Multi-Authority Functional Encryption with Bounded Collusions from Standard Assumptions*

Rishab Goyal
UW-Madison†

Saikumar Yadugiri
UW-Madison‡

Abstract

Multi-Authority Functional Encryption (MA-FE) [*Chase, TCC'07; Lewko-Waters, Eurocrypt'11; Brakerski et al., ITCS'17*] is a popular generalization of functional encryption (FE) with the central goal of decentralizing the trust assumption from a single central trusted key authority to a group of multiple, *independent and non-interacting*, key authorities. Over the last several decades, we have seen tremendous advances in new designs and constructions for FE supporting different function classes, from a variety of assumptions and with varying levels of security. Unfortunately, the same has not been replicated in the multi-authority setting. The current scope of MA-FE designs is rather limited, with positive results only known for (all-or-nothing) attribute-based functionalities, or need full power of general-purpose code obfuscation. This state-of-the-art in MA-FE could be explained in part by the implication provided by Brakerski et al. (ITCS'17). It was shown that a general-purpose obfuscation scheme can be designed from any MA-FE scheme for circuits, even if the MA-FE scheme is secure only in a bounded-collusion model, where at most *two* keys per authority get corrupted.

In this work, we revisit the problem of MA-FE, and show that existing implication from MA-FE to obfuscation is not tight. We provide new methods to design MA-FE for circuits from simple and minimal cryptographic assumptions. Our main contributions are summarized below–

1. We design a $\mathsf{poly}(\lambda)$-authority MA-FE for circuits in the bounded-collusion model. Under the existence of public-key encryption, we prove it to be statically simulation-secure. Further, if we assume sub-exponential security of public-key encryption, then we prove it to be adaptively simulation-secure in the Random Oracle Model.
2. We design a $O(1)$-authority MA-FE for circuits in the bounded-collusion model. Under the existence of 2/3-party non-interactive key exchange, we prove it to be adaptively simulation-secure.
3. We provide a new generic bootstrapping compiler for MA-FE for general circuits to design a simulation-secure $(n_1 + n_2)$-authority MA-FE from any two n_1-authority and n_2-authority MA-FE.

*An extended abstract appeared at TCC 2024. This is the full version.

†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.

Contents

1 Introduction

Functional encryption (FE) [SW05, BSW11] has revolutionized the study of public-key encryption [DH76a]. It challenged the widespread belief that encryption is an *all-or-nothing* primitive, where you either learn the full plaintext or nothing. FE provides fine-grained access over encrypted data enabling recovery of only partial information about plaintext. Over the last several years, viewing through the FE lens has led to dramatic re-envisioning of encryption with varying levels of expressiveness such as identity-based encryption (IBE) [Sha85, BF01, Coc01], attribute-based encryption (ABE) [SW05, GPSW06], inner-product functional encryption (IPFE) [BW07, KSW08], 1-sided/2-sided predicate encryption (PE) [BW07, KSW08, GVW15, GKW17, WZ17] and more.

In the standard FE formulation, there is a central trusted authority that generates the global parameters $(\mathsf{MPK}, \mathsf{MSK})$, a master public-secret key pair. Using the master secret key MSK, it can generate a decryption key SK_x for any attribute x, where SK_x enables decryption to $f(x)$ from any ciphertext CT encrypting a function f. Unfortunately, this trust model is too strong for many applications, as it implicitly embeds a key-escrow problem. To address this deficiency, functional encryption has been generalized and studied in far more weaker trust models. (See [AGT21] for a detailed discussion on many such generalizations.)

