Delegatable ABE with $O(1)$ Delegations 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. We design a 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.

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 [GPSW06, 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 generate new keys for the TAs and other instructional staff, where the access policy

*An earlier version of this work contained a bug. In the earlier version, we incorrectly claimed an adaptively secure delegatable attribute-based encryption with unbounded delegations.

†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.

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, GPSW06, 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 $\mathsf{sk}_{f_1 \wedge f_2}$).

Since a real-world adversary should be allowed to corrupt users as above, proving full security of DABE is significantly more challenging than ABE without delegations. Briefly, the reason is that, in (vanilla) ABE, an attacker never initializes user keys but just directly corrupts them. Therefore, the reduction algorithm is never required to be able to sample even a portion of a secret key that can distinguish the challenge ciphertext. Specifically, whenever a challenger has to create a secret key for some predicate f, *in the pre-challenge query phase*, then it has the guarantee that $f(x^*) = 0$ (i.e., f will not satisfy the challenge attribute x^*) even though the challenger does not know x^* yet. However, this guarantee is **not** available for DABE. Let us elaborate.

An adaptive attacker in DABE can ask the challenger to initialize a secret key sk_f for some predicate f, but not corrupt it. It then can ask the challenger to use sk_f to generate one or more delegated keys $\mathsf{sk}'_1, \ldots$. Interestingly, an adaptive attacker can ask to corrupt these delegated keys $\mathsf{sk}'_1, \ldots$ as long as it ensures that the challenge attribute x^* (that it picks later) does not satisfy the predicates corresponding to these delegated keys. The technical challenge this raises is that if we want to design a reduction algorithm, then as a reduction algorithm we have no clue as to whether $f(x^*)$ will be 0 or 1. Because the adaptive attacker selects x^* after it receives the delegated keys, and it can choose to make $f(x^*)$ be 0 or 1. This causes significant uncertainty for the challenger/reduction algorithm. Concretely, each delegated key $\mathsf{sk}'_1, \ldots$ is sampled using the same secret key sk_f, thus all these secret keys contain some information about sk_f. Now the reduction algorithm is tasked with a job where it must answer these delegated secret key queries in such a way that if $f(x^*) = 0$, then it must be able to give the attacker a secret key for sk_f such that delegated secret keys look like they were generated from the same sk_f; while if $f(x^*) = 1$, then the delegated keys must lose enough information about sk_f such that a challenger could never have

a created a valid key for sk_f.

The above 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] already 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. We provide a new construction for DABE for all poly-sized predicates, that can support any constant number of delegations, from witness encryption [GGSW13]. We prove selective security of our DABE scheme by relying on statistically-sound NIZKs and perfectly-binding commitment schemes. Moreover, we prove full security of our DABE scheme by additionally relying on (bounded-collusion) fully-secure hierarchical functional encryption.

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 [LOS+10, GVW13, BGG+14, Tsa19, Wee22, LLL22, HLL23, AKY24, HLL24] constructed ABE with varying levels of security and functionality from several cryptographic assumptions.

Delegatable ABE is a "hierarchical" variant of ABE, and there have been a few designs [GVW13, BGG+14] of DABE with selective security. A more expressive variant of DABE is referred to as delegatable FE [BCG+17, BS15], and is known to be nearly equivalent to vanilla FE [BW07, KSW08, BSW11]. A less expressive variant of DABE is referred to as hierarchical IBE [HL02, GS02], and is known to be equivalent to IBE [DG17b, DG17a] in the selective security model.

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.

Concurrent work. In a recent concurrent work, Goyal-Koppula-Rajasree [GKR25] also studied the problem of adaptive security of delegatable attribute-based encryption, but their focus was on the less expressive class of prefix predicates which is captured in the form of hierarchical IBE. They proved existence of adaptively-secure HIBE systems under the assumption of selectively-secure IBE.

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, GPSW06, 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 generate secret keys for policy f to create secret key SK_f using MSK (this can be thought of as *private* delegation).

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 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[1] 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 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 perfectly-binding commitment schemes (COM) and perfectly sound non-interactive zero-knowledge (NIZK) proof

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

system for the following language.

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

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

A bit more formally, in the construction, we generate keys for policies f using the master secret key $r^{(0)}$ and the instance (PP, f), i.e, $\sigma_f = \pi_f$. This is the normal mode of operation in the signature scheme. We switch to trapdoor mode by setting $\mathsf{com}^{(1)} = \mathsf{Com}(x^*)$ and $\mathsf{com}^{(0)} = \mathsf{Com}(1)$. To argue selective security, the flow of hybrids look as follows — (1) set $\mathsf{com}^{(1)}$ to be a commitment of x^* (2) during key generation, sample π_f using the witness $(r^{(1)}, x^*)$ (3) set $\mathsf{com}^{(0)}$ to be a commitment of 1. Thus, in the final hybrid, any valid witness for the WE language will be in the trapdoor mode. If NIZK and Com are perfectly sound and binding, instance used in WE encryption is no longer satisfiable because if π_g verifies, it must mean that $g(x^*) = 0$ and the WE language requires $g(x^*) = 1$ (a contradiction).

Now, we will discuss our approach in making GGSW scheme delegatable and selectively secure. At a high level, in this approach we compose proofs on top of π_f to construct a delegation chain. 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 $\mathsf{Delegate}$ 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[2]. Let us elaborate.

In the $\mathsf{Delegate}$ algorithm, we will use (f, π_f) as witness to a NIZK language that checks if $\mathsf{NIZK.Verify}((\mathsf{PP}, f), \pi_f) = 1$ in the normal mode and trapdoor mode remains unchanged. 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. Specifically, the WE language is now defined with respect to d-many NIZK languages $\mathcal{L}_1, \ldots, \mathcal{L}_d$. Key generation is performed using $\mathcal{L}_1$ and delegation uses $\mathcal{L}_2, \ldots, \mathcal{L}_d$. $\mathcal{L}_i$ in the normal mode checks if the NIZK proof present in the witness verifies with respect to $\mathcal{L}_{i-1}$ for $i \in \{2, \ldots, d\}$. $\mathcal{L}_i$'s trapdoor mode checks if we have $r^{(1)}$ such that $\mathsf{com}^{(1)}$ is a commitment of x^* and $f_1 \wedge \ldots \wedge f_i(x^*) = 0$ for each $i \in [d]$. WE language when provided with witness $(f_1, \ldots, f_\ell, \pi_\ell)$ for some $\ell \in [d]$, checks if π_ℓ verifies with respect to $\mathcal{L}_\ell$ and $f_1 \wedge \ldots \wedge f_\ell(x^*) \stackrel{?}{=} 1$.

In order to argue *selective* security, the same ideas from GGSW flow naturally. Firstly, we can commit x^* as $\mathsf{com}^{(1)}$ instead of 0^ℓ similar to GGSW. Secondly, we move to a hybrid where we do not generate any keys during Store or $\mathsf{Delegate}$ phase and push generation of proofs to $\mathsf{Corrupt}$ phase by storing the delegation chain for each $\mathsf{Corrupt}$ query. That is, when Store query is made, we simply note down f_1 and do not generate any π_{f_1}. Whenever the attacker asks for delegation on top of f_1, we note down the sequence of functions that f_1 is delegated upon, $f_2, \ldots, f_\ell$. When $\mathsf{Corrupt}$ query is made for f_ℓ, then we generate π_{f_1} for policy f_1 under language $\mathcal{L}_1$, π_{f_2} for policy f_2 under language $\mathcal{L}_2$, and so on. Next, in a series of hybrids, we simulate all the NIZK proofs generated for $\mathcal{L}_i$ for $i = 1, \ldots d$.

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

At this point, we can observe two things – (1) we are no longer using $r^{(0)}$ anywhere in the game, (2) we are no longer π_{f_i} to generate $\pi_{f_{i+1}}$ for any $i \in [d-1]$. Thus, whenever $\mathsf{Corrupt}$ query is made for the policy $f_1 \wedge \ldots \wedge f_\ell$ for some $\ell \in [d]$, we simply simulate a NIZK proof π_{f_ℓ} for the language $\mathcal{L}_\ell$ and send $(f_1, \ldots, f_\ell, \pi_{f_\ell})$ to the attacker. Finally, we can generate all NIZK proofs in the trapdoor mode and set $\mathsf{com}^{(0)}$ to be a commitment of 1. If NIZK and COM are perfectly sound and binding, we can argue that the instance in WE is not satisfiable similar to GGSW. Full construction and security proof is provided in Section 5.

Note that the above construction can only handle constant depth delegations as we can only compose proofs constant-many times. More specifically, 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 $|\pi|$. Hence, 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.

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[3] [GKW18] that played an important part in executing the dual-systems paradigm. Below, we briefly recall the definition of Mixed-FE and how [WW24] used it to realize adaptively secure ABE from WE 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[4] 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 $\mathsf{pkEnc}(\mathsf{pp})$. In essence, a Mixed-FE scheme consists of the following 5 algorithms: $\mathsf{Setup}, \mathsf{KGen}, \mathsf{pkEnc}, \mathsf{skEnc}, \mathsf{Dec}$. $(\mathsf{skEnc}, \mathsf{KGen})$ 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[5]. This attacker cannot distinguish between the modes in which ciphertexts are generated for any policy as long as $\forall i, f_i(x) = 0$. That is,

$$\left(\begin{array}{c}\{\mathsf{mfe.ct} : \mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(\mathsf{pp})\}_i \\ \mathsf{mfe.sk}_x\end{array}\right) \approx_c \left(\begin{array}{c}\{\mathsf{mfe.ct} : \mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_i)\}_i \\ \mathsf{mfe.sk}_x\end{array}\right)$$

where $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$.

Adaptive ABE via Mixed-FE. We briefly provide an overview of the argument used in [WW24] to argue adaptive security of GGSW. At a high level, in this approach, we will use, (Mix-FE.KGen,

[3] 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.

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

[5] 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.

$\mathsf{Mix\text{-}FE.pkEnc}$) in the functional system and ($\mathsf{Mix\text{-}FE.KGen}, \mathsf{Mix\text{-}FE.skEnc}$) in the semi-functional system. The switch to semi-functional system is unnoticeable by security of $\mathsf{Mix\text{-}FE}$. In a bit more detail, in the construction, along with π_f, they give out ciphertexts for $\mathsf{Mix\text{-}FE}$, $\mathsf{mfe.ct}_f$ that are generated in the public key mode, i.e, $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(\mathsf{pp})$. Along with witness encryption ciphertext, they sample a *new* $\mathsf{Mix\text{-}FE}$ master secret key $\mathsf{mfe.msk}$, generate a secret key $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$. Normal mode of NIZK language remains the same. However, in the trapdoor mode, we check if $\mathsf{com}^{(1)}$ is a commitment of $\mathsf{mfe.msk}$ and $\mathsf{mfe.ct}_f$ is generated in the secret key mode, i.e, $\exists\ r$ such that $\mathsf{mfe.ct}_f = \mathsf{Mix\text{-}FE.skEnc}(\mathsf{msk}, f; r)$. WE language on the other hand, checks if NIZK proof verifies, $\mathsf{Mix\text{-}FE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}_f) \stackrel{?}{=} 0$, and $f(x) \stackrel{?}{=} 1$. Note that in the construction, as $\mathsf{mfe.ct}_f$ is generated in the public key mode, WE language is always satisfied if $f(x) = 1$.

To argue adaptive security, we start by sampling $\mathsf{mfe.msk}$ early and switching $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f)$ for each key generation query. At this point, we are not using $\mathsf{mfe.msk}$ anywhere in NIZK language and thus this change is unnoticeable by the attacker using the security of $\mathsf{Mix\text{-}FE}$. From here on we proceed similarly to GGSW and set $\mathsf{com}^{(1)}$ to be a commitment of $\mathsf{mfe.msk}$, generate proofs in the trapdoor mode, set $\mathsf{com}^{(0)}$ to be a commitment of 1. Once this happens, any valid witness for the WE language must be such that $f(x) = 1$ and $\mathsf{Mix\text{-}FE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}_f) = 0$. But as $\mathsf{com}^{(0)}$ perfectly binds to 1 and $\mathsf{com}^{(1)}$ perfectly binds to $\mathsf{mfe.msk}$, $\mathsf{mfe.ct}_f$ must be generated in the secret key mode and thus $\mathsf{Mix\text{-}FE.Dec}$ reveal $f(x)$. Here, we reach a contradiction as we require $f(x) = 0$ and 1 simultaneously and thus WE instance is unsatisfiable. Now, we will look at how to use $\mathsf{Mix\text{-}FE}$ to argue adaptive security of DABE described previously.

