to our original (metaproof-based) construction. In a few words, when we switch to trapdoor mode where we set com_f for $f = f_1 \wedge \ldots \wedge f_\ell$ to be a commitment of 1, for any secret key where each $\{\pi_i\}_{i\in[\ell]}$ verifies, there will be $\ell^* \leq \ell$ such that $f_1 \wedge \ldots f_{\ell^*}(x) = 0$ which makes the WE instance unsatisfiable.

We remark that prior to this work, the only known construction for delegatable ABE (that did not rely on general-purpose obfuscation) was by Boneh et al. [BGG+14], and their construction could only support a '*fixed*' number of key delegations. Our above witness encryption based approach is the first DABE construction in which secret keys can be delegated an unbounded number of times, even when we set the final goal to be just selective security. Next, we show that the above design can be further improved to resist fully adaptive attackers.

Adaptive Security from Mixed-FE. It stands to reason that if we use Mixed-FE similarly to the metaproof construction, we could realize adaptive security readily. However, there is a major flaw in that argument. If we generate ct_f and include it as part of SK_f, we will be giving away $\{\mathsf{ct}_{f_i}\}_i$ for a chain of $f_1, \ldots, f_\ell$ as part of SK_f. This will be problematic as we cannot switch to the semi-functional system of Mixed-FE. Recall that Mixed-FE security requires that ciphertexts should be generated only for 'unsatisfying' policies. While any function $f = f_1 \wedge \ldots \wedge f_\ell$ (for which the attacker ever receives a key for) must be unsatisfying, it could be that f_1 is a satisfying policy. Thus, if we want to rely on Mixed-FE security, then we must not give away ct_{f_1} to the attacker. This is because then an adaptive attacker can trivially distinguish between honest and trapdoor modes, by simply trying to decrypt ct_{f_1}.

Splittable semi-functional keys. In order to hide ct_f, we once again rely on our idea that ct_f is only needed when a $\mathsf{Corrupt}$ query has to be answered, thus its generation can be delayed. That is, ct_f can also be treated as a <u>hidden</u> part of the key. In particular, we set $\mathsf{SK}_f = (f, \mathsf{com}_f, \pi_f, \underline{\mathsf{com}_f^{\mathsf{ct}}, \mathsf{ct}_f, r_f})$ and $\underline{\mathsf{com}_f^{\mathsf{ct}}} = \mathsf{Com}(\underline{\mathsf{ct}_f}; \underline{r_f})$ where π_f now ensures that this relation between $\underline{\mathsf{com}_f^{\mathsf{ct}}}$ and $\underline{\mathsf{ct}_f}$ is valid. And then delegation proceeds similar to the selectively secure version where we will create $\mathsf{com}_{f\wedge g}, \mathsf{com}_{f\wedge g}^{\mathsf{ct}}$ and generate $\pi_{f\wedge g}$ that checks the relation between $\mathsf{com}_f^{\mathsf{ct}}, \mathsf{ct}_f$ and $\mathsf{com}_{f\wedge g}^{\mathsf{ct}}, \mathsf{ct}_{f\wedge g}$. With this, we can sample the public part whenever a key query is made but we can compute the hidden part only when a $\mathsf{Corrupt}$ query is made. As only the <u>hidden</u> portion is semi-functional, we call this a *splittable* semi-functional key. Adaptive security of the scheme can be argued similarly to the $O(1)$ delegation construction where we simulate all NIZK

proofs to stop relying on previous proofs. We will then switch to the semi-functional system and then start to use the trapdoor mode. A high-level sketch of our final construction looks as follows.

Setup. We will generate two commitments $\mathsf{com}^{(0)}, \mathsf{com}^{(1)}$ where $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{nizk.crs} \leftarrow \mathsf{NIZK.Setup}$, and $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}$. We will use $\mathsf{PP} := (\mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{nizk.crs}, \mathsf{mfe.pp})$ and $\mathsf{MSK} := r^{(0)}$.

Key Generation. Sample $\mathsf{ct}_f \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f)$, $\mathsf{com}_f^{\mathsf{ct}} = \mathsf{Com}(\mathsf{ct}_f; r)$, and $\mathsf{com}_f = \mathsf{Com}(0; r')$ Using $(r^{(0)}, r, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{ct}_f, r')$ as witness, generate π_f that checks the validity of $\mathsf{com}^{(0)}, \mathsf{com}_f, \mathsf{com}_f^{\mathsf{ct}}$. Set $\mathsf{SK}_f := (f, \mathsf{com}_f, \pi_f, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{ct}_f, r')$.

Delegation. Given secret key $(f_1, \mathsf{com}_1, \pi_1, \ldots, f_\ell, \mathsf{com}_\ell, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{ct}_\ell, r')$ and policy g, sample $\mathsf{ct}_{\ell+1} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots \wedge f_\ell \wedge g)$, $\mathsf{com}_{\ell+1}^{\mathsf{ct}} = \mathsf{Com}(\mathsf{ct}_{\ell+1}; r)$, and $\mathsf{com}_{\ell+1} = \mathsf{Com}(0; r'')$. Using $(r'', \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{ct}_\ell, r', \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{ct}_{\ell+1}, r)$ as witness, generate π_f that checks the validity of $\mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{com}_{\ell+1}, \mathsf{com}_\ell^{\mathsf{ct}}$. Set $\mathsf{SK}_f := (f_1, \mathsf{com}_1, \pi_1, \ldots, f_\ell, \mathsf{com}_\ell, \pi_\ell, g, \mathsf{com}_{\ell+1}, \pi_{\ell+1}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{ct}_{\ell+1}, r'')$.

Encryption. Sample a master secret key msk for $\mathsf{Mix\text{-}FE}$ and generate sk_x for the attribute x. Generate a WE ciphertext for the language where $\mathtt{inst} := (\mathsf{PP}, x, \mathsf{sk}_x)$ and μ where the relationship circuit uses SK_f and checks if each π_i is valid and $\mathsf{Mix\text{-}FE.Dec}(\mathsf{ct}_f, \mathsf{sk}_x) = 0$, and $f(x) = 1$. Output $\mathsf{CT} := (x, \mathsf{sk}_x, \mathsf{we.ct})$.

Trapdoor modes for all the NIZK proofs remain same. Moreover, decryption follows naturally. Note that delegation chain can run as long as possible or until the NIZK proof can handle it. So, if we have a NIZK scheme that can prove unbounded $\mathtt{inst}$ using unbounded $\mathtt{wit}$, we can handle unbounded delegation. [GGI+15] constructs such a statistically sound NIZK scheme from fully homomorphic encryption. In addition, we also point out a few alternate approaches to achieve full delegation in Section 7. On the flip side, we need an upper bound on the witness size for WE scheme and thus we can only decrypt a given ciphertext using secret keys that are delegated a bounded d number of times. It is an interesting question to design DABE where a ciphertext can be decrypted by a secret key generated by an unrestricted sequence of delegation operations.

Another technical subtlety is that size of Mixed-FE ciphertext grows with delegation. If we use Mixed-FE obtained via lockable obfuscation [GKW17, WZ17] like in [CVW+18], we can handle unbounded size. This is because encryption in 1-key secret-key FE from [SS10] can handle unbounded size functions as long as input and output sizes are used as global parameters. The same cannot be said about functional tag system [WW24] that crucially relies on maximum size of policies as a global parameter. Hence, if we use functional tag system instead of Mixed-FE, we can only handle bounded delegations. As a straight-forward corollary of our main result, we get a fully secure delegatable (one-sided) predicate encryption scheme for any polynomial-size policy class using lockable obfuscation [GKW17, WZ17]. We provide full details of our construction in Section 5.

3 Preliminaries

Notation. By PPT we denote probabilistic polynomial-time. We denote the security parameter by λ and the set of positive integers by $\mathbb{N}$. For any $a, b \in \{0\} \cup \mathbb{N}$, $a \leq b$, we denote by $[a, b]$, the set of all integers from a to b including a and b. In other words, $[a, b] = \{a, \ldots, b\}$. We denote by $[n] := [1, n]$. We denote by $x \xleftarrow{\$} \mathcal{X}$, the process of sampling an element x from the set $\mathcal{X}$, with uniform probability. Similarly, for any PPT algorithm $\mathcal{A}$, $x \leftarrow \mathcal{A}(y)$ denotes the process of sampling x from the output distribution of $\mathcal{A}$ when run on y. By $\mathsf{negl}(\cdot)$, we denote negligible functions. By $\mathsf{poly}(\cdot)$, we denote positive polynomials.

We say that two efficiently samplable probability distributions $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if for any non-uniform PPT distinguisher $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$, and for large enough $\lambda \in \mathbb{N}$,

$$\left| \Pr_{\alpha\leftarrow\mathcal{X}_\lambda}\left[1 \leftarrow \mathcal{D}(1^\lambda, \alpha)\right] - \Pr_{\alpha\leftarrow\mathcal{Y}_\lambda}\left[1 \leftarrow \mathcal{D}(1^\lambda, \alpha)\right] \right| \leq \mathsf{negl}(\lambda)$$

For all preliminaries below, $\mu \in \{0,1\}^*$. Note that for encryption schemes we can use hybrid encryption to encrypt messages of arbitrary length.

3.1 Statistically Binding Commitments

Syntax. A statistically binding commitment scheme (COM) consists of the following polynomial time algorithms.

$\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. The probabilistic setup algorithm takes as input security parameter λ and outputs a common reference string crs. The following algorithms take crs implicitly.

$\mathsf{Com}(\mu; r) \to \mathsf{com}$. The probabilistic commitment algorithm takes as input message μ, randomness r, and outputs commitment value com.

Definition 3.1 (COM)**.** A COM scheme $(\mathsf{Setup}, \mathsf{Com})$ is said to be a COM scheme if it satisfies the following properties.

Computational Hiding. For any stateful PPT adversary $\mathcal{A}$, there exists a negligible function such that $\forall\ \lambda \in \mathbb{N}$,

$$\Pr\left[\ b \leftarrow \mathcal{A}^{\mathcal{O}_b(\cdot,\cdot)}(\mathsf{crs}) \quad : \quad \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), b \xleftarrow{\$} \{0,1\}\ \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathcal{O}_b(\mu_0, \mu_1)$ responds with $\mathsf{com} \leftarrow \mathsf{Com}(\mathsf{crs}, \mu_b)$.

Statistical Binding. There exists $\mathsf{negl}(\cdot)$ such that $\forall\ \lambda \in \mathbb{N}$,

$$\Pr\left[\begin{array}{l} \mathsf{com} = \mathsf{Com}(\mu_0; r_0)\wedge \\ \mathsf{com} = \mathsf{Com}(\mu_1; r_1) \end{array} : \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), \\ \exists\ (\mathsf{com}, \mu_0, \mu_1, r_0, r_1) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where probability is taken on the randomness of crs.