Decentralizing FE. One of the most well adopted such generalizations of FE is multi-authority functional encryption (MA-FE) [Cha07, CC09, LW11, BCG+17]. MA-FE generalizes FE by decentralizing the trust to a group of multiple, *independent and non-interacting*, key authorities. Each key authority generates its own master public-secret key $(\mathsf{mpk}_i, \mathsf{msk}_i)$ *asynchronously* and completely oblivious of other authorities. From an end-user's perspective, MA-FE just looks like a regular (single-authority) FE system with master public key $\mathsf{MPK} = (\mathsf{mpk}_i)_i$, except it receives its decryption key SK_x in the form of n disjoint partial keys $(\mathsf{sk}_{\mathsf{GID},i,x[i]})_i$ from n different key authorities (where each authority just uses its own msk_i and GID is the user's global identifier to tie together partial keys coming from different authorities). In other words, each key authority controls the master key material for only a portion of the full encryption system. This decentralizes trust from one central authority to a group of n individually operating authorities, while GID ensures partial keys issued to the same user by different authorities are "linked".

To model real-world threats while formalizing MA-FE security, the standard approach is to consider two types of corruptions – (1) *partial decryption key corruptions:* here an adversary gets a partial key $\mathsf{sk}_{\mathsf{GID},i,x[i]}$ for some authority i and partial attribute $x[i]$ and identifier GID, and (2) *key authority corruptions:* here an adversary gets a master key msk_i for authority i. Informally, MA-FE security states that an attacker cannot learn anything from a ciphertext ct, encrypting function f, except from what it can learn by legitimately decrypting using valid combinations of the partial decryption keys and authority master keys it has.

What do we know? Although MA-FE was formally introduced almost a decade ago[1] [BCG+17], we have not seen significant progress in terms of new designs for MA-FE. And, this is not due to the lack of community-wide efforts. Since the original proposal of multi-authority attribute-based encryption (MA-ABE) by Chase [Cha07] in 2007, numerous works have studied MA-FE for different classes, but the progress from simple assumptions has either been stuck at MA-ABE for monotone span programs [LW11], or its generalization to inner-product functionality [AGT21]. Moreover, to the best of our knowledge, the only MA-FE construction that we currently have, that goes beyond such attribute-based functionalities, is the original obfuscation based construction by Brakerski et al. [BCG+17], who formally defined and designed MA-FE for arbitrary polynomial-time computations based on *sub-exponentially secure* indistinguishability obfuscation (iO) [GGH+16, SW14] and injective one-way functions.

The current state-of-the-art for MA-FE is embarrassingly dissatisfying! This is unlike (single-authority) FE, where we have seen tremendous progress over the years. Just in the bounded collusion model, we have had multiple FE constructions for general circuits for over a decade [SS10, GVW12, AR17, AV19, AMVY21, GGLW22, GGL24]. And, even in the fully collusion resistant setting, we have numerous advanced systems such as for quadratic functions [Lin17, BCFG17], "degree 2.5" or partially hiding quadratic functions [AJL+19], attribute-based predicates and their variants [GPSW06, GVW13, BGG+14], recently culminating in FE for general circuits from a combination of well-founded assumptions [JLS21, JLS22].

[1] Its predecessor, multi-authority attribute-based encryption (MA-ABE) [Cha07, CC09, LW11], was proposed nearly two decades ago.

Why is MA-FE this hard? Although the lack of progress towards new designs of MA-FE for general circuits suggests an inherent difficulty, there is a deeper underlying barrier that we discuss next. In addition to the MA-FE construction from (sub-exponential) iO, Brakerski et al. [BCG+17] provided a complementary result proving that any secure MA-FE scheme for general circuits implies an obfuscation scheme for general circuits. They proved that the implication to obfuscation follows as long as the MA-FE system is secure when either (i) one key authority gets corrupted, or (ii) two partial decryption keys can be corrupted for every authority. This two-sided implication between MA-FE and obfuscation highlights a major technical barrier.

At this point, it seems quite convincing that breaking new ground in the context of MA-FE suffers from the same barriers that we have for code obfuscation [BGI+12]. Moreover, the implication by Brakerski et al. [BCG+17] follows even when an attacker just learns 2 partial decryption keys per authority (i.e., total of $2n$ partial keys with n authorities). Thus, it appears that even designing MA-FE in the bounded collusion model [DKXY02, SS10, GLW12] faces the same strong barriers as we have for code obfuscation. For example, this points to the impossibility of simulation-secure MA-FE for general circuits in the standard model.