Adaptive DABE via Mixed-FE? Using Mixed-FE similar to [WW24], it stands to reason that we can more or less realize adaptive security readily. However, this argument is subtle and not so straight-forward. Consider the following (erroneous) way of using Mixed-FE to realize adaptively secure DABE from selectively secure DABE via dual-systems paradigm.

The changes to the selectively secure DABE as follows — in KGen and $\mathsf{Delegate}$ algorithm, we simply generate $\mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(\mathsf{pp})$ and give this out along with the secret key. In Enc, we sample a fresh $\mathsf{mfe.msk}$, generate $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}FE.KGen}(\mathsf{mfe.msk}, x)$, and give this out along with WE ciphertext. WE language checks if NIZK verifies, $\mathsf{Mix\text{-}FE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}) = 0$, and $f_1 \wedge \ldots f_\ell(x) = 1$. Trapdoor for NIZK language $\mathcal{L}_i$, similar to [WW24] checks if $\exists\ r$ such that $\mathsf{mfe.ct} = \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_i; r)$ and $\mathsf{com}^{(1)}$ is a commitment of said $\mathsf{mfe.msk}$. As values from ($\mathsf{Mix\text{-}FE.pkEnc}, \mathsf{Mix\text{-}FE.KGen}$) always output 0, correctness still holds even for any delegated keys.

However, we cannot argue adaptive security readily. The issue is that when we are delegating keys, generating $\mathsf{mfe.ct}_{f\wedge g}$ in the public key mode completely erases any dependence on the underlying $\mathsf{mfe.ct}_f$. To see this more clearly, consider the final security hybrid — we set $\mathsf{com}^{(0)} \leftarrow \mathsf{Com}(1)$, $\mathsf{com}^{(1)} = \mathsf{Com}(\mathsf{mfe.msk}; r^{(1)})$, and each π_ℓ is generated in the trapdoor mode using $(r^{(1)}, \mathsf{mfe.msk}, r)$ as the witness. At this point, we hope to use WE security and argue that the instance is unsatisfiable. However, that is not the case! A satisfying witness for the WE instance can be generated as follows: sample $\pi_f \leftarrow \mathsf{NIZK.Prove}(\mathtt{inst} := (\mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \mathsf{mfe.ct}_f), \mathtt{wit} := (r^{(1)}, \mathsf{mfe.msk}, r))$ where $\mathsf{mfe.ct}_f = \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f; r)$ and then simply delegate (f, π_f). While delegating, we generate $\mathsf{mfe.ct}_{f\wedge g} \leftarrow \mathsf{Mix\text{-}FE.pkEnc}(\mathsf{pp})$ and verify that π_f is a valid witness to generate $\pi_{f\wedge g}$. Note that $\pi_{f\wedge g}$ does not require any secret information to generate and by generating $\mathsf{mfe.ct}_{f\wedge g}$ in the public key mode, we made sure that $\mathsf{Mix\text{-}FE.Dec}$ outputs 0 and not $f \wedge g(x)$. Thus, if $f \wedge g(x) = 1$,

this is a valid witness for WE instance and we cannot use WE security anymore. Hence, by using $\mathsf{Mix\text{-}FE.pkEnc}$ to delegate, we cannot reach the contradiction[6] similar to [WW24].

In order to achieve adaptive security $\mathsf{mfe.ct}_{f\wedge g}$ should be generated such that it contains some information about f embedded in it and yet it is unnoticeable by the attacker as long as $f(x) = 0$. Hence, we require some notion of "hierarchical" variant of mixed functional encryption in which we have a separate $\mathsf{Delegate}$ algorithm that preserves information about f. That is, if $\mathsf{Mix\text{-}FE}$ has a public delegation algorithm such that we can generate $\mathsf{mfe.ct}_{f\wedge g} \leftarrow \mathsf{Delegate}(\mathsf{mfe.ct}_f, g)$ and $\mathsf{Mix\text{-}FE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}_{f\wedge g}) = f \wedge g(x)$ where $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}FE.skEnc}(\mathsf{mfe.msk}, f)$, then we can overcome this issue. We can readily use the "hierarchical" variant's $\mathsf{Delegate}$ algorithm in DABE's delegation algorithm to achieve adaptive security. Towards this end, we propose the notion of mixed hierarchical functional encryption scheme and prove adaptive security of DABE using this primitive. We briefly review the algorithms and the security definition we require from this primitive below.

Mixed Hierarchical Functional Encryption. A mixed hierarchical functional encryption (Mixed-HFE) is the mixed notion of hierarchical functional encryption [BCG+17] where we can generate ciphertexts in public mode using pkEnc and in secret mode using skEnc. In addition, there is a public delegation algorithm $\mathsf{Delegate}$ that can be used to delegate *ciphertexts*[7]. That is, $\mathsf{Delegate}$ takes $\mathsf{mfe.ct}_f$ and function g to output $\mathsf{mfe.ct}_{f\wedge g}$. Similar to Mixed-FE, we require that any $\mathsf{mfe.ct}_f$ generated using pkEnc and $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{msk}, x)$ should always output 0. Moreover, any ciphertext generated as a result of delegation done on top of $\mathsf{mfe.ct}_f$ should also output 0. For DABE, we require a stronger notion of skEnc where this algorithm generates ct for any sequence of policies $(f_1, \ldots, f_{\ell^*})$. That is $\mathsf{mfe.ct}_{\ell^*} \leftarrow \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{msk}, (f_1, \ldots, f_{\ell^*}))$ and $\mathsf{mfe.sk}_x$ should reveal $f_1 \wedge \ldots f_{\ell^*}(x)$ similar to Mixed-FE. Moreover, we also require that for any $i \in [\ell^* + 1, \ell]$, $\mathsf{mfe.ct}_i \leftarrow \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{ct}_{i-1}, f_i)$, we need that $\mathsf{Mix\text{-}HFE.Dec}(\mathsf{sk}_x, \mathsf{ct}_\ell) = f_1 \wedge \ldots \wedge f_\ell(x)$.

The security of Mixed-HFE we require is quite strong. Essentially, we require an attacker to adaptively query for one secret key for x and arbitrarily-many ciphertexts in three modes — $\mathsf{Store}, \mathsf{Delegate}, \mathsf{Corrupt}$. These are defined similarly to DABE security definition. The attacker's job is to distinguish between two oracles $\mathcal{O}_0$ and $\mathcal{O}_1$. $\mathcal{O}_0$ on Store query generates $\mathsf{mfe.ct}_f \leftarrow \mathsf{Mix\text{-}HFE.pkEnc}(\mathsf{mfe.pp})$ and stores it, on $\mathsf{Delegate}$ query performs $\mathsf{mfe.ct}_{f\wedge g} \leftarrow \mathsf{Delegate}(\mathsf{mfe.ct}_f, g)$, and on $\mathsf{Corrupt}$ query sends $\mathsf{mfe.ct}_p$ to the attacker. $\mathcal{O}_1$ on the other hand, on Store and $\mathsf{Delegate}$ query, does not generate any ciphertexts and simply stores the entire delegation tree. $\mathcal{O}_1$ only on the $\mathsf{Corrupt}$ query performs $\mathsf{mfe.ct}_p \leftarrow \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{msk}, p)$ where p is interpreted as a sequence of policies $(f_1, \ldots, f_\ell)$. Both $\mathcal{O}_0$ and $\mathcal{O}_1$ generate $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x)$ and an admissible attacker only corrupts the policies such that $f_1 \wedge \ldots \wedge f_\ell(x) = 0$. We provide the formal definition of this notion in Section 6.

Adaptive DABE via Mixed-HFE. Realizing adaptive security via Mixed-HFE is fairly straightforward and similar to [WW24]. We alter the $\mathcal{L}_i$ of selective DABE to check in the normal mode that $\mathsf{mfe.ct}_i$ of Mixed-HFE is delegated properly by additionally requiring a random r'_i such that $\mathsf{mfe.ct}_i = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}_{i-1}, f_i; r'_i)$. In the trapdoor mode, we check $\mathsf{com}^{(1)} = \mathsf{Com}(\mathsf{msk}; r^{(1)})$ and $\mathsf{mfe.ct}_i = \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, (f_1, \ldots, f_i))$. WE language now checks for

[6] A prior version of this work made this exact error and incorrectly claimed that using WE, there exists a DABE scheme with unbounded delegations.

[7] Note that in regular definition of hierarchical functional encryption, delegation is done on secret keys. However, for DABE, we require the ability to delegate on ciphertexts generated over functions.

any witness of the format $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ that π_ℓ verifies with respect to $\mathcal{L}_\ell$, $\mathsf{Mix\text{-}HFE.Dec}(\mathsf{mfe.ct}_\ell, \mathsf{mfe.sk}_x) \stackrel{?}{=} 0$ and $f_1 \wedge \ldots \wedge f_\ell(x) \stackrel{?}{=} 1$.

In the construction as all $\mathsf{mfe.ct}_1$ are generated using the pkEnc and delegated appropriately, correctness holds. In order to argue adaptive security, we proceed similarly to selective version where we delay generation of NIZK proofs to $\mathsf{Corrupt}$ phase and simulate all NIZKs. Now, as neither delegation randomness r' nor $\mathsf{mfe.msk}$ is present in the view of attacker anywhere, we can switch to skEnc mode of Mixed-HFE and generate $\mathsf{mfe.ct}_i$ only when $\mathsf{Corrupt}$ query is made using $\mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, (f_1, \ldots, f_\ell))$. This change remains unnoticeable by the attacker by security of Mixed-HFE. Then, we change $\mathsf{com}^{(1)} = \mathsf{Com}(\mathsf{mfe.msk}; r^{(1)})$, generate NIZKs in the trapdoor mode, and set $\mathsf{com}^{(0)} \leftarrow \mathsf{Com}(1)$. Now, as $\mathsf{com}^{(0)}$ is perfectly binding to 1, $\mathsf{com}^{(1)}$ is perfectly binding to $\mathsf{mfe.msk}$, any satisfying WE witness $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ must contain an ℓ^* such that $\mathsf{mfe.ct}_{\ell^*} \leftarrow \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, (f_1, \ldots, f_{\ell^*}))$ and for each $i \in [\ell^* + 1, \ell], \mathsf{mfe.ct}_i \leftarrow \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}_{i-1}, f_i)$. Thus, $\mathsf{Mix\text{-}HFE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}_\ell) = f_1 \wedge \ldots \wedge f_\ell(x)$. Thus, we reach the same contradiction as [WW24] and can use WE security as the instance is unsatisfiable. Full construction and proof are provided in Section 7. Note that the same constant-depth delegation restriction to this construction as well.

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 $[n] := [1, n]$. If $a > b$, then $[a, b]$ should be interpreted as an empty set. 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 $\mathsf{negl}(\cdot)$ such that $\forall\ \lambda \in \mathbb{N}$,

$$\left|\Pr\left[0 \leftarrow \mathcal{A}^{\mathcal{O}_0(\cdot,\cdot)}(\mathsf{crs})\right] - \Pr\left[0 \leftarrow \mathcal{A}^{\mathcal{O}_1(\cdot,\cdot)}(\mathsf{crs})\right]\right| \leq \mathsf{negl}(\lambda)$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$ and $\mathcal{O}_b(\mu_0, \mu_1)$ responds with $\mathsf{com} \leftarrow \mathsf{Com}(\mathsf{crs}, \mu_b)$ for $b \in \{0, 1\}$.

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

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

where probability is taken on the randomness of crs.

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.

$\mathsf{Setup}(1^\lambda, 1^s) \rightarrow \mathsf{crs}$**.** 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.

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

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

Definition 3.2 (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 $\forall\ \lambda \in \mathbb{N}, s = s(\lambda)$,

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

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 3.3 (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 $\forall\ \lambda \in \mathbb{N}, x \notin \mathcal{L}, |\mu_0| = |\mu_1|$,

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

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 $\mathcal{F} = \{\mathcal{F}_{n(\lambda),s(\lambda)}\}_{\lambda \in \mathbb{N}}$ consists of the following polynomial time algorithms.