Remark 3.2 ([Nao91])**.** Statistically binding commitments in the CRS model can be constructed by assuming one-way functions via length tripling PRGs.

3.2 Non-Interactive Zero-Knowledge Proofs

Syntax. A non-interactive zero-knowledge proof system (NIZK) for $\mathcal{L}_s = \{x : \exists\, w, \mathsf{R}_s(x, w) = 1\}$ consists of the following polynomial time algorithms.

$\mathsf{Setup}(1^\lambda, 1^s) \to \mathsf{crs}$. The probabilistic setup algorithm takes as input security parameter λ, language index s, and outputs a common reference string crs. The following algorithms take crs implicitly.

$\mathsf{Prove}(x, w) \to \pi$. The probabilistic proving algorithm takes as input instance $x \in \mathcal{L}$, witness w, and outputs proof π.

$\mathsf{Verify}(x, \pi) \to 0/1$. The deterministic verification algorithm takes as input instance x, proof π, and outputs 0 (reject) or 1 (accept).

Definition 3.3 (NIZK). A NIZK scheme $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is said to be a NIZK scheme for language $\mathcal{L}_s$ if it satisfies the following properties.

Correctness. $\forall\, \lambda \in \mathbb{N}, x \in \mathcal{L}$, $\Pr[1 = \mathsf{Verify}(x, \pi) : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^s), \pi \leftarrow \mathsf{Prove}(x, w)] = 1$.

Statistical Soundness. There exists a negligible function $\mathsf{negl}(\cdot)$ such that $\forall\, \lambda \in \mathbb{N}, s = s(\lambda), x \notin \mathcal{L}_s, \forall\, \pi$, $\Pr[1 = \mathsf{Verify}(x, \pi) : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^s), \exists\, \pi] \leq \mathsf{negl}(\lambda)$.

Computational Zero Knowledge. For any admissible adversary $\mathcal{A}$, there exists a stateful simulator Sim such that $\forall\, \lambda \in \mathbb{N}, s = s(\lambda)$,

$$\Bigg| \Pr\left[1 \leftarrow \mathcal{A}^{\mathcal{O}_0(\cdot,\cdot)}(1^\lambda, \mathsf{crs}) : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^s)\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathcal{O}_1^{\mathsf{Sim}(\cdot)}(\cdot,\cdot)}(1^\lambda, \mathsf{crs}) : \mathsf{crs} \leftarrow \mathsf{Sim}(1^\lambda, 1^s)\right] \Bigg| \leq \mathsf{negl}(\lambda)$$

where $\mathcal{O}_0$ uses $\pi \leftarrow \mathsf{Prove}(x, w)$ and $\mathcal{O}_1$ uses $\pi \leftarrow \mathsf{Sim}(x)$ to respond to $\mathcal{A}$'s queries of the form (x, w). A stateful PPT machine is said to be admissible if for each of its queries, $\mathsf{R}_s(x, w) = 1$.

3.3 Witness Encryption

Syntax. A witness encryption (WE) scheme for language $\mathcal{L}_s = \{x : \exists\, w, \mathsf{R}_s(x, w) = 1\}$ consists of the following polynomial time algorithms.

$\mathsf{Enc}(1^\lambda, 1^s, x, \mu) \to \mathsf{CT}$. The probabilistic encryption algorithm takes as input the security parameter λ, language index s, instance $x \in \mathcal{L}_s$, a message μ, and outputs a ciphertext CT.

$\mathsf{Dec}(\mathsf{CT}, w) \to \mu'$. The deterministic decryption algorithm takes as input a ciphertext CT, a witness w, and outputs a message μ'.

Definition 3.4 (WE). A WE scheme $(\mathsf{Enc}, \mathsf{Dec})$ is said to be a WE scheme for language $\mathcal{L}_s$ if it satisfies the following properties.

Correctness. $\forall\, \lambda \in \mathbb{N}$, $\mu, x \in \mathcal{L}_s$, if $\mathsf{R}_s(x, w) = 1, \Pr[\mu = \mathsf{Dec}(\mathsf{CT}, w) : \mathsf{CT} \leftarrow \mathsf{Enc}(1^\lambda, 1^s, x, \mu)] = 1$.

Semantic Security. For any stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\forall\ \lambda \in \mathbb{N}, x \notin \mathcal{L}, |\mu_0| = |\mu_1|$,

$$\Pr\left[\ b \leftarrow \mathcal{A}(\mathsf{CT}) \quad : \quad \begin{array}{l} (1^s, \mu_0, \mu_1) \leftarrow \mathcal{A}(1^\lambda), b \xleftarrow{\$} \{0,1\}, \\ \mathsf{CT} \leftarrow \mathsf{Enc}(1^\lambda, 1^s, x, \mu_b) \end{array}\ \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

3.4 Mixed Functional Encryption

Syntax. A mixed functional encryption scheme (Mixed-FE) for any polynomial-size policies consists of the following polynomial-time algorithms.

$\mathsf{Setup}(1^\lambda, 1^n) \rightarrow \mathsf{PP}$. The probabilistic setup algorithm takes as input security parameter λ, input size n, and outputs public parameters PP. The following algorithms take PP implicitly.

$\mathsf{Gen}(\mathsf{PP}) \rightarrow \mathsf{MSK}$. The probabilistic master key generation algorithm takes as input public parameters PP and outputs master secret key MSK.

$\mathsf{KGen}(\mathsf{MSK}, x) \rightarrow \mathsf{SK}_x$. The possibly randomized key generation algorithm takes as input master secret key MSK, input x, and outputs secret key SK_x.

$\mathsf{skEnc}(\mathsf{MSK}, f) \rightarrow \mathsf{CT}$. The probabilistic *secret key* encryption algorithm takes as input master secret key MSK, policy f, and outputs ciphertext CT.

$\mathsf{pkEnc}(f) \rightarrow \mathsf{CT}$. The probabilistic *public-key* encryption algorithm takes as input policy f and outputs ciphertext CT.

$\mathsf{Dec}(\mathsf{SK}_x, \mathsf{CT}) \rightarrow 0/1$. The deterministic decryption algorithm takes as input secret key SK_x, ciphertext CT, and outputs either 0 or 1.

Definition 3.5 (Mixed-FE). A $\mathsf{Mix\text{-}FE}$ scheme $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{skEnc}, \mathsf{pkEnc}, \mathsf{Dec})$ is said to be a Mixed-FE scheme for policy class $\mathcal{F}$ if it satisfies the following properties.

Correctness. There exist negligible functions $\mathsf{negl}_1(\cdot), \mathsf{negl}_2(\cdot)$ such that for any $\lambda \in \mathbb{N}$,

$$\Pr\left[\ f(x) = \mathsf{Dec}(\mathsf{SK}_x, \mathsf{CT}) \quad : \quad \begin{array}{l} \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^n), \mathsf{MSK} \leftarrow \mathsf{Gen}(\mathsf{PP}), \\ \mathsf{SK}_x \leftarrow \mathsf{KGen}(\mathsf{MSK}, x), \mathsf{CT} \leftarrow \mathsf{skEnc}(\mathsf{MSK}, f) \end{array}\ \right] \geq 1 - \mathsf{negl}_1(\lambda)$$

$$\Pr\left[\ 0 = \mathsf{Dec}(\mathsf{SK}_x, \mathsf{CT}) \quad : \quad \begin{array}{l} \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^n), \mathsf{MSK} \leftarrow \mathsf{Gen}(\mathsf{PP}), \\ \mathsf{SK}_x \leftarrow \mathsf{KGen}(\mathsf{MSK}, x), \mathsf{CT} \leftarrow \mathsf{pkEnc}(f) \end{array}\ \right] \geq 1 - \mathsf{negl}_2(\lambda)$$

Mode Indistinguishability. For any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\forall\ \lambda \in \mathbb{N}$,

$$\Pr\left[\ b \leftarrow \mathcal{A}^{\mathcal{O}_b(\cdot)}(\mathsf{CT}) \quad : \quad \begin{array}{l} b \xleftarrow{\$} \{0,1\}, 1^n \leftarrow \mathcal{A}(1^\lambda), \\ \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^n), \\ \mathsf{MSK} \leftarrow \mathsf{Gen}(\mathsf{PP}), \\ x \leftarrow \mathcal{A}^{\mathcal{O}_b(\cdot)}(\mathsf{PP}), \\ \mathsf{SK}_x \leftarrow \mathsf{KGen}(\mathsf{MSK}, x) \end{array}\ \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where, $\mathcal{O}_0(f)$ outputs $\mathsf{pkEnc}(f)$ and $\mathcal{O}_1(f)$ outputs $\mathsf{skEnc}(\mathsf{MSK}, f)$. A stateful PPT machine $\mathcal{A}$ is said to be admissible if for each input x queried to $\mathcal{O}_b$, $f(x) = 0$.

Remark 3.6. The above definition is a weaker version of Mixed-FE which was originally proposed in [GKW18, CVW$^+$18]. Specifically, the definition only considers one key generation query and we forgo function indistinguishability. Furthermore, we switched the "accept" condition for pkEnc in order to align with the classical definition for ABE (i.e, μ is revealed when $f(x) = 1$).

Remark 3.7. Assuming that the delegation depth is bounded, Definition 3.5 can be instantiated from one-way functions [WW24]. However, for unbounded delegations, the size of policies grow arbitrarily and we require lockable obfuscation to construct this similar to [GKW18, CVW$^+$18].

4 Delegatable Attribute-Based Encryption

In this section, we provide the definition of a delegatable ABE (DABE) scheme.

Syntax. A DABE scheme for the policy class $\mathcal{F} = \{\mathcal{F}_s\}_{s\in\mathbb{N}}$[7] consists of the following polynomial time algorithms.

$\mathsf{Setup}(1^\lambda, 1^s) \to (\mathsf{PP}, \mathsf{MSK})$**.** The probabilistic setup algorithm takes as input security parameter λ, functionality index s, outputs public parameters PP and master secret key MSK.

$\mathsf{KGen}(\mathsf{MSK}, f) \to \mathsf{SK}_f$**.** The possibly randomized key generation algorithm takes as input master secret key MSK, policy f, and outputs secret key SK_f.

$\mathsf{Enc}(1^d, x, \mu) \to \mathsf{CT}_x$**.** The probabilistic encryption algorithm takes delegation depth d, attribute x, message $\mu \in \{0,1\}^*$, and outputs ciphertext for attribute x, CT_x.