Is this implication tight? Does bounded-collusion MA-FE really give obfuscation? Let us recall the MA-FE-based obfuscation construction by Brakerski et al. [BCG+17]. To obfuscate an n-bit circuit C, an obfuscator samples n MA-FE master key pairs $(\mathsf{mpk}_i, \mathsf{msk}_i)$, and it encrypts the circuit C under the joint master public key $\mathsf{MPK} = (\mathsf{mpk}_i)_i$ to create an encrypted circuit $\mathsf{Enc}(\mathsf{MPK}, C)$. Now, to enable circuit evaluation, the obfuscator generates $2n$ partial decryption keys $(\mathsf{sk}_{\mathsf{GID},i,b})_{i,b}$. That is, for authority i, it generates two partial keys for both attribute bits, 0 and 1. The obfuscated circuit contains the encrypted circuit $\mathsf{Enc}(\mathsf{MPK}, C)$ and $2n$ partial secret keys $(\mathsf{sk}_{\mathsf{GID},i,b})_{i,b}$. An evaluator picks half of the keys $(\mathsf{sk}_{\mathsf{GID},i,x[i]})_i$, depending upon its n-bit input x, and uses them to decrypt the MA-FE ciphertext. Correctness of obfuscation follows from MA-FE correctness, and as long as MA-FE is secure even when two partial keys per key authority can be corrupted, then security of obfuscation follows. Brakerski et al. [BCG+17] used the above argument (and its extensions[2]) to argue necessity of obfuscation.

While this might seem tight, a closer inspection reveals a fundamental issue. The claim that MA-FE is as hard as code obfuscation, even when a bounded number of secret keys get corrupted, is not true! This is because in the above obfuscation construction, the obfuscator generates **two** partial secret keys for the same GID for every authority. In other words, it generates two partial keys for same GID with *distinct* attribute bits. At first this seems a benign thing to do, but this conflicts with the classical motivation and application scenario of MA-FE. As explained by Chase, Chow, Lewko, and Waters [Cha07, CC09, LW11], the notion of a per-user global identifier was introduced to avoid "mix-and-match" attacks. Basically, as the authorities are decentralized and working asynchronously, each authority only gives a partial key for its portion of the attribute, without knowledge of other authorities. Then what prevents two users from combining their partial secret keys!? To avoid this, a popular design strategy is to associate each user with a global (public) identifier GID (which may or may not contain information[3] about its full attribute x). This ensures that only partial keys generated for the same GID can be used together, preventing mix-and-match attacks.

The summary of above discussion is that, in any typical application scenario, an authority does not need to generate *more than one* partial key for a single GID. Thus, it seems most reasonable to consider attackers that receive only **one** partial key per authority for every unique GID. This is the standard approach in the vast MA-ABE literature too [Cha07, CC09, LW11, MJ18, Kim19, WFL19, OT20, DKW21, AG21, AGT21, DKW23, DP23, GGL24]. However, to design obfuscation from MA-FE, we need security when at least **two** partial keys per authority (for some GID) have been corrupted, or an attacker can corrupt key authorities themselves. Due to this mismatch, it is unclear whether the implication from MA-FE to obfuscation still holds when an attacker can corrupt any fixed number of identifiers $(\mathsf{GID}_q)_q$, but only one partial key share fro each authority per GID.

Our results. In this work, we provide new methods to design MA-FE for general circuits from simple and minimal assumptions. All our results are in the bounded-collusion model [SS10, GVW12], naturally generalized to the multi-authority setting [WFL19, GGL24]. We summarize our main results below.

[2]The extension was to consider authority corruptions instead, and that could reduce n, the number of key authorities, to just 1 while maintaining the implication to obfuscation.

[3]The motivation behind not including attribute x in the clear in identifier GID is to get some hiding property about a user's attribute. Since each key authority only learns a portion of the user's attribute along with the public user identifier, thus it provides attribute privacy from a malicious key authority.