$\mathsf{Setup}(1^\lambda, 1^d, 1^n, 1^s) \to (\mathsf{PP}, \mathsf{MSK})$**.** The probabilistic setup algorithm takes as input security parameter λ, maximum delegation depth d, maximum attribute length $n = n(\lambda)$, maximum policy size $s = s(\lambda)$, outputs public parameters PP and master secret key MSK. The following algorithms take PP implicitly.

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

$\mathsf{Enc}(x, \mu) \to \mathsf{CT}_x$**.** The probabilistic encryption algorithm takes attribute $x \in \{0,1\}^n$, 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 $f \wedge g, \mathsf{SK}_{f \wedge g}$.

$\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}) \to \mu'/\perp$**.** 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 $\perp$.

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 $\mathcal{F}_{n,s}$ if it satisfies the following properties.

Correctness. There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, positive polynomial $n = n(\lambda), s = s(\lambda), f \in \mathcal{F}_{n,s}, x \in \{0,1\}^n$,

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

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

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 $\ell \in [d]$ where delegation depth $d = d(\lambda)$ is a positive polynomial. 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 PPT adversary, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\forall\ \lambda \in \mathbb{N}$,

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

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 $\mathsf{h} := \mathsf{h} + 1$.
- $(\mathsf{Delegate}, (\mathsf{h}', g)) \rightarrow \mathsf{h}$. $\mathcal{A}$ 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 $\mathsf{h} := \mathsf{h} + 1$.
- $(\mathsf{Corrupt}, \mathsf{h}') \rightarrow \mathsf{SK}_f$. $\mathcal{A}$ sends handle h'. If there is no tuple of the form $(\mathsf{SK}_f, \mathsf{h}', *)$, output $\perp$. Otherwise, send SK_f to $\mathcal{A}$.

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 (Selectively secure DABE)**.** A DABE scheme $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Delegate}, \mathsf{Dec})$ is said to be a selectively secure DABE scheme for policy class $\mathcal{F}_{n,s}$ if in the above security game, the challenge attribute x^* is declared a-priori by $\mathcal{A}$ even before receiving PP.

5 Selectively Secure DABE with Bounded Delegations from Witness Encryption

In this section, we provide the construction of a selectively secure DABE scheme with bounded delegations.

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

- A statistically-sound NIZK scheme (Definition 3.2) for languages $\mathcal{L}_1, \ldots, \mathcal{L}_d$ (Figure 1, 2 resp.).
- A WE scheme (Definition 3.3) for language $\mathcal{L}_{\mathsf{WE}}$ (Figure 3).
- A statistically-binding COM scheme (Definition 3.1).

Language $\mathcal{L}_1$

Instance: $\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f$
Witness: $r^{(0)}, r^{(1)}, x^*$
Relation: Output 1 if and only if $\phi_1 \vee (\phi_2 \wedge \phi_3) = 1$ where,

- $\phi_1 : \mathsf{com}^{(0)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, 0; r^{(0)})$
- $\phi_2 : \mathsf{com}^{(1)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, x^*; r^{(1)})$
- $\phi_3 : f(x^*) \stackrel{?}{=} 0$.

Figure 1: Description of $\mathcal{L}_1$

Language $\mathcal{L}_i$ for $i \in [2, d]$

Instance: $\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, f_2, \ldots, f_i$
Witness: $\pi_{i-1}, r^{(1)}, x^*$
Relation: Output 1 if and only if $\phi_1 \vee (\phi_2 \wedge \phi_3) = 1$ where,

- $\phi_1 : 1 \stackrel{?}{=} \mathsf{NIZK.Verify}(\mathsf{nizk.crs}_{i-1}, (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_{i-1}), \pi_{i-1})$.
- $\phi_2 : \mathsf{com}^{(1)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, x^*; r^{(1)})$
- $\phi_3 : f_1 \wedge \ldots \wedge f_i(x^*) \stackrel{?}{=} 0$.

Figure 2: Description of $\mathcal{L}_2, \ldots, \mathcal{L}_d$

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

Instance: $\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s, x$
Witness: $f_1, \ldots, f_\ell, \pi_\ell$ for some $\ell \in [d]$.
Relation: Output 1 if and only if,

1. If $\ell = 1$, $1 \stackrel{?}{=} \mathsf{NIZK.Verify}(\mathsf{nizk.crs}_1, (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1), \pi_1)$. Otherwise, $1 \stackrel{?}{=} \mathsf{NIZK.Verify}(\mathsf{nizk.crs}_\ell, (\mathsf{nizk.crs}_{\ell-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_\ell), \pi_\ell)$.
2. $1 \stackrel{?}{=} f_1(x) \wedge \ldots \wedge f_\ell(x)$

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

$\underline{\mathsf{Setup}(1^\lambda, 1^d, 1^n, 1^s)}$**.** Sample $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. In addition, sample $\mathsf{com.crs} \leftarrow \mathsf{COM.Setup}(1^\lambda)$. Compute $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$ and $\mathsf{com}^{(1)} = \mathsf{Com}(0^n; r^{(1)})$ where $r^{(0)}, r^{(1)} \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$.

Set $\mathsf{PP} := (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$ and $\mathsf{MSK} := r^{(0)}$. Output $(\mathsf{PP}, \mathsf{MSK})$.

$\underline{\mathsf{KGen}(\mathsf{MSK}, f)}$**.** Parse MSK as $r^{(0)}$. Compute $\pi_f \leftarrow \mathsf{NIZK.Prove}(\mathsf{nizk.crs}_1, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} := (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f), \mathtt{wit} := (r^{(0)}, \overline{0})$[8]. Output $\mathsf{SK}_f := (f, \pi_f)$.

$\underline{\mathsf{Enc}(x, \mu)}$**.** Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}, \mu)$ where $\mathtt{inst} := (\mathsf{PP}, x)$. Output $\mathsf{CT}_x := (x, \mathsf{we.ct})$.

$\underline{\mathsf{Delegate}(\mathsf{SK}_f, g)}$**.** Parse SK_f as $(f_1, \ldots, f_\ell, \pi_\ell)$ for some $\ell \in [d-1]$.

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

$\underline{\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}_x)}$**.** Parse SK_f as $(f_1, \ldots, f_\ell, \pi_\ell)$ for some $\ell \in [d]$ and CT_x as $(x, \mathsf{we.ct})$. If $f_i(x) = 0$ for any $i \in [\ell]$, abort and output $\perp$. Otherwise, output $\mu' := \mathsf{WE.Dec}(\mathsf{we.ct}, \mathsf{SK}_f)$.

Correctness. The correctness of the construction follows from the correctness of WE, NIZK, and COM. Since $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$ for some $r^{(0)}$, we have by correctness of NIZK and COM, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_1, (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f)), \pi_1) = 1$. Hence, if $f(x) = 1$, we have by correctness of WE, $\mu' = \mu$.

Delegation Correctness. We argue this by induction on $\ell \in [d]$. When $\ell = 1$, its the same as correctness property. Consider $\ell = 2$. We have that $\mathsf{SK}_{f \wedge g} = (f, g, \pi_2)$. If SK_f was generated using KGen, we have that $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_1, (\mathsf{com}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f), \pi_1) = 1$ (from correctness property). Hence, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_2, \mathtt{inst}_2, \pi_2) = 1$ for $\mathtt{inst}_2 = (\mathsf{nizk.crs}_1, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f, g)$. 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, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_{k+1}, \mathtt{inst}_{k+1}, \pi_{k+1}) = 1$. This is because by induction hypothesis,

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

$\forall\ i \in [2,k], \mathsf{NIZK.Verify}(\mathsf{nizk.crs}_i, (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i), \pi_i) = 1$. Thus, by WE correctness, $\mu' = \mu$.

Delegation Depth. It is clear from the description of $\mathcal{L}_1$ that $\mathsf{NIZK}_1.\mathsf{Verify}$ runs in $\mathsf{poly}(\lambda, n, s)$ time. Hence, running time $\mathsf{NIZK}_2.\mathsf{Verify}$ that depends on the circuit $\mathsf{NIZK}_1.\mathsf{Verify}$ is $\mathsf{poly}(\lambda, n, s, \mathsf{poly}(\lambda, n, s))$. Proceeding this way, running time of $\mathsf{NIZK}_d.\mathsf{Verify}$ is at most $\mathsf{poly}(\lambda, n, s)^{O(d)}$. Hence, for decryption to be efficient, we require the $\mathcal{L}_{\mathsf{WE}}$ to be efficiently verifiable. Thus $d = O(1)$. Moreover, running times of Delegate and Enc in the worst case are bounded by $\mathsf{poly}(\lambda, n, s)^{O(d)}$.

5.1 Security Analysis

In this section, we prove that Construction 5.1 satisfies selective security as defined in Definition 4.2. Specifically, we prove the following theorem.

Theorem 5.2. If WE is a WE scheme for $\mathcal{L}_{\mathsf{WE}}$ (Definition 3.3), COM is a COM scheme (Definition 3.1), $\mathsf{NIZK}_1, \ldots \mathsf{NIZK}_d$ are statistically-sound NIZK schemes for $\mathcal{L}_1, \ldots, \mathcal{L}_d$ (Definition 3.2), then Construction 5.1 is a selectively secure DABE scheme (Definition 4.2) for polynomial-size policies.

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

$\mathbf{Expt}_0^{\mathcal{A},b}(1^\lambda)$. This is the honest experiment with $\mathcal{O}$ where either $b = 0/1$.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(0^n; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Sample $\pi_f \leftarrow \mathsf{NIZK}_1.\mathsf{Prove}(\mathsf{nizk.crs}_1, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f), \mathtt{wit} = (r^{(0)}, \overline{0})$. Set $\mathsf{SK}_f = (f, \pi_f)$. 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 $(f_1, \ldots, f_\ell, \pi_\ell)$. Sample $\pi_{\ell+1} \leftarrow \mathsf{NIZK}_{\ell+1}.\mathsf{Prove}(\mathsf{nizk.crs}_{\ell+1}, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{nizk.crs}_\ell, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \ldots, f_\ell, g), \mathtt{wit} = (\pi_\ell, \overline{0})$.

  Set $\mathsf{SK}_{f \wedge g} = (\ldots, f_\ell, g, \pi_{\ell+1})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_{f\wedge g}, \mathsf{h}, \mathsf{h}')$ and set $\mathsf{h} := \mathsf{h} + 1$.
- **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 μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

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

$\underline{\mathbf{Expt}_1^{\mathcal{A},b}(1^\lambda)}$. In this hybrid, we set $\mathsf{com}^{(1)}$ to be a commitment of x^*. The changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store, Delegate, Corrupt.** Same as $\mathbf{Expt}_0^{\mathcal{A},b}(1^\lambda)$.
- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

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

$\underline{\mathbf{Expt}_2^{\mathcal{A},b}(1^\lambda)}$. In this experiment we delay generation of all NIZK proofs to corrupt phase. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup1}^\lambda)$ for $i \in [d]$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{h}), \mathsf{h}, \bot)$, send h to $\mathcal{A}$, and $\mathsf{h} := \mathsf{h} + 1$.
- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$ to $\mathcal{A}$. Otherwise, store $((\mathsf{SK}_f, g, \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Let $\ell' < \ell$ be the largest value such that $(*, \mathsf{h}_{\ell'}, *)$-th entry contains a NIZK proof.

  If no such ℓ' exists, sample $\pi_1 \leftarrow \mathsf{NIZK}_1.\mathsf{Prove}(\mathsf{nizk.crs}_1, \mathtt{inst}_1, \mathtt{wit}_1)$. For each $i \in [2, \ell]$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_1, \ldots, \mathsf{h}_\ell$ with the corresponding proofs. That is, set h_i-th entry as $((f_1, \ldots, f_i, \pi_i), \mathsf{h}_i, \mathsf{h}_{i-1})$ for $i \in [2, \ell]$ and h_1-th entry as $((f_1, \pi_1), \mathsf{h}_1, \bot)$.

  Otherwise, for each $i \in [\ell' + 1, \ell]$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_{\ell'+1}, \ldots, \mathsf{h}_\ell$ with the corresponding proofs.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i)$. $\mathtt{wit}_1 = (r^{(0)}, \overline{0})$ and $\mathtt{wit}_i = (\pi_{i-1}, \overline{0})$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell)$ to $\mathcal{A}$.
- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

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

$\mathbf{Expt}^{\mathcal{A},b}_{3,j}(1^\lambda)$ for $j \in [d+1]$. In this experiment we simulate the NIZK proofs for the first $j-1$ NIZK instantiations. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)})$, for $i \in [d]$, if $i < j$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$. Otherwise, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{h}), \mathsf{h}, \perp)$, send h to $\mathcal{A}$, 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, store $((\mathsf{SK}_f, g, \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Let $\ell' < \ell$ be the largest value such that $(*, \mathsf{h}_{\ell'}, *)$-th entry contains a NIZK proof.