$\mathsf{Delegate}(\mathsf{SK}_f, g) \to \mathsf{SK}_{f\wedge g}$**.** The possibly randomized delegation algorithm takes as input a secret key for policy f, SK_f, policy g and outputs a secret key for the policy $f \wedge g, \mathsf{SK}_{f\wedge g}$.

$\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}) \to \mu'/\bot$**.** The deterministic decryption algorithm takes as input secret key for policy f, ciphertext for attribute x, CT_x, and outputs message μ' or aborts and outputs $\bot$.

Definition 4.1 (Delegatable ABE)**.** A DABE scheme $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Delegate})$ is said to be a DABE scheme for $\mathcal{F}_s$ if it satisfies the following properties.

Correctness. There exist negligible functions $\mathsf{negl}_1(\cdot), \mathsf{negl}_2(\cdot)$ such that for any $\lambda \in \mathbb{N}, s = s(\lambda), f \in \mathcal{F}_s, x \in \mathcal{X}_s$,

$$\Pr\left[\mu = \mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}_x) \quad : \quad \begin{array}{l} (\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s), \\ \mathsf{SK}_f \leftarrow \mathsf{KGen}(\mathsf{MSK}, f), \\ \mathsf{CT}_x \leftarrow \mathsf{Enc}(x, \mu), f(x) = 1 \end{array}\right] \geq 1 - \mathsf{negl}_1(\lambda)$$

$$\Pr\left[\bot = \mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}_x) \quad : \quad \begin{array}{l} (\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s), \\ \mathsf{SK}_f \leftarrow \mathsf{KGen}(\mathsf{MSK}, f), \\ \mathsf{CT}_x \leftarrow \mathsf{Enc}(x, \mu), f(x) = 0 \end{array}\right] \geq 1 - \mathsf{negl}_2(\lambda)$$

[7]We use $s = s(\lambda)$ as an all-encompassing index that bestows the user with all the necessary information about the policy class in question.

Delegation Correctness. This is defined similarly to correctness. Except that the policy f can now be parsed as $(f_1 \wedge \ldots \wedge f_\ell)$ for some $\ell \in [d]$. Here, SK_f is defined as follows —

$$\mathsf{SK}_{f_1} \leftarrow \mathsf{KGen}(\mathsf{MSK}, f_1) \text{ and for } i \in [2, \ell], \mathsf{SK}_{f_1 \wedge \ldots \wedge f_i} \leftarrow \mathsf{Delegate}(\mathsf{SK}_{f_1 \wedge \ldots \wedge f_{i-1}}, f_i)$$

Dec with high probability now outputs $\perp$ if for any $i \in [\ell], f_i(x) = 0$ and μ otherwise.

Adaptive Security. For any admissible adversary, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\forall\ \lambda \in \mathbb{N}, |\mu_0| = |\mu_1|$,

$$\Pr\left[b \leftarrow \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(\mathsf{CT}) \quad : \quad \begin{array}{l} 1^s \leftarrow \mathcal{A}(1^\lambda), b \xleftarrow{\$} \{0,1\}, \\ (\mathsf{PP}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^s), \\ (x^*, \mu_0, \mu_1) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(\mathsf{PP}), \\ \mathsf{CT} \leftarrow \mathsf{Enc}(x^*, \mu_b) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathcal{O}$ is a stateful oracle that initiates $\mathsf{h} := 1$ and answers these queries:

$(\mathsf{Store}, f) \rightarrow \mathsf{h}$**.** $\mathcal{A}$ sends policy f. $\mathcal{O}$ samples $\mathsf{SK}_f \leftarrow \mathsf{KGen}(\mathsf{MSK}, f)$, stores $(\mathsf{SK}_f, \mathsf{h}, \perp)$, and sends with h to $\mathcal{A}$. Also, update $\mathsf{h} := \mathsf{h} + 1$

$(\mathsf{Corrupt}, \mathsf{h}') \rightarrow \mathsf{SK}_f$**.** $\mathcal{A}$ sends handle h'. If there is no tuple of the form $(\mathsf{SK}_f, \mathsf{h}', *)$, output $\perp$. Otherwise, send SK_f to $\mathcal{A}$.

$(\mathsf{Delegate}, \mathsf{h}', g) \rightarrow \mathsf{h}$**.** $\mathcal{A}$ sends handle h' and policy g. If there is no tuple of the form $(\mathsf{SK}_f, \mathsf{h}', *)$, output $\perp$. Otherwise, $\mathcal{O}$ samples $\mathsf{SK}_{f \wedge g} \leftarrow \mathsf{Delegate}(\mathsf{SK}_f, g)$, stores $(\mathsf{SK}_{f \wedge g}, \mathsf{h}, \mathsf{h}')$ and sends h to $\mathcal{A}$. Also, update $\mathsf{h} := \mathsf{h} + 1$.

A stateful PPT machine $\mathcal{A}$ is said to be admissible if for any $\mathsf{Corrupt}$ query made by $\mathcal{A}$, SK_f is for a policy such that $f(x^*) = 0$.

Definition 4.2 (DABE with Bounded Delegations)**.** A DABE scheme $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Delegate}, \mathsf{Dec})$ is said to be a DABE scheme with bounded delegations for policy class $\mathcal{F}_s = \{\mathcal{F}_s\}_{s \in \mathbb{N}}$ if the number of times a secret key can be delegated is a-priori bounded. Setup takes this bound 1^d as input. Security definition is also altered accordingly.

5 DABE with Bounded Delegations from Witness Encryption

In this section, we provide the construction of a DABE scheme with bounded delegations. In Section 7, we will show various approaches that will achieve a DABE scheme with unbounded delegations.

Construction 5.1 (DABE)**.** We provide the construction of a DABE scheme with bounded delegations (Definition 4.2) for any family of polynomial-size policies using the following ingredients:

- A statistically-sound NIZK scheme (Definition 3.3) for languages $\mathcal{L}_{\mathsf{KG}}, \mathcal{L}_{\mathsf{Del}}$ (Figure 1, 2 resp.).
- A WE scheme (Definition 3.4) for language $\mathcal{L}_{\mathsf{WE}}$ (Figure 3).

Language $\mathcal{L}_{\mathsf{KG}}$

Instance: $\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f, \mathsf{com}_f^{(0)}$
Witness: $r^{(0)}, r_f^{(0)}, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r_f^{\mathsf{ct}}}$
Relation: Output 1 if and only if $(\phi_1 \wedge \phi_2 \wedge \phi_3) \vee (\phi_4 \wedge \phi_5) = 1$ where,

– $\phi_1 : \mathsf{com}^{(0)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, 0; r^{(0)})$

– $\phi_2 : \mathsf{com}_f^{\mathsf{ct}} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, \mathsf{mfe.ct}_f; r_f^{\mathsf{ct}})$

– $\phi_3 : \mathsf{com}_f^{(0)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, 0; r_f^{(0)})$

– $\phi_4 : \mathsf{com}^{(1)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, \mathsf{mfe.msk}; r^{(1)})$

– $\phi_5 : \mathsf{mfe.ct}_f \stackrel{?}{=} \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f; \widetilde{r_f^{\mathsf{ct}}})$.

Figure 1: Description of $\mathcal{L}_{\mathsf{KG}}$

Language $\mathcal{L}_{\mathsf{Del}}$

Instance: $\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \ldots, f_\ell, \mathsf{com}_{\ell-1}^{(0)}, \mathsf{com}_\ell^{(0)}$ for some $\ell \in [d]$.
Witness: $\mathsf{com}_{\ell-1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell-1}, r_{\ell-1}^{\mathsf{ct}}, r_\ell^{(0)}, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r_\ell^{\mathsf{ct}}}$
Relation: Output 1 if and only if $(\phi_1 \wedge \phi_2 \wedge \phi_3) \vee (\phi_4 \wedge \phi_5) = 1$ where,

– $\phi_1 : \mathsf{com}_{\ell-1}^{\mathsf{ct}} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, \mathsf{mfe.ct}_{\ell-1}; r_{\ell-1}^{\mathsf{ct}})$

– $\phi_2 : \mathsf{com}_\ell^{\mathsf{ct}} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, \mathsf{mfe.ct}_\ell; r_\ell^{\mathsf{ct}})$

– $\phi_3 : \mathsf{com}_\ell^{(0)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, 0; r_\ell^{(0)})$

– $\phi_4 : \mathsf{com}^{(1)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, \mathsf{mfe.msk}; r^{(1)})$

– $\phi_5 : \mathsf{mfe.ct}_\ell \stackrel{?}{=} \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_\ell; \widetilde{r_\ell^{\mathsf{ct}}})$.

Figure 2: Description of $\mathcal{L}_{\mathsf{Del}}$

- A statistically-binding COM scheme (Definition 3.1).
- A Mixed-FE scheme (Definition 3.5) for polynomial-size policies.

$\underline{\mathsf{Setup}(1^\lambda, 1^d, 1^s).}$ Sample $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{Del}})$. Sample $\mathsf{com.crs} \leftarrow \mathsf{COM.Setup}(1^\lambda)$. Compute $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$ and $\mathsf{com}^{(1)} = \mathsf{Com}(0^{\mathsf{poly}(s)}; r^{(1)})$ where $r^{(0)}, r^{(1)} \xleftarrow{\$} \{0,1\}^\lambda$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s)$.

Set $\mathsf{PP} := (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$, and output $(\mathsf{PP}, \mathsf{MSK})$.

$\underline{\mathsf{KGen}(\mathsf{MSK}, f).}$ Parse MSK as $r^{(0)}$. Sample $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f)$, $\mathsf{com}_f^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_f; r_f^{\mathsf{ct}})$, $\mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)})$, where $r_f^{(0)}, r_f^{\mathsf{ct}} \leftarrow \{0,1\}^\lambda$.

Language $\mathcal{L}_{\mathsf{WE}}$

Instance: $\mathsf{nizk.crs_{KG}}, \mathsf{nizk.crs_{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d, x, \mathsf{mfe.sk}_x$
Witness: $f_1, \mathsf{com}_1^{(0)}, \pi_1, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}}$ for some $\ell \in [d]$.
Relation: Output 1 if and only if,

1. $1 \stackrel{?}{=} \mathsf{NIZK.Verify}(\mathsf{nizk.crs_{KG}}, \mathtt{inst}_{\mathsf{KG}}, \pi_1)$
2. $\forall\, i \in [2, \ell], 1 \stackrel{?}{=} \mathsf{NIZK.Verify}(\mathsf{nizk.crs_{Del}}, \mathtt{inst}_{\mathsf{Del},i}, \pi_i)$
3. $\mathsf{com}_\ell^{\mathsf{ct}} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, \mathsf{mfe.ct}_\ell; r_\ell^{\mathsf{ct}})$
4. $0 \stackrel{?}{=} \mathsf{Mix\text{-}FE.Dec}(\mathsf{mfe.ct}_\ell, \mathsf{mfe.sk}_x)$
5. $1 \stackrel{?}{=} f_1(x) \wedge \ldots \wedge f_\ell(x)$

Here, $\mathtt{inst}_{\mathsf{KG}} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \mathsf{com}_1^{(0)})$,
$\mathtt{inst}_{\mathsf{Del},i} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \ldots, f_i, \mathsf{com}_{i-1}^{(0)}, \mathsf{com}_i^{(0)})$.