1. For all polynomials $n = n(\lambda), Q = Q(\lambda)$, we design a *statically*[4]-secure n-authority MA-FE for general circuits secure, as long as at most Q users get corrupted, under the minimal assumption of PKE.

2. Next, we show two interesting and rather incomparable approaches to boost to full adaptive security.

   (a) First, we show that under the additional assumption of an n-party non-interactive key exchange (niKE) [DH76b, Jou04, BS03, BZ14], we can improve our design to an *adaptively* secure n-authority MA-FE for general circuits with Q-corruptions.

   (b) Second, we show that by assuming sub-exponential hardness of PKE, we can improve our design to an *adaptively* secure n-authority MA-FE for general circuits with Q-corruptions in the Random Oracle Model (ROM) [BR93].

3. Lastly, we also provide a new generic bootstrapping compiler for MA-FE for general circuits. We show that any two adaptively secure n_1-authority and n_2-authority MA-FE schemes for general circuits with Q-corruptions can be generically upgraded to an adaptively secure $(n_1 + n_2)$-authority MA-FE for general circuits with Q-corruptions.

All our MA-FE schemes are proven to be simulation-secure. And, by plugging in 2/3-party key exchange protocols based on DDH hard groups/pairing-friendly groups, we obtain an adaptively secure 2/3-authority MA-FE for general circuits based on DDH or standard bilinear pairing assumptions. Further, by applying our generic bootstrapping compiler, we can design an *adaptively* secure $O(1)$-authority MA-FE for general circuits in the bounded-collusion model, under DDH or standard pairing assumptions. Alternatively, we can also design an *adaptively* secure $\mathsf{poly}(\lambda)$-authority MA-FE for general circuits in the bounded-collusion model from sub-exponential hardness of regular public-key encryption, in the Random Oracle Model. In this work, we only consider MA-FE with honest authorities (i.e., an adversary can only corrupt user secret keys), and leave the problem of designing MA-FE secure against malicious authorities as a major open problem.

Related work. Fully collusion resistant FE with indistinguishability-based security is known to be (nearly) equivalent to iO [GGH+16, AJ15, BV15, AJS15], while simulation-based security is known to make the object impossible [AGVW13]. However, simulation-secure FE with bounded collusion resistance [DKXY02, SS10, GLW12] is achievable [SS10, GVW12, AR17, AV19, AMVY21, GGLW22, GGL24] from minimal assumptions. In the multi-authority setting, numerous constructions for fully collusion resistant MA-ABE have been designed in the past decade; see [Cha07, CC09, LW11, MJ18, Kim19, OT20, DKW21, AG21, AGT21, DP23, DKW23] and the references there in. While in the bounded collusion setting, [WFL19] and [GGL24] have designed MA-ABE for monotone boolean formulae and circuits (respectively) from minimal assumptions.

2 Technical Overview

In this section, we provide a high level overview of our techniques, and summarize the key ideas.

Reviewing MA-FE. A multi-authority functional encryption scheme contains four[5] algorithms – AuthSetup, KeyGen, Enc, Dec. The authority setup algorithm, AuthSetup, is used by an authority to create its public-secret key pair $(\mathsf{mpk}_{\mathsf{id}}, \mathsf{msk}_{\mathsf{id}})$, where id denotes its identity/index. Consider there are n total authorities with public keys $\mathsf{mpk}_1, \ldots, \mathsf{mpk}_n$. All n keys jointly are regarded as the master public key MPK for the full system. The special feature of MA-FE is that any authority can use its secret key $\mathsf{msk}_{\mathsf{id}}$ to create a partial secret key $\mathsf{sk}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}$ for some identifier GID and attribute bit x_{id}[6]. An encryptor takes the full public key MPK and encrypts a circuit C, such that any user with identifier GID and partial keys $\mathsf{sk}_{\mathsf{GID},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{GID},n,x_n}$,

[4] As we discuss later, by static security we mean that an attacker declares all secret key queries at the beginning of the security game. Although our construction is secure in a slightly stronger model (we elaborate in the technical overview), but for simplicity we just state it to be statically secure for the purposes of this introduction.