  If no such ℓ' exists, for $i \in [\ell]$, if $i < j$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(\mathtt{inst}_i)$. Otherwise, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_1, \ldots, \mathsf{h}_\ell$ with the corresponding proofs. That is, set h_i-th entry as $((f_1, \ldots, f_i, \pi_i), \mathsf{h}_i, \mathsf{h}_{i-1})$ for $i \in [2, \ell]$ and h_1-th entry as $((f_1, \pi_1), \mathsf{h}_1, \perp)$.

  Otherwise, for each $i \in [\ell' + 1, \ell]$, if $i < j$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(\mathtt{inst}_i)$. Otherwise, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_{\ell'+1}, \ldots, \mathsf{h}_\ell$ with the corresponding proofs.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i)$. $\mathtt{wit}_1 = (r^{(0)}, \bar{0})$ and $\mathtt{wit}_i = (\pi_{i-1}, \bar{0})$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell)$ to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

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

$\mathbf{Expt}^{\mathcal{A},b}_{4}(1^\lambda)$. In this experiment we do not generate a chain of metaproofs anymore. We simply simulate the last proof in the chain during the Corrupt query. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$ for $i \in [d]$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{h}), \mathsf{h}, \perp)$, send h to $\mathcal{A}$, 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, store $((\mathsf{SK}_f, g, \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Sample $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Sim}(\mathtt{inst}_\ell)$.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i)$. Send $(f_1, \ldots, f_\ell, \pi_\ell)$ to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

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

$\mathbf{Expt}^{\mathcal{A},b}_{5,j}(1^\lambda)$ for $j \in [d+1]$. In this experiment we generate NIZK proofs using $r^{(1)}, x^*$ as the witness for the first $j-1$ NIZK instantiations. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)})$, for $i \in [d]$, if $i < j$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$. Otherwise, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{h}), \mathsf{h}, \perp)$, send h to $\mathcal{A}$, 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, store $((\mathsf{SK}_f, g, \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. If $\ell < j$, $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Prove}(\mathsf{nizk.crs}_\ell, \mathtt{inst}_\ell, \mathtt{wit}_\ell)$. Otherwise, $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Sim}(\mathtt{inst}_\ell)$.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i)$. $\mathtt{wit}_i = (\overline{0}, r^{(1)}, x^*)$ for $i \in [d]$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell)$ to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

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

$\mathbf{Expt}^{\mathcal{A},b}_{6}(1^\lambda)$. In this experiment we set $\mathsf{com}^{(0)}$ as a commitment of 1. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, $\mathsf{com}^{(0)} = \mathsf{Com}(1; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store, Delegate, Corrupt.** Same as $\mathbf{Expt}^{\mathcal{A},b}_{5,d+1}(1^\lambda)$.

- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

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

$\underline{\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)}$**.** In this experiment, we encrypt all zero string using WE. This experiment is independent of b. Changes are highlighted in red.

- **Setup, Store, Delegate, Corrupt.** Same as $\mathbf{Expt}_6^{\mathcal{A},b}(1^\lambda)$.

- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, 0^{|\mu_b|})$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $(x, \mathsf{we.ct})$ to $\mathcal{A}$.

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

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

Notation. Let $\mathcal{A}$ be an admissible adversary in the DABE security definition. By $p^{\mathcal{A}}_{i(,j)}$ we denote the probability of $b' = b$ in the experiment $\mathbf{Expt}^{\mathcal{A},b}_{i(,j)}(1^\lambda)$. That is $p^{\mathcal{A}}_{i(,j)} := \Pr\left[b \leftarrow \mathbf{Expt}^{\mathcal{A},b}_{i(,j)}(1^\lambda)\right]$.

Lemma 5.3. Assuming that COM satisfies computational hiding property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p^{\mathcal{A}}_0 - p^{\mathcal{A}}_1\right| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that there exists an adversary $\mathcal{A}$ such that $\left|p^{\mathcal{A}}_0 - p^{\mathcal{A}}_1\right| = \epsilon(\lambda)$ for some non-negligible $\epsilon(\cdot)$. We will construct a reduction adversary $\mathcal{B}$ that can break the computational hiding property of COM. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows:

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Receive $\mathsf{com.crs}$ from $\mathcal{O}$. Compute $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$. Send $(0^n, x^*)$ to $\mathcal{O}$ to receive $\mathsf{com}^{(1)}$. Sample $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store, Delegate, Corrupt.** Same as $\mathbf{Expt}_0^{\mathcal{A},b}(1^\lambda)$.

- **Encryption.** Sample $b \xleftarrow{\$} \{0,1\}$. $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

- **Guess.** $\mathcal{A}$ outputs b'. If $b = b'$, output 0. Otherwise output 1.

As we can see, the running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$ and λ. If $\mathcal{O}$ commits to 0^n, $\mathcal{B}$ simulates $\mathbf{Expt}_0^{\mathcal{A},b}(1^\lambda)$ and if $\mathcal{O}$ commits to x^*, $\mathcal{B}$ behaves like $\mathbf{Expt}_1^{\mathcal{A},b}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the computational hiding property of COM that can break its security with probability $\epsilon(\lambda)$. Thus, $\mathbf{Expt}_0^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_1^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable. □

Lemma 5.4. For any admissible adversary $\mathcal{A}$, $p^{\mathcal{A}}_1 = p^{\mathcal{A}}_2$.

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 $\mathsf{NIZK}_i.\mathsf{Prove}$.
- *$\mathcal{A}$ 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 already generated π_ℓ previously. This forms a key for the required policy.

Hence, these two experiments are identical. Thus, lemma follows. □

Lemma 5.5. For any admissible adversary $\mathcal{A}$, $p_2^{\mathcal{A}} = p_{3,1}^{\mathcal{A}}$.

Proof. In $\mathbf{Expt}_{3,1}^{\mathcal{A},b}(1^\lambda)$, we are not simulating any NIZK instantiations. Hence, both experiments are identical. Thus, the lemma follows. □

Lemma 5.6. For any $j \in [d]$, assuming that NIZK_j satisfies adaptive computational zero-knowledge property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p_{3,j}^{\mathcal{A}} - p_{3,j+1}^{\mathcal{A}}\right| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that there exists admissible adversary $\mathcal{A}$ such that $\left|p_{3,j}^{\mathcal{A}} - p_{3,j+1}^{\mathcal{A}}\right| = \epsilon(\lambda)$ for some $j \in [d]$ and non-negligible $\epsilon(\cdot)$. We will construct a reduction adversary $\mathcal{B}$ that can break the adaptive computational zero-knowledge property of NIZK_j. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows:

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)})$, for $i \in [d]$, if $i < j$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$. Otherwise, if $i = j$, receive $\mathsf{nizk.crs}_j$ from $\mathcal{O}$. Otherwise, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{h}), \mathsf{h}, \perp)$, send h to $\mathcal{A}$, 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, store $((\mathsf{SK}_f, g, \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Let $\ell' < \ell$ be the largest value such that $(*, \mathsf{h}_{\ell'}, *)$-th entry contains a NIZK proof.

  If no such ℓ' exists, for $i \in [\ell]$, if $i < j$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(\mathtt{inst}_i)$. Otherwise, if $i = j$, send $(\mathtt{inst}_i, \mathtt{wit}_i)$ to $\mathcal{O}$ to receive π_i. Otherwise, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_1, \ldots, \mathsf{h}_\ell$ with the corresponding proofs. That is, set h_i-th entry as $((f_1, \ldots, f_i, \pi_i), \mathsf{h}_i, \mathsf{h}_{i-1})$ for $i \in [2, \ell]$ and h_1-th entry as $((f_1, \pi_1), \mathsf{h}_1, \perp)$.

  Otherwise, for each $i \in [\ell' + 1, \ell]$, if $i < j$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(\mathtt{inst}_i)$. Otherwise, if $i = j$, send $(\mathtt{inst}_i, \mathtt{wit}_i)$ to $\mathcal{O}$ to receive π_i. Otherwise, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_{\ell'+1}, \ldots, \mathsf{h}_\ell$ with the corresponding proofs.

Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i)$. $\mathtt{wit}_1 = (r^{(0)}, \overline{0})$ and $\mathtt{wit}_i = (\pi_{i-1}, \overline{0})$.

Send $(f_1, \ldots, f_\ell, \pi_\ell)$ to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

- **Guess.** $\mathcal{A}$ outputs b'. If $b = b'$, output 0. Otherwise, output 1.

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 $(\mathsf{NIZK}_j.\mathsf{Setup}, \mathsf{NIZK}_j.\mathsf{Prove})$, $\mathcal{B}$ simulates $\mathbf{Expt}^{\mathcal{A},b}_{3,j}(1^\lambda)$ and if $\mathcal{O}$ use $\mathsf{NIZK}_j.\mathsf{Sim}$, $\mathcal{B}$ behaves like $\mathbf{Expt}^{\mathcal{A},b}_{3,j+1}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the adaptive computational zero-knowledge property of NIZK_j that can break its security with probability $\epsilon(\lambda)$. Thus, $\mathbf{Expt}^{\mathcal{A},b}_{3,j}(1^\lambda)$ and $\mathbf{Expt}^{\mathcal{A},b}_{3,j+1}(1^\lambda)$ are computationally indistinguishable. □

Lemma 5.7. For any admissible adversary $\mathcal{A}$, $p^{\mathcal{A}}_{3,d+1} = p^{\mathcal{A}}_4$.

Proof. In both $\mathbf{Expt}^{\mathcal{A},b}_{3,d+1}(1^\lambda)$ and $\mathbf{Expt}^{\mathcal{A},b}_{4}(1^\lambda)$, π_ℓ and $\pi_{\ell+1}$ are completely unrelated for any $\ell \in [d+1]$. Thus, it wouldn't matter whether all proofs are generated or only the final proof in the chain is generated. Thus, the lemma follows. □

Lemma 5.8. For any admissible adversary $\mathcal{A}$, $p^{\mathcal{A}}_4 = p^{\mathcal{A}}_{5,1}$.

Proof. In $\mathbf{Expt}^{\mathcal{A},b}_{5,1}(1^\lambda)$, we do not stop simulating NIZK instantiations. Hence, both experiments are identical. Thus, the lemma follows. □

Lemma 5.9. For any $j \in [d]$, assuming that NIZK_j satisfies adaptive computational zero-knowledge property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p^{\mathcal{A}}_{5,j} - p^{\mathcal{A}}_{5,j+1}\right| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that there exists admissible adversary $\mathcal{A}$ such that $\left|p^{\mathcal{A}}_{5,j} - p^{\mathcal{A}}_{5,j+1}\right| = \epsilon(\lambda)$ for some $j \in [d]$ and non-negligible $\epsilon(\cdot)$. We will construct a reduction adversary $\mathcal{B}$ that can break the adaptive computational zero-knowledge property of NIZK_j. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows:

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s, x^*$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(x^*; r^{(1)})$, for $i \in [d]$, if $i < j$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$. Otherwise, if $i = j$, receive $\mathsf{nizk.crs}_i$ from $\mathcal{O}$. Otherwise, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

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

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

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. If $\ell < j$, $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Prove}(\mathsf{nizk.crs}_\ell, \mathtt{inst}_\ell, \mathtt{wit}_\ell)$. Otherwise, if $\ell = j$, send $(\mathtt{inst}_i, \mathtt{wit}_i)$ to $\mathcal{O}$ to receive π_ℓ. Otherwise, $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Sim}(\mathtt{inst}_\ell)$.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i)$. $\mathtt{wit}_i = (\overline{0}, r^{(1)}, x^*)$ for $i \in [d]$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell)$ to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends μ_0, μ_1. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x)$. Send $\mathsf{CT} = (x^*, \mathsf{we.ct})$ to $\mathcal{A}$.

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

- **Guess.** $\mathcal{A}$ sends b'. If $b = b'$, output 0. Otherwise, output 1.