Figure 3: Description of $\mathcal{L}_{\mathsf{WE}}$

Compute $\pi_f \leftarrow \mathsf{NIZK.Prove}(\mathsf{nizk.crs_{KG}}, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} := (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f, \mathsf{com}_f^{(0)})$, $\mathtt{wit} := (r^{(0)}, r_f^{(0)}, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}}, \bar{0})$[8].

Output $\mathsf{SK}_f := (f, \mathsf{com}_f^{(0)}, \pi_f, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}})$.

$\underline{\mathsf{Enc}(x, \mu)}$**.** Sample $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{mfe.pp}), \mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}, \mu)$ where $\mathtt{inst} := (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$.

Output $\mathsf{CT}_x := (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$.

$\underline{\mathsf{Delegate}(\mathsf{SK}_f, g)}$**.** Parse SK_f as $(f_1, \mathsf{com}_1^{(0)}, \pi_1, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$ for some $\ell \in [d-1]$. Sample $\mathsf{mfe.ct}_{\ell+1} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots f_\ell \wedge g), \mathsf{com}_{\ell+1}^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_{\ell+1}; r_{\ell+1}^{\mathsf{ct}})$, $\mathsf{com}_{\ell+1}^{(0)} = \mathsf{Com}(0; r_{\ell+1}^{(0)})$ where $r_{\ell+1}^{(0)}, r_{\ell+1}^{\mathsf{ct}} \leftarrow \{0,1\}^\lambda$.

Sample $\pi_{\ell+1} \leftarrow \mathsf{NIZK.Prove}(\mathsf{nizk.crs_{Del}}, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} := (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \ldots, f_\ell, g, \mathsf{com}_\ell^{(0)}, \mathsf{com}_{\ell+1}^{(0)})$, $\mathtt{wit} := (\mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}}, r_{\ell+1}^{(0)}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}}, \bar{0})$.

Output $\mathsf{SK}_{f\wedge g} := (f_1, \mathsf{com}_1^{(0)}, \pi_1, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, g, \mathsf{com}_{\ell+1}^{(0)}, \pi_{\ell+1}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}})$.

$\underline{\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}_x)}$**.** Parse SK_f as $(f_1, \mathsf{com}_1^{(0)}, \pi_1, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.sk}_\ell, r_\ell^{\mathsf{ct}})$ for some $\ell \in [d]$ and CT_x as $(x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$. If $f_i(x) = 0$ for any $i \in [\ell]$, abort and output $\perp$.

Otherwise, output $\mu' := \mathsf{WE.Dec}(\mathsf{SK}_f, \mathsf{we.ct})$.

Correctness. The correctness of the construction follows from the correctness of WE, NIZK, $\mathsf{Mix\text{-}FE}$, and COM. Since $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)})$, $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f)$, $\mathsf{com}_f^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_f; r_f^{\mathsf{ct}})$ for some $r^{(0)}, r_f^{\mathsf{ct}}, r_f^{(0)}$, we have $\mathsf{NIZK.Verify}(\mathsf{nizk.crs_{KG}}, \mathtt{inst}_{\mathsf{KG}}, \mathtt{wit}_{\mathsf{KG}}) = 1$ for the corresponding $\mathtt{inst}_{\mathsf{KG}}, \mathtt{wit}_{\mathsf{KG}}$. By correctness of $\mathsf{Mix\text{-}FE}$, $\mathsf{Mix\text{-}FE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}_f) = 0$ with high probability. Hence, if $f(x) = 1$, we have by correctness of WE, $\mu' = \mu$.

[8]By $\bar{0}$, we mean a zero string of sufficient length.

Delegation Correctness. We prove this by induction on $\ell \in [d]$. When $\ell = 1$, its the same as correctness property. Consider $\ell = 2$. We have that $\mathsf{SK}_{f\wedge g} = (f, \mathsf{com}_1^{(0)}, \pi_1, g, \mathsf{com}_2^{(0)}, \pi_2, \mathsf{com}_2^{\mathsf{ct}}, \mathsf{mfe.ct}_2, r_2^{\mathsf{ct}})$. If SK_f was generated using KGen, we have that $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_{\mathsf{KG}}, (\mathsf{com}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f, \mathsf{com}_1^{(0)}), \pi_1) = 1$ (from correctness property). In addition, we also have some r_1^{ct} such that $\mathsf{Com}(\mathsf{mfe.ct}_1; r_1^{\mathsf{ct}}) = \mathsf{com}_1^{\mathsf{ct}}$. Similarly, we have r_2^{ct} and $r_2^{(0)}$ that satisfy the respective properties. Hence, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_{\mathsf{Del}}, \mathtt{inst}_2, \pi_2) = 1$ for $\mathtt{inst}_2 = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f, \mathsf{com}_1^{(0)}, \pi_1, g, \mathsf{com}_2^{(0)})$. By correctness of Mix-FE, with high probability, $\mathsf{Mix\text{-}FE.Dec}(\mathsf{mfe.ct}_2, \mathsf{mfe.sk}_x) = 0$. Hence, by correctness of WE if $f \wedge g(x) = 1$, i.e, $f(x) = 1 \wedge g(x) = 1$, we have that $\mu' = \mu$.

For the induction step, assume that delegation correctness holds for some $k \in [d-1]$ levels of delegation. And assume that $f_1 \wedge \ldots \wedge f_k(x) = 1$ and $g(x) = 1$. Arguing similarly as above, we have r_{k+1}^{ct} and $r_{k+1}^{(0)}$ that satisfy the respective properties for $\mathsf{mfe.ct}_{k+1}$ and $\mathsf{com}_{k+1}^{(0)}$ respectively. Hence, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_{\mathsf{Del}}, \mathtt{inst}_{\mathsf{Del},k+1}, \pi_{k+1}) = 1$. By induction hypothesis, we have some r_k^{ct} such that $\mathsf{mfe.ct}_k \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots f_k \wedge g), \mathsf{com}_k^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_k; r_k^{\mathsf{ct}})$, and $\forall\, i \in [2, k]$, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_{\mathsf{Del}}, \mathtt{inst}_{\mathsf{Del},i}, \pi_i) = 1$. Thus, by WE correctness, $\mu' = \mu$.

6 Security Analysis

In this section, we prove that Construction 5.1 satisfies adaptive security as defined in Definition 4.1. Specifically, we prove the following theorem.

Theorem 6.1. If WE is a WE scheme for $\mathcal{L}_{\mathsf{WE}}$ (Definition 3.4), COM is a COM scheme (Definition 3.1), NIZK is a NIZK scheme for $\mathcal{L}_{\mathsf{KG}}$ and $\mathcal{L}_{\mathsf{Del}}$ (Definition 3.3), Mix-FE is a Mixed-FE scheme for polynomial-size circuits (Definition 3.5), then Construction 5.1 is a DABE scheme with bounded delegations (Definition 4.2) for polynomial-size policies.

Proof. We prove this theorem using the following experiments and lemmas.

$\underline{\mathbf{Expt}_0^{\mathcal{A},b}(1^\lambda)}$. This is almost the honest experiment with $\mathcal{O}$ where either $b = 0/1$. The only change is that we sample $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$ during setup. The output distributions of this experiment and honest experiment are identical.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^{\mathsf{poly}(s)}; r^{(1)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s), \mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Sample $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f), \mathsf{com}_f^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_f; r_f^{\mathsf{ct}}), \mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)}), \pi_f \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f, \mathsf{com}_f^{(0)}), \mathtt{wit} = (r^{(0)}, r_f^{(0)}, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}}, \overline{0})$.

  Set $\mathsf{SK}_f = (f, \mathsf{com}_f^{(0)}, \pi_f, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise, Parse SK_f as $(\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$. Sample $\mathsf{mfe.ct}_{\ell+1} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots \wedge f_\ell \wedge g), \mathsf{com}_{\ell+1}^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_{\ell+1}; r_{\ell+1}^{\mathsf{ct}}), \mathsf{com}_{\ell+1}^{(0)} = \mathsf{Com}(0; r_{\ell+1}^{(0)})$.

Sample $\pi_{\ell+1} \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \ldots, f_\ell, g, \mathsf{com}_\ell^{(0)}, \mathsf{com}_{\ell+1}^{(0)}), \mathtt{wit} = (\mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}}, r_{\ell+1}^{(0)}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}}, \bar{0})$.

Set $\mathsf{SK}_{f\wedge g} = (\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, g, \mathsf{com}_{\ell+1}^{(0)}, \pi_{\ell+1}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_{f\wedge g}, g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, send SK_f to $\mathcal{A}$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

$\underline{\mathbf{Expt}_1^{\mathcal{A},b}(1^\lambda)\mathbf{.}}$ In this experiment, we will simulate all the proofs generated by NIZK for $\mathcal{L}_{\mathsf{Del}}$ using $\underline{\mathsf{NIZK.Sim}}$.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(0^{\mathsf{poly}(s)}; r^{(1)}), \mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{KG}}), \mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s), \mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Sample $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f), \mathsf{com}_f^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_f; r_f^{\mathsf{ct}}), \mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)}), \pi_f \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f, \mathsf{com}_f^{(0)}), \mathtt{wit} = (r^{(0)}, r_f^{(0)}, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}}, \bar{0})$.

  Set $\mathsf{SK}_f = (f, \mathsf{com}_f^{(0)}, \pi_f, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.
- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise,

  Parse SK_f as $(\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$. Sample $\mathsf{mfe.ct}_{\ell+1} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots \wedge f_\ell \wedge g), \mathsf{com}_{\ell+1}^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_{\ell+1}; r_{\ell+1}^{\mathsf{ct}}), \mathsf{com}_{\ell+1}^{(0)} = \mathsf{Com}(0; r_{\ell+1}^{(0)})$.

  Sample $\pi_{f\wedge g} \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \ldots, f_\ell, g, \mathsf{com}_\ell^{(0)}, \mathsf{com}_{\ell+1}^{(0)})$.