[5] Technically, MA-FE schemes also have a global setup algorithm that generates global public parameters. Typically, this is just some common random string that can be computed easily in the ROM, or by any party without compromising system security. Thus, we ignore this detail in this overview.

[6] For simplicity, we consider each authority controls a single attribute bit. This can be generically extended to longer attributes.

can decrypt to learn $C(x_1, \ldots, x_n)$. MA-FE schemes are very useful because they do not require interaction between authorities and users to generate keys or compute ciphertexts.

To formally capture adversarial corruptions, one can consider attackers that can corrupt key authorities as well as partial secret keys. However, Brakerski et al. [BCG+17] showed that MA-FE, secure in presence of a single corrupt authority, is as powerful as obfuscation. It is unclear whether MA-FE secure against corrupt authorities is feasible from non-obfuscation assumptions, and we leave this as a very interesting open problem for future research. In this work, we keep our focus on user key corruptions, thus master key authorities are also honest (but mistrusting) in our model.

A bit more formally, we consider attackers that receive master public keys $(\mathsf{mpk}_{\mathsf{id}})_{\mathsf{id}}$ for all authorities, and can make key generation queries to any authority, where it must not submit more than one key query for a given GID to any key authority. That is, the attacker can submit as many key queries as it wants, as long as, for any GID, it does not obtain more than one key per authority[7].

In this work, we consider simulation-based security for MA-FE with bounded collusions. In this setting, the adversary must a-priori commit to a collusion bound Q, where Q denotes the maximum number of unique GID[8] it queries to any single authority in the security game. Throughout the paper, we refer to this as Q-GID corruption model. Simulation security states that no PPT distinguisher can distinguish between a simulated transcript and an honestly generated transcript[9]. And, the simulated transcript must only reveal $\{(\mathsf{GID}, x_{\mathsf{GID}}, C(x_{\mathsf{GID}})) : \text{ for every unique } \mathsf{GID}\}$, where x_{GID} denotes the input string obtained by appropriately concatenating attribute bits queried to each authority for identifier GID.

Organization. Our overview is split into 4 parts. First, we start with designing MA-FE secure against only a single GID corruption. Next, we amplify the collusion bound from 1 to Q GID corruption, for any polynomial Q. However, we are only able to prove security of our (amplified) construction in a weak (static) security model. Next, we develop two separate approaches to boost the security our MA-FE to adaptive security. And, finally, we provide a generic bootstrapping compiler to combine two separate MA-FE systems and turn them into a single bigger MA-FE system.

2.1 Step 1: Adaptively secure MA-FE with 1-GID corruption

We start by designing MA-FE that is secure as long as only 1 key per authority gets corrupted, i.e. in the 1-GID corruption model. To begin, let us keep our focus on proving non-adaptive security, where the attacker has to make all key queries before receiving challenge ciphertext.

Non-adaptive 1-GID MA-FE. Our starting point is the Sahai-Seyalioglu (single-authority) FE construction [SS10]. During setup, one samples $2n$ public-secret key pairs $(\mathsf{PK}_{i,b}, \mathsf{SK}_{i,b})_{i\in[n],b\in\{0,1\}}$. For any input x, the key generator outputs n PKE secret keys, $(\mathsf{SK}_{i,x_i})_{i\in[n]}$ (one for each bit of the input). And, given a circuit C, encryptor garbles C and encrypts $2n$ wire labels appropriately under $\{\mathsf{PK}_{i,b}\}$. Decryption works by first extracting the garbled wire keys, and then evaluate the garbled circuit using them.

Surprisingly, this construction already is multi-authority to begin with. It has an inherent divisibility property, where a secret key for the bit x_i, and encryption for the i-th wire labels are independent of other bits and wire labels. Thus, we can easily transform this into a multi-authority version where each authority samples just two public-secret key pairs, corresponding to the attribute bits 0 and 1. When the authority has to generate secret key for an attribute b, it simply outputs $\mathsf{SK}_{i,b}$. During decryption, all individual keys are combined, and note that secret key from each authority decrypts exactly one wire label, which can be used to evaluate the garbled circuit. The security of this scheme mirrors the security proof of [SS10] construction, given that at most key per authority is corrupted.