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 $(\mathsf{NIZK}_j.\mathsf{Setup}, \mathsf{NIZK}_j.\mathsf{Prove})$, $\mathcal{B}$ simulates $\mathbf{Expt}^{\mathcal{A},b}_{5,j+1}(1^\lambda)$ and if $\mathcal{O}$ use $\mathsf{NIZK}_j.\mathsf{Sim}$, $\mathcal{B}$ behaves like $\mathbf{Expt}^{\mathcal{A},b}_{5,j}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the adaptive computational zero-knowledge property of NIZK_j that can break its security with probability $\epsilon(\lambda)$. Thus, $\mathbf{Expt}^{\mathcal{A},b}_{5,j}(1^\lambda)$ and $\mathbf{Expt}^{\mathcal{A},b}_{5,j+1}(1^\lambda)$ are computationally indistinguishable. □

Lemma 5.10. Assuming that COM satisfies computational hiding property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p^{\mathcal{A}}_{5,d+1} - p^{\mathcal{A}}_6\right| \leq \mathsf{negl}(\lambda)$.

Proof. Note that in both $\mathbf{Expt}^{\mathcal{A},b}_{5,d+1}(1^\lambda)$ and $\mathbf{Expt}^{\mathcal{A},b}_{6}(1^\lambda)$, we no longer use $r^{(0)}$ anywhere. Thus, we can readily construct a reduction to computational hiding property of COM. The proof of this lemma is similar to proof of Lemma 5.3. □

Lemma 5.11. Assuming the semantic security property of WE, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p^{\mathcal{A}}_6 - p^{\mathcal{A}}_7\right| \leq \mathsf{negl}(\lambda)$

Proof. In experiments $\mathbf{Expt}^{\mathcal{A},b}_{6}(1^\lambda)$ and $\mathbf{Expt}^{\mathcal{A},b}_{7}(1^\lambda)$, $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s, x)$ is unsatisfiable with high probability. Assume that there is $\mathtt{wit}_{\mathsf{WE}} = (f_1, \ldots, f_\ell, \pi_\ell)$ for some $\ell \in [d]$ such that $\mathsf{WE.R}(\mathtt{inst}_{\mathsf{WE}}, \mathtt{wit}_{\mathsf{WE}}) = 1$. Then as $\mathsf{com}^{(0)}$ is a statistically binding commitment of 1 and $\mathsf{com}^{(1)}$ is a statistically binding commitment of x^*, there must exist an $\ell^* \in [\ell]$ such that $f_1 \wedge \ldots \wedge f_{\ell^*}(x^*) = 0$. But as $\mathsf{WE.R}(\mathtt{inst}_{\mathsf{WE}}, \mathtt{wit}_{\mathsf{WE}}) = 1$, it also means that $f_1 \wedge \ldots \wedge f_{\ell^*}(x^*) = 1$ which is a contradiction. Hence, with high probability due to statistical soundness of $\mathsf{NIZK}_1, \ldots, \mathsf{NIZK}_d$, we have that $\mathcal{L}_{\mathsf{WE}}$ is an empty language. Thus, we can use the semantic security of WE to switch μ_b to $0^{|\mu_b|}$. Any PPT adversary that can notice this switch can be easily reduced to break semantic security of WE. □

Note that the ciphertext in $\mathbf{Expt}^{\mathcal{A},b}_{7}(1^\lambda)$ is independent of b. Thus $p^{\mathcal{A}}_7 = \frac{1}{2}$. Hence, security of Construction 5.1 follows. □

6 Mixed Hierarchical Functional Encryption

In this section, we formally define a mixed hierarchical functional encryption scheme (Mixed-HFE) that is sufficient to realize adaptively secure delegatable attribute-based encryption scheme. As mentioned in Section 2, a Mixed-HFE scheme is an extension of mixed functional encryption scheme [GKW18] with the capacity to delegate ciphertexts.

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

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

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

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

$\mathsf{skEnc}(\mathsf{MSK}, f_1, \ldots, f_\ell) \to \mathsf{CT}$**.** The probabilistic *secret key delegation* encryption algorithm takes as input master secret key MSK, policies $f_1, \ldots, f_\ell$, and outputs ciphertext CT.

$\mathsf{pkEnc}(\mathsf{PP}) \to \mathsf{CT}$**.** The probabilistic *public-key* encryption outputs ciphertext CT.

$\mathsf{Delegate}(\mathsf{CT}, g) \to \mathsf{CT}'$**.** The probabilistic delegation algorithm takes as input a ciphertext CT for some sequence of policies $f_1, \ldots, f_\ell$, policy g, and outputs a ciphertext CT' for policies $f_1, \ldots, f_\ell, g$.

$\mathsf{Dec}(\mathsf{SK}_x, \mathsf{CT}) \to 0/1$**.** The deterministic decryption algorithm takes as input secret key SK_x, ciphertext for some sequence of policies $f_1, \ldots, f_\ell$, CT, and outputs either 0 or 1.

Definition 6.1 (Mixed-HFE)**.** A Mix-HFE scheme $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{skEnc}, \mathsf{pkEnc}, \mathsf{Delegate}, \mathsf{Dec})$ is said to be a Mixed-HFE scheme for policy class $\mathcal{F} = \{\mathcal{F}_{n(\lambda)})\}_{\lambda \in \mathbb{N}}$ if it satisfies the following properties.

Public Correctness. There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, positive polynomial $s = s(\lambda), n = n(\lambda), \ell = \ell(\lambda)$, input $x \in \{0,1\}^n$, and any policies $f_1, \ldots, f_\ell$ such that size of $f_1 \wedge \ldots \wedge f_\ell$ is at most s and each policy takes n-bit inputs,

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

$$\Pr\left[0 = \mathsf{Dec}(\mathsf{SK}_x, \mathsf{CT}_\ell) \quad : \quad \begin{array}{l} \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^s), \mathsf{MSK} \leftarrow \mathsf{Gen}(\mathsf{PP}), \\ \mathsf{SK}_x \leftarrow \mathsf{KGen}(\mathsf{MSK}, x), \\ \mathsf{CT}_1 \leftarrow \mathsf{pkEnc}(\mathsf{PP}), \\ \forall\ i \in [2, \ell], \mathsf{CT}_i \leftarrow \mathsf{Delegate}(\mathsf{CT}_{i-1}, f_i) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda)$$

Secret Correctness. There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, positive polynomial $s = s(\lambda), n = n(\lambda), \ell = \ell(\lambda)$, input $x \in \{0,1\}^n$, and policies $f_1, \ldots, f_\ell$ such that size of $f_1 \wedge \ldots \wedge f_\ell$ is at most s and each take n-bit inputs, and any $\ell^* \in [\ell]$,

$$\Pr\left[\begin{array}{l|l} f_1 \wedge \ldots \wedge f_\ell(x) = & \mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^s), \mathsf{MSK} \leftarrow \mathsf{Gen}(\mathsf{PP}), \\ \mathsf{Dec}(\mathsf{SK}_x, \mathsf{CT}_\ell) & \mathsf{SK}_x \leftarrow \mathsf{KGen}(\mathsf{MSK}, x), \\ & \mathsf{CT}_{\ell^*} \leftarrow \mathsf{skEnc}(\mathsf{MSK}, f_1, \ldots, f_{\ell^*}), \\ & \forall\ i \in [\ell^*+1, \ell], \mathsf{CT}_i \leftarrow \mathsf{Delegate}(\mathsf{CT}_{i-1}, f_i) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

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

$$\left|\Pr\left[b \leftarrow \mathbf{Expt}_b^{\mathcal{A}}(1^\lambda)\ :\ b \xleftarrow{\$} \{0,1\}\right] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$$

where $\mathbf{Expt}_b^{\mathcal{A}}(1^\lambda)$ is defined as follows:

- **Setup.** $\mathcal{A}$ sends $1^n, 1^s$. Sample $\mathsf{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^s), \mathsf{MSK} \leftarrow \mathsf{Gen}(\mathsf{PP})$. Send PP to $\mathcal{A}$. Initialize $\mathsf{h} := 1$.
- **Store.** $\mathcal{A}$ sends policy f. If $b = 0$, sample $\mathsf{CT} \leftarrow \mathsf{pkEnc}(\mathsf{PP})$ and store $(\mathsf{CT}, f, \mathsf{h}, \perp)$ and send h to $\mathcal{A}$. Otherwise, store $(f, \mathsf{h}, \perp)$ and send h to $\mathcal{A}$. Increment $\mathsf{h} := \mathsf{h} + 1$.
- **Delegate.** $\mathcal{A}$ sends handle h' and policy g.

  If $b = 0$: and there is no entry for $(\mathsf{CT}, (f_1, \ldots, f_\ell), \mathsf{h}', *)$, respond with $\perp$. Otherwise, sample $\mathsf{CT}' \leftarrow \mathsf{Delegate}(\mathsf{CT}, g)$, store $(\mathsf{CT}', (f_1, \ldots, f_\ell, g), \mathsf{h}, \mathsf{h}')$, and send h to $\mathcal{A}$.

  If $b = 1$: If there is no entry for $((f_1, \ldots, f_\ell), \mathsf{h}', *)$, respond with $\perp$. Otherwise, store $((f_1, \ldots, f_\ell, g), \mathsf{h}, \mathsf{h}')$ and send h to $\mathcal{A}$.

  Increment $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends handle h'.

  If $b = 0$: and there is no entry for $(\mathsf{CT}, (f_1, \ldots, f_\ell), \mathsf{h}', *)$, respond with $\perp$. Otherwise, retrieve the entry and send CT to $\mathcal{A}$.

  If $b = 1$: If there is no entry for $((f_1, \ldots, f_\ell), \mathsf{h}', *)$, respond with $\perp$. Otherwise, retrieve the entry and sample $\mathsf{CT} \leftarrow \mathsf{skEnc}(\mathsf{MSK}, f_1, \ldots, f_\ell)$ and send CT to $\mathcal{A}$.
- **Secret Key.** $\mathcal{A}$ sends input $x \in \{0,1\}^n$. Sample $\mathsf{SK}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{MSK}, x)$. Send SK_x to $\mathcal{A}$.
- **Store, Delegate, Corrupt.** These are handled similarly to the queries before secret key declaration.
- **Guess.** $\mathcal{A}$ outputs guess b'. Output b'.

A stateful PPT machine $\mathcal{A}$ is said to be admissible if for any **Corrupt** query made by $\mathcal{A}$, CT is for any sequence of policies $f_1, \ldots, f_\ell$ such that $f_1(x) \wedge \ldots \wedge f_\ell(x) = 0$.

7 Adaptively Secure DABE with Bounded Delegations from Witness Encryption and Mixed Hierarchical Functional Encryption

In this section, we provide the construction of an adaptively secure DABE scheme with bounded delegations by realizing delegatable dual-systems framework via delegatable Mixed-HFE.

Construction 7.1. We provide the construction of an adaptively secure DABE scheme with bounded d-many delegations (Definition 4.1) for any family of polynomial-size policies using the following components:

- A statistically-sound NIZK scheme (Definition 3.2) for languages $\mathcal{L}_1, \ldots, \mathcal{L}_d$ (Figure 4, 5 resp.).
- A WE scheme (Definition 3.3) for language $\mathcal{L}_{\mathsf{WE}}$ (Figure 6).
- A statistically-binding COM scheme (Definition 3.1).
- A Mixed-HFE scheme (Definition 6.1) for polynomial-size policies.