  Set $\mathsf{SK}_{f\wedge g} = (\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, g, \mathsf{com}_{\ell+1}^{(0)}, \pi_{\ell+1}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_{f\wedge g}, g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, send SK_f to $\mathcal{A}$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

$\text{Expt}_2^{\mathcal{A},b}(1^\lambda)$. In this experiment, we will simulate all the proofs generated by NIZK for $\mathcal{L}_{\mathsf{KG}}$ using $\mathsf{NIZK.Sim}$.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^{\mathsf{poly}(s)}; r^{(1)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s), \mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Sample $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f), \mathsf{com}_f^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_f; r_f^{\mathsf{ct}}), \mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)})$, $\pi_f \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f, \mathsf{com}_f^{(0)})$.

  Set $\mathsf{SK}_f = (f, \mathsf{com}_f^{(0)}, \pi_f, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise,

  Parse SK_f as $(\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$. Sample $\mathsf{mfe.ct}_{\ell+1} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots \wedge f_\ell \wedge g), \mathsf{com}_{\ell+1}^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_{\ell+1}; r_{\ell+1}^{\mathsf{ct}}), \mathsf{com}_{\ell+1}^{(0)} = \mathsf{Com}(0; r_{\ell+1}^{(0)})$.

  Sample $\pi_{f \wedge g} \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \ldots, f_\ell, g, \mathsf{com}_\ell^{(0)}, \mathsf{com}_{\ell+1}^{(0)})$.

  Set $\mathsf{SK}_{f \wedge g} = (\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, g, \mathsf{com}_{\ell+1}^{(0)}, \pi_{\ell+1}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_{f \wedge g}, g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, send SK_f to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

$\text{Expt}_3^{\mathcal{A},b}(1^\lambda)$. In this experiment, as we do not depend on the chain of proofs anymore for delegation, we will only perform KGen and/or $\mathsf{Delegate}$ when a $\mathsf{Corrupt}$ query is made.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^{\mathsf{poly}(s)}; r^{(1)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s), \mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Send h to $\mathcal{A}$. Store $(f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ asks sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise, send h to $\mathcal{A}$. Store $(g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(f, \mathsf{h}', *)$, send $\perp$. Otherwise, let the chain from root-to-leaf for f be $f_1 \wedge \ldots \wedge f_\ell$ for some $\ell \in [d]$.

  Sample $\mathsf{mfe.ct}_\ell \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots \wedge f_\ell), \mathsf{com}^{\mathsf{ct}}_\ell = \mathsf{Com}(\mathsf{mfe.ct}_\ell; r^{\mathsf{ct}}_\ell)$.

  Sample $\mathsf{com}^{(0)}_f = \mathsf{Com}(0; r^{(0)}_f), \pi_f \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst}_{\mathsf{KG}})$.

  For each $i \in [2, \ell]$, sample $\mathsf{com}^{(0)}_i = \mathsf{Com}(0; r^{(0)}_i)$ and $\pi_i \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst}_{\mathsf{Del},i})$.

  Send $\mathsf{SK}_f = (\ldots, f_\ell, \mathsf{com}^{(0)}_\ell, \pi_\ell, \mathsf{com}^{\mathsf{ct}}_\ell, \mathsf{mfe.ct}_\ell, r^{\mathsf{ct}}_\ell)$ to $\mathcal{A}$. For each of the root-to-leaf functions, update information accordingly. The general case can be done similarly as explained in the proof of Lemma 6.4.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

$\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$. In this experiment, we generate $\mathsf{mfe.ct}_f$ using $\mathsf{Mix\text{-}FE.skEnc}$.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^{\mathsf{poly}(s)}; r^{(1)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s), \mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Send h to $\mathcal{A}$. Store $(f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.
- **Delegate.** $\mathcal{A}$ asks sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise, send h to $\mathcal{A}$. Store $(g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(f, \mathsf{h}', *)$, send $\perp$. Otherwise, let the chain from root-to-leaf for f be $f_1 \wedge \ldots \wedge f_\ell$ for some $\ell \in [d]$.

  Sample $\mathsf{mfe.ct}_\ell \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_\ell; \widetilde{r^{\mathsf{ct}}_\ell})$, $\mathsf{com}^{\mathsf{ct}}_\ell \leftarrow \mathsf{Com}(\mathsf{mfe.ct}_\ell)$

  Sample $\mathsf{com}^{(0)}_f = \mathsf{Com}(0; r^{(0)}_f), \pi_f \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst}_{\mathsf{KG}})$.

  For each $i \in [2, \ell]$, sample $\mathsf{com}^{(0)}_i = \mathsf{Com}(0; r^{(0)}_i)$ and $\pi_i \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst}_{\mathsf{Del},i})$.

  Send $\mathsf{SK}_f = (\ldots, f_\ell, \mathsf{com}^{(0)}_\ell, \pi_\ell, \mathsf{com}^{\mathsf{ct}}_\ell, \mathsf{mfe.ct}_\ell, r^{\mathsf{ct}}_\ell)$ to $\mathcal{A}$. For each of the root-to-leaf functions, update information accordingly.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

$\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$. In this experiment, we will commit $\mathsf{mfe.msk}$ using $\mathsf{com}^{(1)}$ rather than an all zero string.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{nizk.crs_{KG}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs_{Del}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s), \mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$, $\mathsf{com}^{(1)} = \mathsf{Com}(\mathsf{mfe.msk}; r^{(1)})$. Set $\mathsf{PP} = (\mathsf{nizk.crs_{KG}}, \mathsf{nizk.crs_{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- Perform the remaining steps same as $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$.

$\mathbf{Expt}_6^{\mathcal{A},b}(1^\lambda)$. In this experiment, we generate NIZK proofs honestly for $\mathcal{L}_{\mathsf{KG}}$ but use the "trapdoor" witnesses.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{nizk.crs_{KG}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs_{Del}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s), \mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$, $\mathsf{com}^{(1)} = \mathsf{Com}(\mathsf{mfe.msk}; r^{(1)})$. Set $\mathsf{PP} = (\mathsf{nizk.crs_{KG}}, \mathsf{nizk.crs_{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Send h to $\mathcal{A}$. Store $(f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ asks sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise, send h to $\mathcal{A}$. Store $(g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(f, \mathsf{h}', *)$, send $\perp$. Otherwise, let the chain from root-to-leaf for f be $f_1 \wedge \ldots \wedge f_\ell$ for some $\ell \in [d]$.

  Sample $\mathsf{mfe.ct}_\ell \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_\ell; \widetilde{r_\ell^{\mathsf{ct}}})$, $\mathsf{com}_\ell^{\mathsf{ct}} \leftarrow \mathsf{Com}(\mathsf{mfe.ct}_\ell)$

  Sample $\mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)})$, $\pi_f \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}_{\mathsf{KG}}, \mathtt{wit}_{\mathsf{KG}})$ where $\mathtt{wit}_{\mathsf{KG}} = (\overline{0}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r_f^{\mathsf{ct}}})$.

  For each $i \in [2, \ell]$, sample $\mathsf{com}_i^{(0)} = \mathsf{Com}(0; r_i^{(0)})$ and $\pi_i \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst}_{\mathsf{Del},i})$.

  Send $\mathsf{SK}_f = (\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$ to $\mathcal{A}$. For each of the root-to-leaf functions, update information accordingly.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

$\underline{\textbf{Expt}_7^{\mathcal{A},b}(1^\lambda).}$ In this experiment, we generate NIZK proofs honestly for $\mathcal{L}_{\mathsf{Del}}$ but use the "trapdoor" witnesses.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s)$, $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$, $\mathsf{com}^{(1)} = \mathsf{Com}(\mathsf{mfe.msk}; r^{(1)})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Send h to $\mathcal{A}$. Store $(f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ asks sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise, send h to $\mathcal{A}$. Store $(g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(f, \mathsf{h}', *)$, send $\perp$. Otherwise, let the chain from root-to-leaf for f be $f_1 \wedge \ldots \wedge f_\ell$ for some $\ell \in [d]$.

  Sample $\mathsf{mfe.ct}_\ell \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_\ell; \widetilde{r_\ell^{\mathsf{ct}}})$, $\mathsf{com}_\ell^{\mathsf{ct}} \leftarrow \mathsf{Com}(\mathsf{mfe.ct}_\ell)$

  Sample $\mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)})$, $\pi_f \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}_{\mathsf{KG}}, \mathtt{wit}_{\mathsf{KG}})$ where $\mathtt{wit}_{\mathsf{KG}} = (\bar{0}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r_f^{\mathsf{ct}}})$.

  For each $i \in [2, \ell]$, sample $\mathsf{com}_i^{(0)} = \mathsf{Com}(0; r_i^{(0)})$ and $\pi_i \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}_{\mathsf{Del},i}, \mathtt{wit}_{\mathsf{Del},i})$ where $\mathtt{wit}_{\mathsf{Del},i} = (\bar{0}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r_f^{\mathsf{ct}}})$.

  Send $\mathsf{SK}_f = (\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$ to $\mathcal{A}$. For each of the root-to-leaf functions, update information accordingly.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

$\underline{\textbf{Expt}_8^{\mathcal{A},b}(1^\lambda).}$ In this hybrid, we will set $\mathsf{com}_f^{(0)}$ for every queried function to be commitment of 1.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, $\mathsf{com}^{(0)} = \mathsf{Com}(1; r^{(0)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s)$, $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$, $\mathsf{com}^{(1)} = \mathsf{Com}(\mathsf{mfe.msk}; r^{(1)})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Send h to $\mathcal{A}$. Store $(f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ asks sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise, send h to $\mathcal{A}$. Store $(g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(f, \mathsf{h}', *)$, send $\perp$. Otherwise, let the chain from root-to-leaf for f be $f_1 \wedge \ldots \wedge f_\ell$ for some $\ell \in [d]$.

  Sample $\mathsf{mfe.ct}_\ell \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_\ell; \widetilde{r^{\mathsf{ct}}_\ell})$, $\mathsf{com}^{\mathsf{ct}}_\ell \leftarrow \mathsf{Com}(\mathsf{mfe.ct}_\ell)$

  Sample $\mathsf{com}^{(0)}_f = \mathsf{Com}(1; r^{(0)}_f)$, $\pi_f \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}_{\mathsf{KG}}, \mathtt{wit}_{\mathsf{KG}})$ where $\mathtt{wit}_{\mathsf{KG}} = (\overline{0}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r^{\mathsf{ct}}_f})$.