However, this is not yet adaptively secure, but only non-adaptively secure. Unless the challenger knows the value of $C(x) = C(x_1, \ldots, x_n)$, we can't leverage the security of the garbling scheme (since an adversary could query keys for different attribute bits x_i adaptively). In order to attain adaptively secure 1-GID MA-FE scheme, we look back at the ideas used in the single-authority setting [GVW12]. The key idea was to use a

[7] As discussed earlier, this closely models the real world scenarios, where any user with GID can only receive a key for a single attribute bit from any authority. This is fairly common approach in prior works [Cha07, CC09, LW11, AGT21].

[8] In our constructions, we denote Q to be the total number of queries the adversary makes across n authorities. However, for the ease of exposition, we denote Q to be the number of unique GIDs the adversary queries throughout the overview. This would only alter the query bound by a factor of n in the actual security game.

[9] By transcript, we mean the full interaction transcript between the challenger and attacker.

non-committing encryption scheme (NCE) to defer the garbling simulation to when the secret key actually gets corrupted. Recall an NCE scheme is a PKE with a special feature that we can "fake" ciphertexts (i.e., encrypt a vacuous message) and later "equivocate" it to be an encryption of any message m of our choice.

Can we use NCE in the multi-authority setting too? As a first attempt, let us naively plug-in NCE in place of PKE in the above construction. While this lets us fake the wire labels for adaptive queries, this is not enough because we don't see the full input x as a whole. Let us elaborate. For any (adaptive) key corruption query, we can fake the encryption of wire labels and reveal them later when we receive the query. However, in order to generate these wire labels by relying on garbling scheme's security, we require the input x as a whole by the time of encryption. Now an attacker could actually ask for half of the partial keys before challenge ciphertext and half of them after. This is why naively using NCE is not enough.

Layering multiple NCE. In order to resolve this, we employ a new approach of composing NCE in the multi-authority setting. Let ct be the ciphertext of non-adaptive 1-GID MA-FE as described above. We sample n random strings $R_1, \ldots, R_n$ (of the same length as ct), and set the new ciphertext as $(\mathsf{NCE}_1.\mathsf{Enc}(R_1), \ldots, \mathsf{NCE}_n.\mathsf{Enc}(R_n), \mathsf{ct} \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}})$. This can be simply viewed as using a one-time pad to mask the actual ciphertext, and additively secret share the mask across NCE. Naturally, during key generation, each authority additionally provides a secret key for $\mathsf{NCE}_{\mathsf{id}}$. We show the above is sufficient to prove adaptive security. Basically, the idea is to wait until the last (partial) secret key is corrupted, and only then 'equivocate' the ciphertext to put in the right non-adaptive ciphertext. More details are provided later in the main body Section 5.

2.2 Step 2: Amplifying to Q-GID corruptions with *static* security

Our next step is to amplify the collusion bound from 1 to any (a-priori bounded) polynomial Q. We start by reviewing the collusion amplification techniques developed in the single-authority setting [GVW12, AV19]. (Readers familiar with prior techniques can safely skip Section 2.2.1 and move to Section 2.2.2.)

2.2.1 Recapping single-authority collusion amplification techniques

Intrinsically, [GVW12, AV19] amplify collusion bound by designing a bounded FE scheme (BFE) secure against Q collusions by combining $\mathsf{poly}(\lambda, Q)$ instantiations of an FE system (1FE) secure against a 'single' collusion. The core idea is to take randomized encodings [AIK06] and compile them via specific multi-party computation (MPC) protocols [BGW88, BMR90]. Using these two tools, one can tie together $\mathsf{poly}(\lambda, Q)$ (parallel) instances of 1FE such that any successful attacker on BFE can be reduced to a successful attacker on at least one instance of 1FE. The MPC protocol is built using special randomized encodings that ensure there is enough redundancy such that the protocol remains secure, even if a small subset of the clients are corrupted. And, by ensuring this protocol has just two rounds (and is stateless), we can run it in-the-head and use each round's messages during encryption/key-generation to construct BFE. Let us elaborate.