Language $\mathcal{L}_1$

Instance: $\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f, \mathsf{mfe.ct}$
Witness: $r^{(0)}, r^{(1)}, r^*, r_1^{\mathsf{ct}}$
Relation: Output 1 if and only if $\phi_1 \vee (\phi_2 \wedge \phi_3) = 1$ where,

- $\phi_1 : \mathsf{com}^{(0)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, 0; r^{(0)})$
- $\phi_2 : \mathsf{com}^{(1)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, r^*; r^{(1)})$
- $\phi_3 : \mathsf{mfe.msk} \stackrel{?}{=} \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*) \wedge \mathsf{mfe.ct} \stackrel{?}{=} \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, f; r_1^{\mathsf{ct}})$

Figure 4: Description of $\mathcal{L}_1$

Language $\mathcal{L}_i$ for $i \in [2, d]$

Instance: $\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, f_2, \ldots, f_i, \mathsf{mfe.ct}'$
Witness: $\pi_{i-1}, \mathsf{mfe.ct}, r', r^{(1)}, r^*, r_i^{\mathsf{ct}}$
Relation: Output 1 if and only if $(\phi_1 \wedge \phi_2) \vee (\phi_3 \wedge \phi_4) = 1$ where,

- $\phi_1 : 1 \stackrel{?}{=} \mathsf{NIZK.Verify}(\mathsf{nizk.crs}_{i-1}, (\mathsf{com.crs}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_{i-1}), \pi_{i-1})$
- $\phi_2 : \mathsf{mfe.ct}' \stackrel{?}{=} \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}, f_i; r')$
- $\phi_3 : \mathsf{com}^{(1)} \stackrel{?}{=} \mathsf{Com}(\mathsf{com.crs}, r^*; r^{(1)})$
- $\phi_4 : \mathsf{mfe.msk} \stackrel{?}{=} \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*) \wedge \mathsf{mfe.ct}' \stackrel{?}{=} \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_i; r_i^{\mathsf{ct}})$

Figure 5: Description of $\mathcal{L}_2, \ldots, \mathcal{L}_d$

$\underline{\mathsf{Setup}(1^\lambda, 1^d, 1^n, 1^s)}$**.** Sample $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. In addition, sample $\mathsf{com.crs} \leftarrow \mathsf{COM.Setup}(1^\lambda)$. Compute $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$ and $\mathsf{com}^{(1)} = \mathsf{Com}(0^\lambda; r^{(1)})$ where $r^{(0)}, r^{(1)} \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$. $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$.

Set $\mathsf{PP} := (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$ and $\mathsf{MSK} := r^{(0)}$. Output $(\mathsf{PP}, \mathsf{MSK})$.

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

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

1. If $\ell = 1$, $1 \stackrel{?}{=} \mathsf{NIZK.Verify}(\mathsf{nizk.crs}_1, \mathtt{inst}_1, \pi_1)$. Otherwise, $1 \stackrel{?}{=} \mathsf{NIZK.Verify}(\mathsf{nizk.crs}_\ell, \mathtt{inst}_\ell, \pi_\ell)$.
2. $1 \stackrel{?}{=} f_1(x) \wedge \ldots \wedge f_\ell(x)$
3. $0 \stackrel{?}{=} \mathsf{Mix\text{-}HFE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}_\ell)$

Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \mathsf{mfe.ct}_1)$ and
$\mathtt{inst}_\ell = (\mathsf{nizk.crs}_{\ell-1}, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_\ell, \mathsf{mfe.ct}_\ell)$

Figure 6: Description of $\mathcal{L}_{\mathsf{WE}}$

$\underline{\mathsf{KGen}(\mathsf{MSK}, f)}$**.** Parse MSK as $r^{(0)}$. Sample $\mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}HFE.pkEnc}(\mathsf{mfe.pp})$. Compute $\pi_f \leftarrow \mathsf{NIZK.Prove}(\mathsf{nizk.crs}_1, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} := (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f, \mathsf{mfe.ct}), \mathtt{wit} := (r^{(0)}, \overline{0})$. Output $\mathsf{SK}_f := (f, \pi_f, \mathsf{mfe.ct})$.

$\underline{\mathsf{Enc}(x, \mu)}$**.** Sample $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp})$ and $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}, \mu)$ 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 $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ for some $\ell \in [d-1]$. Compute $\mathsf{mfe.ct}_{\ell+1} = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}_\ell, g; r')$ for some $r' \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$.

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

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

$\underline{\mathsf{Dec}(\mathsf{SK}_f, \mathsf{CT}_x)}$**.** Parse SK_f as $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ 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 $\mu' := \mathsf{WE.Dec}(\mathsf{we.ct}, \mathsf{SK}_f)$.

Correctness. The correctness of the construction follows from the correctness of WE, NIZK, Mix-HFE, and COM. Since $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$ for some $r^{(0)}$, we have by correctness of NIZK, and COM, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_1, (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f, \mathsf{mfe.ct})), \pi_1) = 1$. Moreover as $\mathsf{mfe.msk} \leftarrow \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp})$, $\mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}HFE.pkEnc}(\mathsf{mfe.pp})$, and $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x)$, with high probability, $\mathsf{Mix\text{-}HFE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}) = 0$. Hence, if $f(x) = 1$, we have by correctness of WE, $\mu' = \mu$.

Delegation Correctness. We argue this by induction on $\ell \in [d]$. When $\ell = 1$, its the same as correctness property. Consider $\ell = 2$. We have that $\mathsf{SK}_{f \wedge g} = (f, g, \pi_2, \mathsf{mfe.ct}')$ where $\mathsf{mfe.ct}' = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}, g; r')$ for some $r' \stackrel{\$}{\leftarrow} \{0,1\}^\lambda$. If SK_f was generated using KGen, we have that $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_1, (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f, \mathsf{mfe.ct}), \pi_1) = 1$ (from correctness property). Hence, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_2, \mathtt{inst}_2, \pi_2) = 1$ for $\mathtt{inst}_2 = (\mathsf{nizk.crs}_1, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f, g, \mathsf{mfe.ct}')$. Thus, with high probability, $0 = \mathsf{Mix\text{-}HFE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}')$. 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, $\mathsf{NIZK.Verify}(\mathsf{nizk.crs}_{k+1}, \mathtt{inst}_{k+1}, \pi_{k+1}) = 1$. This is because by induction hypothesis, $\forall\ i \in [2,k], \mathsf{NIZK.Verify}(\mathsf{nizk.crs}_i, (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i, \mathsf{mfe.ct}_i), \pi_i) = 1$ where $\mathsf{mfe.ct}_i = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}_{i-1}, f_i; r_i)$ for some $r_i \xleftarrow{\$} \{0,1\}^\lambda$. Thus, by WE correctness, $\mu' = \mu$.

Delegation Depth. Similar to Construction 5.1, we require $d = O(1)$. Moreover, running times of Delegate and Enc in the worst case are bounded by $\mathsf{poly}(\lambda, n, s)^{O(d)}$.

7.1 Security Analysis

In this section, we prove that Construction 7.1 satisfies adaptive security as defined in Definition 4.1. Specifically, we prove the following theorem.

Theorem 7.2. If WE is a WE scheme for $\mathcal{L}_{\mathsf{WE}}$ (Definition 3.3), COM is a COM scheme (Definition 3.1), $\mathsf{NIZK}_1, \ldots \mathsf{NIZK}_d$ are statistically-sound NIZK schemes for $\mathcal{L}_1, \ldots, \mathcal{L}_d$ (Definition 3.2), Mix-HFE is a Mixed-HFE scheme (Definition 6.1) for polynomial-size predicates, then Construction 7.1 is an adaptively secure DABE scheme (Definition 4.1) for polynomial-size policies.

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

$\mathbf{Expt}_0^{\mathcal{A},b}(1^\lambda)$. This is almost the honest experiment with $\mathcal{O}_b$ where either $b = 0/1$. In this experiment, we sample $\mathsf{mfe.msk}$ during setup. The distributions remain identical nonetheless.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^\lambda; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Compute $\mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}HFE.pkEnc}(\mathsf{mfe.pp})$. Sample $\pi_f \leftarrow \mathsf{NIZK}_1.\mathsf{Prove}(\mathsf{nizk.crs}_1, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f, \mathsf{mfe.ct})$, $\mathtt{wit} = (r^{(0)}, \overline{0})$. Set $\mathsf{SK}_f = (f, \pi_f, \mathsf{mfe.ct})$. 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 $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$. Sample $\mathsf{mfe.ct}_{\ell+1} \leftarrow \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}_\ell, g; r')$ Sample $\pi_{\ell+1} \leftarrow \mathsf{NIZK}_{\ell+1}.\mathsf{Prove}(\mathsf{nizk.crs}_{\ell+1}, \mathtt{inst}, \mathtt{wit})$ where $\mathtt{inst} = (\mathsf{nizk.crs}_\ell, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, \ldots, f_\ell, g, \mathsf{mfe.ct}_{\ell+1})$, $\mathtt{wit} = (\pi_\ell, \mathsf{mfe.ct}_\ell, r', \overline{0})$.

  Set $\mathsf{SK}_{f \wedge g} = (\ldots, f_\ell, g, \pi_{\ell+1}, \mathsf{mfe.ct}_{\ell+1})$. Send h to $\mathcal{A}$. Store $(\mathsf{SK}_{f\wedge g}, \mathsf{h}, \mathsf{h}')$ and set $\mathsf{h} := \mathsf{h} + 1$.
- **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 $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.

$\mathbf{Expt}_1^{\mathcal{A},b}(1^\lambda)$. In this experiment we delay generation of all NIZK proofs to corrupt phase. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^\lambda; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{mfe.ct}, \mathsf{h}), \mathsf{h}, \bot)$ where $\mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}HFE.pkEnc}(\mathsf{mfe.pp})$, send h to $\mathcal{A}$, and $\mathsf{h} := \mathsf{h} + 1$.
- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$ to $\mathcal{A}$. Otherwise, parse SK_f to get $\mathsf{mfe.ct}$, sample $\mathsf{mfe.ct}' = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}, g; r')$ where $r' \xleftarrow{\$} \{0,1\}^\lambda$ store $((\mathsf{SK}_f, g, \mathsf{mfe.ct}', r', \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{mfe.ct}_1, \mathsf{h}_1, f_2, \mathsf{mfe.ct}_2, r_2, \mathsf{h}_2, \ldots, f_\ell, \mathsf{mfe.ct}_\ell, r_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Let $\ell' < \ell$ be the largest value such that $(*, \mathsf{h}_{\ell'}, *)$-th entry contains a NIZK proof.

  If no such ℓ' exists, sample $\pi_1 \leftarrow \mathsf{NIZK}_1.\mathsf{Prove}(\mathsf{nizk.crs}_1, \mathtt{inst}_1, \mathtt{wit}_1)$. For each $i \in [2, \ell]$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_1, \ldots, \mathsf{h}_\ell$ with the corresponding proofs. That is, set h_i-th entry as $((f_1, \ldots, f_i, \pi_i, \mathsf{mfe.ct}_i), \mathsf{h}_i, \mathsf{h}_{i-1})$ for $i \in [2, \ell]$ and h_1-th entry as $((f_1, \pi_1, \mathsf{mfe.ct}_1), \mathsf{h}_1, \bot)$.

  Otherwise, for each $i \in [\ell' + 1, \ell]$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_{\ell'+1}, \ldots, \mathsf{h}_\ell$ with the corresponding proofs.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \mathsf{mfe.ct}_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i, \mathsf{mfe.ct}_i)$. $\mathtt{wit}_1 = (r^{(0)}, \overline{0})$ and $\mathtt{wit}_i = (\pi_{i-1}, \mathsf{mfe.ct}_{i-1}, r_i, \overline{0})$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ to $\mathcal{A}$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.

$\mathbf{Expt}_{2,j}^{\mathcal{A},b}(1^\lambda)$ for $j \in [d+1]$. In this experiment we simulate the NIZK proofs for the first $j-1$ NIZK instantiations. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^\lambda; r^{(1)})$, for $i \in [d]$, if $i < j$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$. Otherwise, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{mfe.ct}, \mathsf{h}), \mathsf{h}, \bot)$ where $\mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}HFE.pkEnc}(\mathsf{mfe.pp})$, send h to $\mathcal{A}$, and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$ to $\mathcal{A}$. Otherwise, parse SK_f to get $\mathsf{mfe.ct}$, sample $\mathsf{mfe.ct}' = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}, g; r')$ where $r' \xleftarrow{\$} \{0,1\}^\lambda$ store $((\mathsf{SK}_f, g, \mathsf{mfe.ct}', r', \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$. Otherwise, parse SK_f as $(f_1, \mathsf{mfe.ct}_1, \mathsf{h}_1, f_2, \mathsf{mfe.ct}_2, r_2, \mathsf{h}_2, \ldots, f_\ell, \mathsf{mfe.ct}_\ell, r_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Let $\ell' < \ell$ be the largest value such that $(*, \mathsf{h}_{\ell'}, *)$-th entry contains a NIZK proof.

  If no such ℓ' exists, for $i \in [\ell]$, if $i < j$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(\mathtt{inst}_i)$. Otherwise, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_1, \ldots, \mathsf{h}_\ell$ with the corresponding proofs. That is, set h_i-th entry as $((f_1, \ldots, f_i, \pi_i, \mathsf{mfe.ct}_i), \mathsf{h}_i, \mathsf{h}_{i-1})$ for $i \in [2, \ell]$ and h_1-th entry as $((f_1, \pi_1, \mathsf{mfe.ct}_1), \mathsf{h}_1, \bot)$.