  For each $i \in [2, \ell]$, sample $\mathsf{com}^{(0)}_i = \mathsf{Com}(1; r^{(0)}_i)$ and $\pi_i \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}_{\mathsf{Del},i}, \mathtt{wit}_{\mathsf{Del},i})$ where $\mathtt{wit}_{\mathsf{Del},i} = (\overline{0}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r^{\mathsf{ct}}_f})$.

  Send $\mathsf{SK}_f = (\ldots, f_\ell, \mathsf{com}^{(0)}_\ell, \pi_\ell, \mathsf{com}^{\mathsf{ct}}_\ell, \mathsf{mfe.ct}_\ell, r^{\mathsf{ct}}_\ell)$ to $\mathcal{A}$. For each of the root-to-leaf functions, update information accordingly.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

$\mathbf{Expt}^{\mathcal{A},b}_9(1^\lambda)$. In this experiment, we encrypt all zero string using WE.

- **Setup, Store, Delegate, Corrupt.** Same as $\mathbf{Expt}^{\mathcal{A},b}_8(1^\lambda)$.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, 0^{|\mu_b|})$.

  Send $(x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

Lemma 6.2. **$\mathbf{Expt}^{\mathcal{A},b}_0(1^\lambda)$, $\mathbf{Expt}^{\mathcal{A},b}_1(1^\lambda)$ are computationally indistinguishable assuming the computational zero-knowledge property of NIZK for $\mathcal{L}_{\mathsf{Del}}$.**

Proof. Assume that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Expt}^{\mathcal{A},b}_0(1^\lambda)$ and $\mathbf{Expt}^{\mathcal{A},b}_1(1^\lambda)$ with non-negligible probability $\epsilon(\lambda)$, i.e,

$$\left|\Pr\left[1 \leftarrow \mathbf{Expt}^{\mathcal{A},b}_0(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathbf{Expt}^{\mathcal{A},b}_1(1^\lambda)\right]\right| > \epsilon(\lambda)$$

We will construct a reduction adversary $\mathcal{B}$ that can break the computational zero-knowledge property of NIZK for $\mathcal{L}_{\mathsf{Del}}$ with non-negligible probability. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^{\mathsf{poly}(s)}; r^{(1)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathcal{O}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s)$, $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Sample $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f), \mathsf{com}_f^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_f; r_f^{\mathsf{ct}}), \mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)}), \pi_f \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f, \mathsf{com}_f^{(0)}), \mathtt{wit} = (r^{(0)}, r_f^{(0)}, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}}, \overline{0})$.

  Set $\mathsf{SK}_f = (f, \mathsf{com}_f^{(0)}, \pi_f, \mathsf{com}_f^{\mathsf{ct}}, \mathsf{mfe.ct}_f, r_f^{\mathsf{ct}})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise,

  Parse SK_f as $(\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$. Sample $\mathsf{mfe.ct}_{\ell+1} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots \wedge f_\ell \wedge g), \mathsf{com}_{\ell+1}^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_{\ell+1}; r_{\ell+1}^{\mathsf{ct}}), \mathsf{com}_{\ell+1}^{(0)} = \mathsf{Com}(0; r_{\ell+1}^{(0)})$.

  Sample $\pi_{f \wedge g} \leftarrow \mathcal{O}(\mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \ldots, f_\ell, g, \mathsf{com}_\ell^{(0)}, \mathsf{com}_{\ell+1}^{(0)})$.

  Set $\mathsf{SK}_{f \wedge g} = (\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, g, \mathsf{com}_{\ell+1}^{(0)}, \pi_{\ell+1}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_{f \wedge g}, g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, send SK_f to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

As we can see, the running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$ and λ. If $\mathcal{O}$ uses $(\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove})$, $\mathcal{B}$ behaves like $\mathbf{Expt}_1^{\mathcal{A},b}(1^\lambda)$ and if $\mathcal{O}$ uses $\mathsf{NIZK.Sim}$, $\mathcal{B}$ behaves like $\mathbf{Expt}_2^{\mathcal{A},b}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the computational zero-knowledge property of NIZK that can break its security with probability $\epsilon(\lambda)$. Thus $\mathbf{Expt}_1^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_2^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable. □

Lemma 6.3. $\mathbf{Expt}_1^{\mathcal{A},b}(1^\lambda)$, $\mathbf{Expt}_2^{\mathcal{A},b}(1^\lambda)$ *are computationally indistinguishable assuming the computational zero-knowledge property of* NIZK *for* $\mathcal{L}_{\mathsf{KG}}$.

Proof. The proof of this lemma is similar to proof of Lemma 6.2. □

Lemma 6.4. $\mathbf{Expt}_2^{\mathcal{A},b}(1^\lambda)$ *and* $\mathbf{Expt}_3^{\mathcal{A},b}(1^\lambda)$ *are identically distributed.*

Proof. Note that all that's changed is we are delaying KGen and $\mathsf{Delegate}$ procedures to $\mathsf{Corrupt}$ query. Rest all is done similarly to $\mathsf{Store}, \mathsf{Delegate}$ queries. There are 3 cases to consider —

- *None of the keys for $f_1, \ldots, f_\ell$ are generated.* In this case, we start from the root node using Store and keep on running $\mathsf{Delegate}$ procedure to get the final key.

- *There exists $1 < \ell' < \ell$ such that key for $f_1 \wedge \ldots \wedge f_{\ell'}$ is generated.* In this case, keep on delegating like in $\mathsf{Delegate}$ procedure to generate final key but using $\mathsf{NIZK.Sim}$ for $\mathcal{L}_{\mathsf{Del}}$.

- *$\mathcal{A}$ requested key for $f_1 \wedge \ldots \wedge f_{\ell'}$ when we have a key for $f_1 \wedge \ldots \wedge f_\ell$ with $\ell > \ell' \geq 1$.* In this case, we take a portion of key for $f_1 \wedge \ldots \wedge f_\ell$ and generate $\mathsf{mfe.ct}_{\ell'}$ and commit it. This forms a key for the required policy.

Hence, these two experiments are identical. $\square$

Lemma 6.5. $\mathbf{Expt}_3^{\mathcal{A},b}(1^\lambda)$, $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable assuming the mode indistinguishability property of $\mathsf{Mix\text{-}FE}$.

Proof. Assume that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$ with non-negligible probability $\epsilon(\lambda)$, i.e,

$$\left|\Pr\left[1 \leftarrow \mathbf{Expt}_3^{\mathcal{A},b}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)\right]\right| > \epsilon(\lambda)$$

We will construct a reduction adversary $\mathcal{B}$ that can break the mode indistinguishability property of $\mathsf{Mix\text{-}FE}$ with non-negligible probability. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^{\mathsf{poly}(s)}; r^{(1)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathcal{O}(1^\lambda, 1^s)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Send h to $\mathcal{A}$. Store $(f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ asks sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise, send h to $\mathcal{A}$. Store $(g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(f, \mathsf{h}', *)$, send $\perp$. Otherwise, let the chain from root-to-leaf for f be $f_1 \wedge \ldots \wedge f_\ell$ for some $\ell \in [d]$.

  Sample $\mathsf{mfe.ct}_\ell \leftarrow \mathcal{O}(f_1 \wedge \ldots \wedge f_\ell)$, $\mathsf{com}_\ell^{\mathsf{ct}} \leftarrow \mathsf{Com}(\mathsf{mfe.ct}_\ell)$

  Sample $\mathsf{com}_f^{(0)} = \mathsf{Com}(0; r_f^{(0)}), \pi_f \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst}_{\mathsf{KG}})$.

  For each $i \in [2, \ell]$, sample $\mathsf{com}_i^{(0)} = \mathsf{Com}(0; r_i^{(0)})$ and $\pi_i \leftarrow \mathsf{NIZK.Sim}(\mathtt{inst}_{\mathsf{Del},i})$.

  Send $\mathsf{SK}_f = (\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$ to $\mathcal{A}$. For each of the root-to-leaf functions, update information accordingly.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

As we can see, the running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$ and λ. If $\mathcal{O}$ uses $\mathsf{Mix\text{-}FE.pkEnc}$, $\mathcal{B}$ behaves like $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ and if $\mathcal{O}$ uses $\mathsf{Mix\text{-}FE.skEnc}$, $\mathcal{B}$ behaves like $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the mode indistinguishability property of $\mathsf{Mix\text{-}FE}$ that can break its security with probability $\epsilon(\lambda)$. Thus $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable. □

Lemma 6.6. $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$, $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable assuming the computational hiding property of COM.

Proof. Assume that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$ with non-negligible probability $\epsilon(\lambda)$, i.e,

$$\left|\Pr\left[1 \leftarrow \mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)\right]\right| > \epsilon(\lambda)$$

We will construct a reduction adversary $\mathcal{B}$ that can break the computational hiding property of COM with non-negligible probability. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Receive $\mathsf{com.crs} \leftarrow \mathcal{O}$, sample $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Sim}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s)$, $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$. Compute $\mathsf{com}^{(1)} = \mathcal{O}(0^{\mathsf{poly}(s)}, \mathsf{mfe.msk})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1.

  Send PP to $\mathcal{A}$
- Perform the remaining steps same as $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$.

As we can see, the running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$ and λ. If $\mathcal{O}$ returns a commitment of $0^{\mathsf{poly}(s)}$, $\mathcal{B}$ behaves like $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ and if $\mathcal{O}$ returns a commitment of $\mathsf{mfe.msk}$, $\mathcal{B}$ behaves like $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the computational hiding property of COM that can break its security with probability $\epsilon(\lambda)$. Thus $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable. □

Lemma 6.7. $\mathbf{Expt}_5^{\mathcal{A},b}(1^\lambda)$, $\mathbf{Expt}_6^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable assuming the computational zero knowledge property of NIZK for $\mathcal{L}_{\mathsf{KG}}$.

Proof. Note that the trapdoor witnesses are also valid witness for $\mathcal{L}_{\mathsf{KG}}$. Hence, the proof of this lemma is similar to proof of Lemma 6.2. □

Lemma 6.8. $\mathbf{Expt}_6^{\mathcal{A},b}(1^\lambda)$, $\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable assuming the computational zero knowledge property of NIZK for $\mathcal{L}_{\mathsf{Del}}$.

Proof. Note that the trapdoor witnesses are also valid witness for $\mathcal{L}_{\mathsf{Del}}$. Hence, the proof of this lemma is similar to proof of Lemma 6.2. □

Lemma 6.9. $\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)$, $\mathbf{Expt}_8^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable assuming the computational hiding property of COM.