The client-server framework. [AV19] proposed a new MPC framework, called client-server framework (CSF). It neatly packs all the technical complications that arise while designing BFE. In this framework[10], there is a *server* that possesses Q inputs $(x_1, \ldots, x_Q)$ and a *client* with a circuit C. They want to jointly compute $C(x_1), \ldots, C(x_Q)$, with the condition that no information about C should be leaked to the server apart from $\{(x_q, C(x_q))\}_q$. Moreover, the server and client have access to a set of N *users* to assist in this task. The client and server perform Q rounds of computation, where they delegate some computation to these users in each round. The protocol has three phases:

Offline Phase: The client gets Q and encodes its circuit C into N shares, $(\widehat{C}^1, \ldots, \widehat{C}^N)$, $\widehat{C}^i$ for user i. This is captured using the ClientEnc procedure.

Online Phase: This phase is executed for Q rounds. In each round, the server encodes the q-th input x_q into $(\widehat{x}_q^1, \ldots, \widehat{x}_q^N)$ using ServEnc procedure. (Here $\widehat{x}_q^i$ is intended for user i.) However, in each round, all the users are not 'activated'. That is, only a subset $\mathbf{S}_q \subset [N]$ of users are invoked in the q-th round. Each user $i \in \mathbf{S}_q$ computes $\widehat{y}_q^i \leftarrow \mathsf{UserComp}(\widehat{x}_q^i, \widehat{C}^i)$ to obtain the i-th output encoding.

[10] For ease of exposition of our main technical ideas, our CSF abstraction slightly deviates from [AV19]. For e.g., we consider a setting in which servers have inputs, and client has a circuit.

Decoding Phase: This phase is also executed for Q rounds. Its purpose is to combine the output encodings computed by the specific subset of users, $\mathbf{S}_q$, in the corresponding round. That is, by running $\mathsf{Decode}(\mathbf{S}_q, \{\widehat{y}_q^u\}_{u\in\mathbf{S}_q})$, one gets $y_q = C(x_q)$.

The notion of CSF security is heavily tailored for BFE, where the goal is to resist attackers that can corrupt a subset of 'users' as well as a certain number of server and output encodings. More formally, adversary sends a single input x, and then receives encodings for a subset of users, $\mathbf{S}_q$, for each round q adaptively. If a user u is contained in more than one subset, then its full client encoding $\widehat{C}^u$ is revealed to the adversary. Else, only the server and output encoding for u, in that particular round, is revealed to the attacker. Overall, none of this should reveal any more information about x beyond $\{(x_q, C(x_q))\}_q$.

The BFE blueprint. Given a CSF with above guarantee, it is natural to use it to design a BFE. Basically, BFE uses CSF to implement N copies of a 1FE system. An encryptor encodes the input circuit C using the offline ClientEnc procedure, and encrypts each encoding under a separate 1FE system. Thus, the BFE ciphertext contains N 1FE ciphertexts. To enable functional decryption, a key generator (who has master keys for all N 1FE systems) starts by running the online ServEnc on the desired input x, samples a random subset $\mathbf{S} \subset [N]$, and for each $i \in \mathbf{S}$, it generates a key for $\mathsf{UserComp}(\widehat{x}^i, \cdot)$ using the i-th 1FE system. These $|\mathbf{S}|$ 1FE secret keys correspond to a single BFE key for input x. One can naturally design a decryption procedure by combining the 1FE decryption algorithm with CSF Decode. We refer to this as the BFE blueprint. In order to prove security of the above BFE scheme, we need to rely on CSF security very carefully. Moreover, multiple (implicit) parameters have to be very carefully defined and analyzed to argue sufficient redundancy. To explain these parameters, we summarize the high-level (state-of-the-art) schema for designing CSF.