  Otherwise, for each $i \in [\ell' + 1, \ell]$, for $i \in [\ell' + 1, \ell]$, if $i < j$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(\mathtt{inst}_i)$. Otherwise, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_{\ell'+1}, \ldots, \mathsf{h}_\ell$ with the corresponding proofs.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \mathsf{mfe.ct}_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i, \mathsf{mfe.ct}_i)$. $\mathtt{wit}_1 = (r^{(0)}, \overline{0})$ and $\mathtt{wit}_i = (\pi_{i-1}, \mathsf{mfe.ct}_{i-1}, r_i, \overline{0})$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ to $\mathcal{A}$.

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

$\mathbf{Expt}_3^{\mathcal{A},b}(1^\lambda)$. In this experiment we do not generate a chain of metaproofs anymore. We simply simulate the last proof in the chain during the Corrupt query. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)}), \mathsf{com}^{(1)} = \mathsf{Com}(0^\lambda; r^{(1)}), \mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$ for $i \in [d]$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{mfe.ct}, \mathsf{h}), \mathsf{h}, \bot)$ where $\mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}HFE.pkEnc}(\mathsf{mfe.pp})$, send h to $\mathcal{A}$, and $\mathsf{h} := \mathsf{h} + 1$.

- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$ to $\mathcal{A}$. Otherwise, parse SK_f to get $\mathsf{mfe.ct}$, sample $\mathsf{mfe.ct}' = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}, g; r')$ where $r' \xleftarrow{\$} \{0,1\}^\lambda$ store $((\mathsf{SK}_f, g, \mathsf{mfe.ct}', r', \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{mfe.ct}_1, \mathsf{h}_1, f_2, \mathsf{mfe.ct}_2, r_2, \mathsf{h}_2, \ldots, f_\ell, \mathsf{mfe.ct}_\ell, r_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Sample $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Sim}(\mathtt{inst}_\ell)$.

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

Send $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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 hybrid, we sample $\mathsf{mfe.ct}$'s in the secret mode. The changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^\lambda; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$ for $i \in [d]$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

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

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

- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Sample $\mathsf{mfe.ct}_\ell = \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, f_1, \ldots, f_\ell; r_\ell^{\mathsf{ct}}), r_\ell^{\mathsf{ct}} \xleftarrow{\$} \{0,1\}^\lambda$, and $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Sim}(\mathtt{inst}_\ell)$.

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

  Send $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ to $\mathcal{A}$.

- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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 hybrid, we set $\mathsf{com}^{(1)}$ to be a commitment of r^*. The changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(r^*; r^{(1)})$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store, Delegate, Corrupt.** Same as $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.

$\mathbf{Expt}_{6,j}^{\mathcal{A},b}(1^\lambda)$ for $j \in [d+1]$. In this experiment we generate NIZK proofs using $r^{(1)}, r^*, r_\ell^{\mathsf{ct}}$ as the witness for the first $j-1$ NIZK instantiations. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(r^*; r^{(1)})$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$, for $i \in [d]$, if $i < j$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i\mathsf{.Setup}(1^\lambda)$. Otherwise, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i\mathsf{.Sim}(1^\lambda)$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{h}), \mathsf{h}, \bot)$, send h to $\mathcal{A}$, and $\mathsf{h} := \mathsf{h} + 1$.
- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$ to $\mathcal{A}$. Otherwise, store $((\mathsf{SK}_f, g, \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$, $\mathsf{mfe.ct}_\ell = \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, f_1, \ldots, f_\ell; r_\ell^{\mathsf{ct}}), r_\ell^{\mathsf{ct}} \xleftarrow{\$} \{0,1\}^\lambda$. If $\ell < j$, $\pi_\ell \leftarrow \mathsf{NIZK}_\ell\mathsf{.Prove}(\mathsf{nizk.crs}_\ell, \mathtt{inst}_\ell, \mathtt{wit}_\ell)$. Otherwise, $\pi_\ell \leftarrow \mathsf{NIZK}_\ell\mathsf{.Sim}(\mathtt{inst}_\ell)$.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \mathsf{mfe.ct}_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i, \mathsf{mfe.ct}_i)$. $\mathtt{wit}_i = (\overline{0}, r^{(1)}, r^*, r_\ell^{\mathsf{ct}})$ for $i \in [d]$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell)$ to $\mathcal{A}$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.

$\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)$. In this experiment we set $\mathsf{com}^{(0)}$ as a commitment of 1. Changes are highlighted in red.

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda)$, $\mathsf{com}^{(0)} = \mathsf{Com}(1; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(r^*; r^{(1)})$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i\mathsf{.Setup}(1^\lambda)$ for $i \in [d]$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.

- **Store, Delegate, Corrupt.** Same as $\mathbf{Expt}^{\mathcal{A},b}_{6,d+1}(1^\lambda)$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.

$\mathbf{Expt}^{\mathcal{A},b}_8(1^\lambda)$. In this experiment, we encrypt all zero string using WE. This experiment is independent of b. Changes are highlighted in red.

- **Setup, Store, Delegate, Corrupt.** Same as $\mathbf{Expt}^{\mathcal{A},b}_7(1^\lambda)$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, 0^{|\mu_b|})$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.

Notation. Let $\mathcal{A}$ be an admissible adversary in the DABE security definition. By $p^{\mathcal{A}}_{i(,j)}$ we denote the probability of $b' = b$ in the experiment $\mathbf{Expt}^{\mathcal{A},b}_{i(,j)}(1^\lambda)$. That is $p^{\mathcal{A}}_{i(,j)} := \Pr\left[b \leftarrow \mathbf{Expt}^{\mathcal{A},b}_{i(,j)}(1^\lambda)\right]$.

Lemma 7.3. For any admissible adversary $\mathcal{A}$, $p^{\mathcal{A}}_0 = p^{\mathcal{A}}_1$.

Proof. Note that all that's changed is we are delaying NIZK proving in KGen and $\mathsf{Delegate}$ procedures to $\mathsf{Corrupt}$ query. We are still generating $\mathsf{mfe.ct}$ exactly as $\mathbf{Expt}^{\mathcal{A},b}_0(1^\lambda)$. In $\mathsf{Corrupt}$ phase everything 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 $\mathsf{NIZK}_i.\mathsf{Prove}$.
- *$\mathcal{A}$ 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 already generated $\pi_\ell, \mathsf{mfe.ct}_\ell$ previously. This forms a key for the required policy.

Hence, these two experiments are identical. Thus, lemma follows. □

Lemma 7.4. For any admissible adversary $\mathcal{A}$, $p^{\mathcal{A}}_1 = p^{\mathcal{A}}_{2,1}$.

Proof. In $\mathbf{Expt}^{\mathcal{A},b}_{2,1}(1^\lambda)$, we are not simulating any NIZK instantiations. Hence, both experiments are identical. Thus, the lemma follows. □

Lemma 7.5. For any $j \in [d]$, assuming that NIZK_j satisfies adaptive computational zero-knowledge property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p^{\mathcal{A}}_{2,j} - p^{\mathcal{A}}_{2,j+1}\right| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that there exists admissible adversary $\mathcal{A}$ such that $\left|p^{\mathcal{A}}_{2,j} - p^{\mathcal{A}}_{2,j+1}\right| = \epsilon(\lambda)$ for some $j \in [d]$ and non-negligible $\epsilon(\cdot)$. We will construct a reduction adversary $\mathcal{B}$ that can break the adaptive computational zero-knowledge property of NIZK_j. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows:

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^\lambda; r^{(1)})$, for $i \in [d]$, if $i < j$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$. Otherwise, if $i = j$, receive $\mathsf{nizk.crs}_i$ from $\mathcal{O}$. Otherwise, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{mfe.ct}, \mathsf{h}), \mathsf{h}, \bot)$ where $\mathsf{mfe.ct} \leftarrow \mathsf{Mix\text{-}HFE.pkEnc}(\mathsf{mfe.pp})$, send h to $\mathcal{A}$, and $\mathsf{h} := \mathsf{h} + 1$.
- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$ to $\mathcal{A}$. Otherwise, parse SK_f to get $\mathsf{mfe.ct}$, sample $\mathsf{mfe.ct}' = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}, g; r')$ where $r' \xleftarrow{\$} \{0,1\}^\lambda$ store $((\mathsf{SK}_f, g, \mathsf{mfe.ct}', r', \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$. Otherwise, parse SK_f as $(f_1, \mathsf{mfe.ct}_1, \mathsf{h}_1, f_2, \mathsf{mfe.ct}_2, r_2, \mathsf{h}_2, \ldots, f_\ell, \mathsf{mfe.ct}_\ell, r_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$. Let $\ell' < \ell$ be the largest value such that $(*, \mathsf{h}_{\ell'}, *)$-th entry contains a NIZK proof.

  If no such ℓ' exists, for $i \in [\ell]$, if $i < j$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(\mathtt{inst}_i)$. Otherwise, if $i = j$, send $(\mathtt{inst}_i, \mathtt{wit}_i)$ to $\mathcal{O}$ to receive π_i. Otherwise, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_1, \ldots, \mathsf{h}_\ell$ with the corresponding proofs. That is, set h_i-th entry as $((f_1, \ldots, f_i, \pi_i, \mathsf{mfe.ct}_i), \mathsf{h}_i, \mathsf{h}_{i-1})$ for $i \in [2, \ell]$ and h_1-th entry as $((f_1, \pi_1, \mathsf{mfe.ct}_1), \mathsf{h}_1, \bot)$.

  Otherwise, for each $i \in [\ell'+1, \ell]$, for $i \in [\ell'+1, \ell]$, if $i < j$, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(\mathtt{inst}_i)$. Otherwise, if $i = j$, send $(\mathtt{inst}_i, \mathtt{wit}_i)$ to $\mathcal{O}$ to receive π_i. Otherwise, sample $\pi_i \leftarrow \mathsf{NIZK}_i.\mathsf{Prove}(\mathsf{nizk.crs}_i, \mathtt{inst}_i, \mathtt{wit}_i)$. Update entries for $\mathsf{h}_{\ell'+1}, \ldots, \mathsf{h}_\ell$ with the corresponding proofs.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \mathsf{mfe.ct}_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i, \mathsf{mfe.ct}_i)$. $\mathtt{wit}_1 = (r^{(0)}, \overline{0})$ and $\mathtt{wit}_i = (\pi_{i-1}, \mathsf{mfe.ct}_{i-1}, r_i, \overline{0})$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ to $\mathcal{A}$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.** $\mathcal{A}$ outputs guess b'. Output 0 if and only if $b = b'$. Otherwise, output 1.

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 $(\mathsf{NIZK}_j.\mathsf{Setup}, \mathsf{NIZK}_j.\mathsf{Prove})$, $\mathcal{B}$ simulates $\mathbf{Expt}^{\mathcal{A},b}_{2,j}(1^\lambda)$ and if $\mathcal{O}$ use $\mathsf{NIZK}_j.\mathsf{Sim}$, $\mathcal{B}$ behaves like

$\mathbf{Expt}_{2,j+1}^{\mathcal{A},b}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the adaptive computational zero-knowledge property of NIZK_j that can break its security with probability $\epsilon(\lambda)$. Thus, $\mathbf{Expt}_{2,j}^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_{2,j+1}^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable. □

Lemma 7.6. For any admissible adversary $\mathcal{A}$, $p_{2,d+1}^{\mathcal{A}} = p_3^{\mathcal{A}}$.