Proof. Assume that there exists an adversary $\mathcal{A}$ that can distinguish between $\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_8^{\mathcal{A},b}(1^\lambda)$ with non-negligible probability $\epsilon(\lambda)$, i.e,

$$\left|\Pr\left[1 \leftarrow \mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathbf{Expt}_8^{\mathcal{A},b}(1^\lambda)\right]\right| > \epsilon(\lambda)$$

We will construct a reduction adversary $\mathcal{B}$ that can break the computational hiding property of COM with non-negligible probability. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^s$. Receive $\mathsf{com.crs} \leftarrow \mathcal{O}$, $\mathsf{com}^{(0)} = \mathcal{O}(0,1)$, $\mathsf{nizk.crs}_{\mathsf{KG}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{KG}})$, $\mathsf{nizk.crs}_{\mathsf{Del}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{Del}})$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}FE.Setup}(1^\lambda, 1^s)$, $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}FE.Gen}(\mathsf{PP})$, $\mathsf{com}^{(1)} = \mathsf{Com}(\mathsf{mfe.msk}; r^{(1)})$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, d)$. Initiate h to 1.

  Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Send h to $\mathcal{A}$. Store $(f, \mathsf{h}, \perp)$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ asks sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$ to $\mathcal{A}$. Otherwise, send h to $\mathcal{A}$. Store $(g, \mathsf{h}, \mathsf{h}')$ and $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(f, \mathsf{h}', *)$, send $\perp$. Otherwise, let the chain from root-to-leaf for f be $f_1 \wedge \ldots \wedge f_\ell$ for some $\ell \in [d]$.

  Sample $\mathsf{mfe.ct}_\ell \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_\ell; \widetilde{r_\ell^{\mathsf{ct}}})$, $\mathsf{com}_\ell^{\mathsf{ct}} \leftarrow \mathsf{Com}(\mathsf{mfe.ct}_\ell)$

  Sample $\mathsf{com}_f^{(0)} \leftarrow \mathcal{O}(0,1)$, $\pi_f \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}_{\mathsf{KG}}, \mathtt{wit}_{\mathsf{KG}})$ where $\mathtt{wit}_{\mathsf{KG}} = (\overline{0}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r_f^{\mathsf{ct}}})$.

  For each $i \in [2, \ell]$, sample $\mathsf{com}_i^{(0)} \leftarrow \mathcal{O}(0,1)$ and $\pi_i \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst}_{\mathsf{Del},i}, \mathtt{wit}_{\mathsf{Del},i})$ where $\mathtt{wit}_{\mathsf{Del},i} = (\overline{0}, r^{(1)}, \mathsf{mfe.msk}, \widetilde{r_f^{\mathsf{ct}}})$.

  Send $\mathsf{SK}_f = (\ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$ to $\mathcal{A}$. For each of the root-to-leaf functions, update information accordingly.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, 1^{\mathsf{poly}(s,d)}, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$.

  Send $\mathsf{CT} = (x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$ to $\mathcal{A}$.

Store, Delegate, Corrupt. These are handled similarly to pre-challenge phase.

- **Guess.** Output whatever $\mathcal{A}$ outputs.

As we can see, the running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$ and λ. If $\mathcal{O}$ returns a commitment of 0, $\mathcal{B}$ behaves like $\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)$ and if $\mathcal{O}$ returns a commitment of 1, $\mathcal{B}$ behaves like $\mathbf{Expt}_8^{\mathcal{A},b}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the computational hiding property of COM that can break its security with probability $\epsilon(\lambda)$. Thus $\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_8^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable. □

Lemma 6.10. $\mathbf{Expt}_8^{\mathcal{A},b}(1^\lambda)$, $\mathbf{Expt}_9^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable assuming the security of WE.

Proof. Note that at this point, $\texttt{inst}_{\mathsf{WE}} = (\mathsf{nizk.crs}_{\mathsf{KG}}, \mathsf{nizk.crs}_{\mathsf{Del}}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, x, \mathsf{mfe.sk}_x, d)$ is unsatisfiable. Assume that there is $\texttt{wit}_{\mathsf{WE}} = (f_1, \mathsf{com}_1^{(0)}, \pi_1, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$ for some $\ell \in [d]$ such that $\mathsf{WE.R}(\texttt{inst}_{\mathsf{WE}}, \texttt{wit}_{\mathsf{WE}}) = 1$ with π_i verifying for every $i \in [\ell]$. Then —

- If $\mathsf{com}_\ell^{(0)}$ is a commitment of 1, then it must be that $\mathsf{mfe.ct}_\ell \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_\ell)$ as $\mathsf{com}^{(1)}$ is a commitment of $\mathsf{mfe.msk}$. Hence, by correctness of Mix-FE, we reach a contradiction.
- If $\ell^* < \ell$ is such that $\mathsf{com}_{\ell^*}^{(0)}$ is a commitment of 1, then it must be that $f_1 \wedge \ldots \wedge f_{\ell^*}(x) = 0$, which is a contradiction.

By statistical binding property of COM and statistical soundness of NIZK, with overwhelming probability, WE instance is unsatisfiable. □

Note that $\mathbf{Expt}_9^{\mathcal{A},b}(1^\lambda)$ is independent of b. Hence, security of Construction 5.1 follows. □

7 Bootstrapping to Unbounded Delegations

In Section 5, we constructed a DABE scheme that can handle a-priori bounded d number of delegations. In this section, we highlight the approaches one can use that can handle unbounded delegations. That is, we can construct DABE for polynomial-size policies satisfying Definition 4.1 from any DABE scheme for polynomial-size policies satisfying Definition 4.2 using the following approaches.

Approach 1: Fully Homomorphic Encryption. The reason we achieve bounded delegation with Construction 5.1 is that we need to bound $|\texttt{inst}|$ and $|\texttt{wit}|$ while creating $\mathsf{nizk.crs}$ for any NIZK (Definition 3.3). However, if we have a statistically sound NIZK that can handle unbounded size $\texttt{inst}$, $\texttt{wit}$, we can readily use this to realize unbounded delegation. Note that it is fine to let running time of $\mathsf{NIZK.Prove}$ grow polynomially in $|\texttt{inst}|, |\texttt{wit}|$, its just that we need $\mathsf{NIZK.Prove}$ to run on unbounded size inputs. [GGI+15] constructs such a statistically sound NIZK scheme using fully homomorphic encryption (which is known from circularly-secure LWE [Gen09, BV11, GSW13]). If we plug-in this NIZK in Construction 5.1, we can achieve DABE that satisfies Definition 4.1 readily.

Approach 2: Create a new NIZK after each delegation. After ℓ many delegations, we know that $\texttt{inst}_{\ell+1}$ for $\mathcal{L}_{\mathsf{Del}}$ grows *additively* in ℓ and functionality index s. Similarly, an upper bound on $|\texttt{wit}_{\ell+1}|$ can be determined using ℓ, s independent of $|\texttt{wit}_\ell|$. Hence, we can sample a new $\mathsf{nizk.crs}$ for these parameters and include it as part of secret key. This way the delegator will use this new $\mathsf{nizk.crs}$ to generate $\pi_{\ell+1}$. We also need to "sign" the new $\mathsf{nizk.crs}$ so that the attacker cannot switch it to delegate further. Moreover, to use WE security, this signature should be statistically secure. Such a signature can be generated using another NIZK scheme[9] where the instance is $\mathsf{com}', \pi_\ell, \mathsf{nizk.crs}$ and witness is r' such that $\mathsf{com}' = \mathsf{Com}(0; r')$. We can repeat this process to handle unbounded delegations. With these modifications, changes in our construction are highlighted in red as follows:

[9] We cannot include $\mathsf{nizk.crs}$ as part of $\texttt{inst}_\ell$ as this would lead to efficiency issues.

Changes in $\mathcal{L}_{\mathsf{WE}}$. We use the witness — $f_1, \mathsf{com}_1^{(0)}, \pi_1, \mathsf{com}'_2, \mathsf{nizk.crs}_2, \sigma_2, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}'_{\ell+1}, \mathsf{nizk.crs}_{\ell+1}, \sigma_{\ell+1}, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}}$ for some $\ell \in [d]$. In the relationship circuit, we check for each $i \in [2, \ell]$ NIZK verifies using $\mathsf{nizk.crs}_i$ and the signatures $\{\sigma_i\}_i$ verify too.

$\underline{\mathsf{Delegate}(\mathsf{SK}_f, g)}$. Parse SK_f as $(f_1, \mathsf{com}_1^{(0)}, \pi_1, \mathsf{com}'_2, \mathsf{nizk.crs}_2, \sigma_2, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{nizk.crs}_{\ell+1}, \sigma_{\ell+1}, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}})$ for some ℓ. Sample $\mathsf{mfe.ct}_{\ell+1} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(f_1 \wedge \ldots f_\ell \wedge g), \mathsf{com}_{\ell+1}^{\mathsf{ct}} = \mathsf{Com}(\mathsf{mfe.ct}_{\ell+1}; r_{\ell+1}^{\mathsf{ct}}), \mathsf{com}_{\ell+1}^{(0)} = \mathsf{Com}(0; r_{\ell+1}^{(0)})$ where $r_{\ell+1}^{(0)}, r_{\ell+1}^{\mathsf{ct}} \leftarrow \{0,1\}^\lambda$.

Sample $\pi_{\ell+1} \leftarrow \mathsf{NIZK.Prove}(\mathsf{nizk.crs}_{\ell+1}, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} := (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.pp}, f_1, \ldots, f_\ell, g, \mathsf{com}_\ell^{(0)}, \mathsf{com}_{\ell+1}^{(0)}), \mathtt{wit} := (\mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.ct}_\ell, r_\ell^{\mathsf{ct}}, r_{\ell+1}^{(0)}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}}, \overline{0})$.

Sample $\mathsf{nizk.crs}_{\ell+2} \leftarrow \mathsf{NIZK.Setup}(1^\lambda, 1^{\mathsf{Del}'})$ where Del' is the new language whose instance and witness length can be efficiently determined.

Sample $\mathsf{com}' = \mathsf{Com}(0; r'), \sigma_{\ell+2} \leftarrow \mathsf{Sign}((\mathsf{com}', \pi_{\ell+1}, \mathsf{nizk.crs}_{\ell+2}), r')$.

Output $\mathsf{SK}_{f \wedge g} := (f_1, \mathsf{com}_1^{(0)}, \pi_1, \mathsf{com}'_2, \mathsf{nizk.crs}_2, \sigma_2, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}'_{\ell+1}, \mathsf{nizk.crs}_{\ell+1}, \sigma_{\ell+1}, g, \mathsf{com}_{\ell+1}^{(0)}, \pi_{\ell+1}, \mathsf{com}', \mathsf{nizk.crs}_{\ell+2}, \sigma_{\ell+2}, \mathsf{com}_{\ell+1}^{\mathsf{ct}}, \mathsf{mfe.ct}_{\ell+1}, r_{\ell+1}^{\mathsf{ct}})$.