Proof. In both $\mathbf{Expt}_{2,d+1}^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_3^{\mathcal{A},b}(1^\lambda)$, π_ℓ and $\pi_{\ell+1}$ are completely unrelated for any $\ell \in [d+1]$. Thus, it wouldn't matter whether all proofs are generated or only the final proof in the chain is generated. Thus, the lemma follows. □

Lemma 7.7. Assuming that Mix-HFE satisfies mode indistinguishability property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p_3^{\mathcal{A}} - p_4^{\mathcal{A}}\right| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that there exists an adversary $\mathcal{A}$ such that $\left|p_3^{\mathcal{A}} - p_4^{\mathcal{A}}\right| = \epsilon(\lambda)$ for some non-negligible $\epsilon(\cdot)$. We will construct a reduction adversary $\mathcal{B}$ that can break the mode indistinguishability property of Mix-HFE. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows:

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(0^\lambda; r^{(1)})$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$ for $i \in [d]$. Also, receive $\mathsf{mfe.pp}$ from $\mathcal{O}(1^n, 1^s)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Send (Store, f) to $\mathcal{O}$. Store $((f, \mathsf{h}), \mathsf{h}, \bot)$, send h to $\mathcal{A}$, and $\mathsf{h} := \mathsf{h} + 1$.
- **Delegate.** $\mathcal{A}$ sends h', g. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$ to $\mathcal{A}$. Otherwise, send $(\mathsf{Delegate}, (\mathsf{h}', g))$ to $\mathcal{O}$, store $((\mathsf{SK}_f, g, \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\bot$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, send $(\mathsf{Corr}, \mathsf{h}')$ to $\mathcal{O}$ to receive $\mathsf{mfe.ct}_\ell$, and $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Sim}(\mathtt{inst}_\ell)$.

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

  Send $(f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ to $\mathcal{A}$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Send x to $\mathcal{O}$ to receive $\mathsf{mfe.sk}_x$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.** $\mathcal{A}$ outputs guess b'. If $b = b'$, output 0. Otherwise output 1.

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 bit $\beta = 0$ $\mathcal{B}$ simulates $\mathbf{Expt}_3^{\mathcal{A},b}(1^\lambda)$ and if $\mathcal{O}$ uses bit $\beta = 1$, $\mathcal{B}$ behaves like $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the mode indistinguishability property of Mix-HFE that can break its security with probability $\epsilon(\lambda)$. Thus, $\mathbf{Expt}_3^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable. □

Lemma 7.8. Assuming that COM satisfies computational hiding property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p_4^{\mathcal{A}} - p_5^{\mathcal{A}}\right| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that there exists an adversary $\mathcal{A}$ such that $\left|p_4^{\mathcal{A}} - p_5^{\mathcal{A}}\right| = \epsilon(\lambda)$ for some non-negligible $\epsilon(\cdot)$. We will construct a reduction adversary $\mathcal{B}$ that can break the computational hiding property of COM. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows:

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Receive $\mathsf{com.crs}$ from $\mathcal{O}$, sample $\mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$. Send $(0^\lambda, r^*)$ to $\mathcal{O}$ to receive $\mathsf{com}^{(1)}$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$ for $i \in [d]$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store, Delegate, Corrupt.** Same as $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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.** $\mathcal{A}$ outputs guess b'. Output 0 if and only if $b = b'$. Otherwise, output 1.

As we can see, the running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$ and λ. If $\mathcal{O}$ commits 0^λ $\mathcal{B}$ simulates $\mathbf{Expt}_4^{\mathcal{A},b}(1^\lambda)$ and if $\mathcal{O}$ commits r^*, $\mathcal{B}$ 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 7.9. For any admissible adversary $\mathcal{A}$, $p_5^{\mathcal{A}} = p_{6,1}^{\mathcal{A}}$.

Proof. In $\mathbf{Expt}_{5,1}^{\mathcal{A},b}(1^\lambda)$, we do not stop simulating NIZK instantiations. Hence, both experiments are identical. Thus, the lemma follows. □

Lemma 7.10. For any $j \in [d]$, assuming that NIZK_j satisfies adaptive computational zero-knowledge property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p_{6,j}^{\mathcal{A}} - p_{6,j+1}^{\mathcal{A}}\right| \leq \mathsf{negl}(\lambda)$.

Proof. Assume that there exists admissible adversary $\mathcal{A}$ such that $\left|p_{6,j}^{\mathcal{A}} - p_{6,j+1}^{\mathcal{A}}\right| = \epsilon(\lambda)$ for some $j \in [d]$ and non-negligible $\epsilon(\cdot)$. We will construct a reduction adversary $\mathcal{B}$ that can break the adaptive computational zero-knowledge property of NIZK_j. The description of $\mathcal{B}^{\mathcal{O}}$ is as follows:

- **Setup.** $\mathcal{A}$ sends $1^d, 1^n, 1^s$. Compute $\mathsf{com.crs} \leftarrow \mathsf{Com.Setup}(1^\lambda), \mathsf{com}^{(0)} = \mathsf{Com}(0; r^{(0)})$, $\mathsf{com}^{(1)} = \mathsf{Com}(r^*; r^{(1)})$ where $r^* \xleftarrow{\$} \{0,1\}^\lambda$, for $i \in [d]$, if $i < j$, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Setup}(1^\lambda)$. Otherwise, if $i = j$, receive $\mathsf{nizk.crs}_i$ from $\mathcal{O}$. Otherwise, $\mathsf{nizk.crs}_i \leftarrow \mathsf{NIZK}_i.\mathsf{Sim}(1^\lambda)$. Also, $\mathsf{mfe.pp} \leftarrow \mathsf{Mix\text{-}HFE.Setup}(1^\lambda, 1^n, 1^s)$ and $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$. Set $\mathsf{PP} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s)$, $\mathsf{MSK} := r^{(0)}$. Initiate h to 1. Send PP to $\mathcal{A}$.
- **Store.** $\mathcal{A}$ sends f. Store $((f, \mathsf{h}), \mathsf{h}, \bot)$, send h to $\mathcal{A}$, 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, store $((\mathsf{SK}_f, g, \mathsf{h}), \mathsf{h}, \mathsf{h}')$, send h to $\mathcal{A}$, and set $\mathsf{h} := \mathsf{h} + 1$.
- **Corrupt.** $\mathcal{A}$ sends h'. If there is no entry $(\mathsf{SK}_f, \mathsf{h}', *)$, send $\perp$. Otherwise, if SK_f contains a NIZK proof, send SK_f to $\mathcal{A}$. Otherwise, parse SK_f as $(f_1, \mathsf{h}_1, \ldots, f_\ell, \mathsf{h}_\ell)$ for some $\ell \in [d]$, $\mathsf{mfe.ct}_\ell = \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, f_1, \ldots, f_\ell; r_\ell^{\mathsf{ct}}), r_\ell^{\mathsf{ct}} \xleftarrow{\$} \{0,1\}^\lambda$. If $\ell < j$, $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Prove}(\mathsf{nizk.crs}_\ell, \mathtt{inst}_\ell, \mathtt{wit}_\ell)$. Otherwise, if $\ell = j$, send $(\mathtt{inst}_\ell, \mathtt{wit}_\ell)$ to $\mathcal{O}$ to receive π_ℓ. Otherwise, $\pi_\ell \leftarrow \mathsf{NIZK}_\ell.\mathsf{Sim}(\mathtt{inst}_\ell)$.

  Here, $\mathtt{inst}_1 = (\mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \mathsf{mfe.ct}_1)$ and $\mathtt{inst}_i = (\mathsf{nizk.crs}_{i-1}, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, f_1, \ldots, f_i, \mathsf{mfe.ct}_i)$. $\mathtt{wit}_i = (\overline{0}, r^{(1)}, r^*, r_\ell^{\mathsf{ct}})$ for $i \in [d]$.

  Send $(f_1, \ldots, f_\ell, \pi_\ell)$ to $\mathcal{A}$.
- **Encryption.** $\mathcal{A}$ sends x^*, μ_0, μ_1. Sample $\mathsf{mfe.sk}_x \leftarrow \mathsf{Mix\text{-}HFE.KGen}(\mathsf{mfe.msk}, x^*)$. Sample $\mathsf{we.ct} \leftarrow \mathsf{WE.Enc}(1^\lambda, \mathtt{inst}_{\mathsf{WE}}, \mu_b)$ where $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{PP}, x, \mathsf{mfe.sk}_x)$. 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 $(\mathsf{NIZK}_j.\mathsf{Setup}, \mathsf{NIZK}_j.\mathsf{Prove})$, $\mathcal{B}$ simulates $\mathbf{Expt}_{6,j+1}^{\mathcal{A},b}(1^\lambda)$ and if $\mathcal{O}$ use $\mathsf{NIZK}_j.\mathsf{Sim}$, $\mathcal{B}$ behaves like $\mathbf{Expt}_{6,j}^{\mathcal{A},b}(1^\lambda)$. Hence, $\mathcal{B}$ is a valid adversary against the adaptive computational zero-knowledge property of NIZK_j that can break its security with probability $\epsilon(\lambda)$. Thus, $\mathbf{Expt}_{6,j}^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_{7,j+1}^{\mathcal{A},b}(1^\lambda)$ are computationally indistinguishable. □

Lemma 7.11. Assuming that COM satisfies computational hiding property, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p_{5,d+1}^{\mathcal{A}} - p_6^{\mathcal{A}}\right| \leq \mathsf{negl}(\lambda)$.

Proof. Note that in both $\mathbf{Expt}_{6,d+1}^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)$, we no longer use $r^{(0)}$ anywhere. Thus, we can readily construct a reduction to computational hiding property of COM. The proof of this lemma is similar to proof of Lemma 7.8. □

Lemma 7.12. Assuming the semantic security property of WE, for any admissible adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\left|p_7^{\mathcal{A}} - p_8^{\mathcal{A}}\right| \leq \mathsf{negl}(\lambda)$

Proof. In experiments $\mathbf{Expt}_7^{\mathcal{A},b}(1^\lambda)$ and $\mathbf{Expt}_8^{\mathcal{A},b}(1^\lambda)$, $\mathtt{inst}_{\mathsf{WE}} = (\mathsf{nizk.crs}_1, \ldots, \mathsf{nizk.crs}_d, \mathsf{com.crs}, \mathsf{mfe.pp}, \mathsf{com}^{(0)}, \mathsf{com}^{(1)}, n, s, x, \mathsf{mfe.sk}_x)$ is unsatisfiable with high probability. Assume that there is $\mathtt{wit}_{\mathsf{WE}} = (f_1, \ldots, f_\ell, \pi_\ell, \mathsf{mfe.ct}_\ell)$ for some $\ell \in [d]$ such that $\mathsf{WE.R}(\mathtt{inst}_{\mathsf{WE}}, \mathtt{wit}_{\mathsf{WE}}) = 1$. Then as $\mathsf{com}^{(0)}$ is a statistically binding commitment of 1 and $\mathsf{com}^{(1)}$ is a statistically binding commitment of r^*, there must exist an $\ell^* \in [\ell]$ such that $\mathsf{mfe.ct}_{\ell^*} = \mathsf{Mix\text{-}HFE.skEnc}(\mathsf{mfe.msk}, f_1 \wedge \ldots \wedge f_{\ell^*}; r_{\ell^*}^{\mathsf{ct}})$, $\mathsf{mfe.msk} = \mathsf{Mix\text{-}HFE.Gen}(\mathsf{mfe.pp}; r^*)$. Moreover as $\mathsf{mfe.ct}_i = \mathsf{Mix\text{-}HFE.Delegate}(\mathsf{mfe.ct}_{i-1}, f_i)$ for $i \in [\ell^* + 1, \ell]$, we have that $\mathsf{Mix\text{-}HFE.Dec}(\mathsf{mfe.sk}_x, \mathsf{mfe.ct}_\ell) = f_1 \wedge \ldots f_\ell(x^*) = 0$. But as $\mathsf{WE.R}(\mathtt{inst}_{\mathsf{WE}}, \mathtt{wit}_{\mathsf{WE}}) = 1$, it also means that $f_1 \wedge \ldots \wedge f_\ell(x^*) = 1$ which is a contradiction. Hence, with high probability due to statistical soundness of $\mathsf{NIZK}_1, \ldots, \mathsf{NIZK}_d$, we have that $\mathcal{L}_{\mathsf{WE}}$ is an empty language. Thus, we can use the semantic security of WE to switch μ_b to $0^{|\mu_b|}$. Any PPT adversary that can notice this switch can be easily reduced to break semantic security of WE. □

Note that the ciphertext in $\mathbf{Expt}_8^{\mathcal{A},b}(1^\lambda)$ is independent of b. Thus $p_8^{\mathcal{A}} = \frac{1}{2}$. Hence, security of Construction 7.1 follows. □

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.

[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.

[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.

[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.

[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.

[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.

[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.

[GKR25] Rishab Goyal, Venkata Koppula, and Mahesh Sreekumar Rajasree. A note on adaptive security in hierarchical identity-based encryption. In *Advances in Cryptology – CRYPTO 2025 (to appear)*, 2025.

[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.

[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.

[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.

[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.