$\underline{\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}_x)}$. Parse SK_f as $(f_1, \mathsf{com}_1^{(0)}, \pi_1, \mathsf{com}'_2, \mathsf{nizk.crs}_2, \sigma_2, \ldots, f_\ell, \mathsf{com}_\ell^{(0)}, \pi_\ell, \mathsf{com}'_{\ell+1}, \mathsf{nizk.crs}_{\ell+1}, \sigma_{\ell+1}, \mathsf{com}_\ell^{\mathsf{ct}}, \mathsf{mfe.sk}_\ell, r_\ell^{\mathsf{ct}})$ for some $\ell \in [d]$ and CT_x as $(x, \mathsf{mfe.sk}_x, \mathsf{we.ct})$. If $f_i(x) = 0$ for any $i \in [\ell]$, abort and output $\perp$.

Otherwise, output $\mu' := \mathsf{WE.Dec}(\mathsf{SK}_f, \mathsf{we.ct})$.

The security argument remains almost the same. However, now we need to go down the delegation tree maintained by the challenger level-by-level to simulate all the new NIZK instantiations and do this for each branch of the delegation tree. In addition, we will also simulate $\mathsf{nizk.crs}_\ell$ inside signatures σ.

Approach 3: Use NIWIs. If we use non-interactive witness indistinguishability proofs (NIWI) [BOV07, FS90] instead of NIZKs, we will not face any of the efficiency issues that plagued NIZKs in the previous two approaches. In addition, there will be no crs for NIZKs as well. However, it wouldn't be enough for the languages to have a normal mode and a trapdoor mode. This is because in both modes, we are using $\mathsf{mfe.ct}_f$ to some capacity (cf. Figs. 1 and 2). In order to resolve this, we need to introduce a "simulation" mode which serves the same purpose as NIZK simulation. That is, this mode lets us to create valid proofs without using $\mathsf{mfe.ct}_f$ and thus we can defer Store and $\mathsf{Delegate}$ queries to $\mathsf{Corrupt}$ phase.

We can do so by simply using another commitment com' that in conjunction with $\mathsf{com}^{(0)}$ will determine which mode we are in. For instance, if $\mathsf{com}^{(0)}, \mathsf{com}'$ are both commitments of 0, we will be in the normal mode. If $\mathsf{com}^{(0)}$ is a commitment of 0, com' is a commitment of 1, we will be in the simulation mode where we will not be using any information about $\mathsf{mfe.ct}_f$. If $\mathsf{com}^{(0)}$ is commitment of 1, we will be in the trapdoor mode where we don't care about com'. Each time we generate SK_f or delegate it, we need to sample a new com'. By relying on statistical soundness of NIWIs and statistical binding of COM we can argue adaptive security similarly to NIZK version.

Note that in all approaches, $\mathcal{L}_{\mathsf{WE}}$ still required bound d as we need a bound for $|\mathtt{wit}_{\mathsf{WE}}|$.

References

[ABG$^+$13] Prabhanjan Ananth, Dan Boneh, Sanjam Garg, Amit Sahai, and Mark Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 657–677, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany.

[AC17] Shashank Agrawal and Melissa Chase. Simplifying design and analysis of complex predicate encryption schemes. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 627–656, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland.

[ACG$^+$24] Abtin Afshar, Jiaqi Cheng, Rishab Goyal, Aayush Yadav, and Saikumar Yadugiri. Encrypted RAM delegation: Applications to rate-1 extractable arguments, homomorphic NIZKs, MPC, and more. Cryptology ePrint Archive, Paper 2024/1806, 2024.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Annual Cryptology Conference*, pages 308–326. Springer, 2015.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. *Cryptology ePrint Archive*, 2015.

[AKY24] Shweta Agrawal, Simran Kumari, and Shota Yamada. Attribute based encryption for turing machines from lattices. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 352–386, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[APG$^+$11] Joseph A Akinyele, Matthew W Pagano, Matthew D Green, Christoph U Lehmann, Zachary NJ Peterson, and Aviel D Rubin. Securing electronic medical records using attribute-based encryption on mobile devices. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 75–86, 2011.

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In *Theory of Cryptography Conference*, pages 174–198. Springer, 2019.

[BBS$^+$09] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pages 135–146, 2009.

[BCG$^+$17] Zvika Brakerski, Nishanth Chandran, Vipul Goyal, Aayush Jain, Amit Sahai, and Gil Segev. Hierarchical functional encryption. In *8th Innovations in Theoretical Computer*

Science Conference (ITCS 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[BDS24] Pedro Branco, Nico Döttling, and Akshayaram Srinivasan. Rate-1 statistical non-interactive zero-knowledge. *Cryptology ePrint Archive*, 2024.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*, pages 533–556. Springer, 2014.

[BIOW20] Ohad Barta, Yuval Ishai, Rafail Ostrovsky, and David J. Wu. On succinct arguments and witness encryption from groups. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 776–806, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

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

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BS15] Zvika Brakerski and Gil Segev. Hierarchical functional encryption. *Cryptology ePrint Archive*, 2015.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE symposium on security and privacy (SP'07)*, pages 321–334. IEEE, 2007.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography: 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings 8*, pages 253–273. Springer, 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *FOCS*, 2015.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*, pages 535–554. Springer, 2007.

[CDEN12] Jan Camenisch, Maria Dubovitskaya, Robert R Enderlein, and Gregory Neven. Oblivious transfer with hidden access control from attribute-based encryption. In *Security and Cryptography for Networks: 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings 8*, pages 559–579. Springer, 2012.

[CGW15] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system ABE in prime-order groups via predicate encodings. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 595–624, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[CVW$^+$18] Yilei Chen, Vinod Vaikuntanathan, Brent Waters, Hoeteck Wee, and Daniel Wichs. Traitor-tracing from lwe made simple and attribute-based. In *Theory of Cryptography: 16th International Conference, TCC 2018, Panaji, India, November 11–14, 2018, Proceedings, Part II 16*, pages 341–369. Springer, 2018.

[DG17a] Nico Döttling and Sanjam Garg. From selective IBE to full IBE and selective HIBE. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 372–408, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[DG17b] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 537–569, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland.

[DSY91] Alfredo De Santis and Moti Yung. Cryptographic applications of the non-interactive metaproof and many-prover systems. In *Advances in Cryptology-CRYPTO'90: Proceedings 10*, pages 366–377. Springer, 1991.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd Annual ACM Symposium on Theory of Computing*, pages 416–426, Baltimore, MD, USA, May 14–16, 1990. ACM Press.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[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 *54th Annual Symposium on Foundations of Computer Science*, pages 40–49, Berkeley, CA, USA, October 26–29, 2013. IEEE Computer Society Press.

[GGI$^+$15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *Journal of Cryptology*, 28(4):820–843, 2015.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th*

Annual ACM Symposium on Theory of Computing, pages 467–476, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[GH09] Craig Gentry and Shai Halevi. Hierarchical identity based encryption with polynomially many levels. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 437–456. Springer Berlin Heidelberg, Germany, March 15–17, 2009.

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run Turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 612–621, 2017.

[GKW18] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 660–670, 2018.

[GLW21] Rishab Goyal, Jiahui Liu, and Brent Waters. Adaptive security via deletion in attribute-based encryption: Solutions from search assumptions in bilinear groups. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 311–341, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.

[GM15] Matthew D Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In *2015 IEEE Symposium on Security and Privacy*, pages 305–320. IEEE, 2015.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 89–98, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. Available as Cryptology ePrint Archive Report 2006/309.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566, Queenstown, New Zealand, December 1–5, 2002. Springer Berlin Heidelberg, Germany.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *Advances in Cryptology–CRYPTO 2012: 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 162–179. Springer, 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523. Springer, 2015.

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer Berlin Heidelberg, Germany.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *64th Annual Symposium on Foundations of Computer Science*, pages 415–434, Santa Cruz, CA, USA, November 6–9, 2023. IEEE Computer Society Press.

[HLL24] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. A general framework for lattice-based ABE using evasive inner-product functional encryption. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part II*, volume 14652 of *Lecture Notes in Computer Science*, pages 433–464, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Advances in Cryptology–EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings 27*, pages 146–162. Springer, 2008.

[LLL22] Hanjun Li, Huijia Lin, and Ji Luo. ABE for circuits with constant-size secret keys and adaptive security. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 680–710, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland.

[LOS$^+$10] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*, pages 62–91. Springer, 2010.

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 455–479, Zurich, Switzerland, February 9–11, 2010. Springer Berlin Heidelberg, Germany.

[LW14] Allison B. Lewko and Brent Waters. Why proving HIBE systems secure is difficult. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 58–76, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of cryptology*, 4:151–158, 1991.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 422–439, Taormina, Sicily, Italy, March 19–21, 2012. Springer Berlin Heidelberg, Germany.

[SRGS12] Nuno Santos, Rodrigo Rodrigues, Krishna P Gummadi, and Stefan Saroiu. {Policy-Sealed} data: A new abstraction for building trusted cloud services. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 175–188, 2012.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 463–472, 2010.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*, pages 457–473. Springer, 2005.

[SW08] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II*, volume 5126 of *Lecture Notes in Computer Science*, pages 560–578, Reykjavik, Iceland, July 7–11, 2008. Springer Berlin Heidelberg, Germany.

[TBEM08] Patrick Traynor, Kevin RB Butler, William Enck, and Patrick D McDaniel. Realizing massive-scale conditional access systems through attribute-based cryptosystems. In *NDSS*, 2008.

[Tsa19] Rotem Tsabary. Fully secure attribute-based encryption for t-CNF from LWE. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 62–85, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In *Annual International Cryptology Conference*, pages 535–559. Springer, 2022.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-io from evasive lwe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 195–221. Springer, 2022.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*, pages 114–127. Springer, 2005.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636, Santa Barbara, CA, USA, August 16–20, 2009. Springer Berlin Heidelberg, Germany.

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Conference on Theory and Practice of Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70, Taormina, Italy, March 6–9, 2011. Springer Berlin Heidelberg, Germany.

[Wat15] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 678–697, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany.

[Wee22] Hoeteck Wee. Optimal broadcast encryption and cp-abe from evasive lattice assumptions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 217–241. Springer, 2022.

[WW24] Brent Waters and Daniel Wichs. Adaptively secure attribute-based encryption from witness encryption. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part III*, volume 15366 of *Lecture Notes in Computer Science*, pages 65–90, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017*, pages 600–611, 2017.