Gorbunov-Vaikuntanathan-Wee (GVW) [GVW12] designed CSF for all NC^1 circuits[11], which was later improved to all poly-size circuits by Ananth-Vaikuntanathan [AV19]. The core idea was to use Shamir's secret sharing [Sha79] to share the circuit C. And, by using standard polynomial interpolation techniques, each 'user' can individually evaluate the share, while a certain threshold of evaluated shares can be combined to reconstruct the circuit output. To guarantee decodability, the secret sharing threshold had to be set such that it was less than the number of users activated in each CSF round, $|\mathbf{S}_q|$.

Proving security of the above template requires a lot of care. First, we need a strict upper bound on the total number of 'users' that get corrupted throughout the entire lifetime (i.e., across Q rounds). If t is the Shamir's sharing threshold, then we need that $\cup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'} < t$. ($\mathbf{S}_q$ is the random set of users selected in q-th round.) With this goal, GVW used a statistical lemma, called *small pairwise intersection lemma*. It states that if number of rounds Q is known in advance, then by setting (N, t) as appropriate polynomials of Q, λ, we can guarantee (with $1 - \mathsf{negl}$ probability) the above 'user' corruption bound does not get violated.

However, setting the parameters as above is still not sufficient to prove CSF security. The reason is an attacker also learns server and output encodings for each "honest" user activated in every round. Shamir's secret sharing of C is done only once during the offline phase, thus, without additional *per-round* re-randomization, the secret sharing guarantee cannot be used to hide C even when the small pairwise intersection guarantee holds. To fix this, GVW explicitly re-randomized the output encodings during each round. The first change to the above CSF schema is to also add T secret shares of 0 as part of the input encoding (during ClientEnc procedure). Next, additionally sample another random set Δ_q during each online round, and include Δ_q as part of the circuit encoding created by ServEnc. Δ_q decides which subset of T shares will be added as re-randomization in the q-th round. To finish everything off, GVW proved another statistical lemma, called *cover-freeness lemma*. It guarantees that (again, with $1 - \mathsf{negl}$ probability) if Q is known, then $|\Delta_q|$ and other parameters can set such that the honest user's encodings are re-randomized by at least one fresh secret share of 0 in every round.

Combining the above two statistical lemmas and the expanded CSF schema, the BFE blueprint has been used to amplify 1FE to Q-collusions in the single-authority setting [GVW12, AV19]. Going forward, our plan is to design and execute a similar template in the multi-authority setting.

2.2.2 Collusion amplification in MA-FE

Our strategy is to generalize the BFE blueprint to design Q-GID MA-FE, however several technical hurdles arise as explain next. In one sentence, all technical issues can be majorly attributed to the *fully-distributed*

[11] Although [GVW12] did not present it as such, one can abstractly view their work as designing CSF too.

key generation functionality of MA-FE. Note that there are multiple, *non-interacting*, key-authorities in the multi-authority setting, and these authorities must generate their master keys as well as *partial* secret keys *asynchronously*. This total asynchronicity (i.e., lack of communication between authorities) is a major technical bottleneck. We start by appropriately generalizing the (single-authority) BFE blueprint to the multi-authority regime, ignoring the above implicit technical challenges.

A *distributed* client-server framework. Our first step is to define and design a "distributional" variant of CSF, that we call dCSF. The purpose is to support n independent servers that can come up with a valid (distributed) server encoding for an input x. Its syntax can be appropriately generalized from the single-authority setting. Recall that the server encoding, in CSF, just looks like (x, Δ). Now in the multi-authority setting, each server only receives a portion of the input x. (In MA-FE, each authority only has $x[i]$ rather than full x.) So, first question is what to replace x with in the server encodings? The answer is simple. There is not really any cryptographic encoding being performed during ServEnc, so we could switch x with $x[i]$, in each individual server's encoding. However, there is another term that we have to worry about, Δ. Recall that the purpose of Δ is to select a subset of *re-randomizing zero shares* to make evaluated shares in every round look like fresh output shares.

Now this brings us to the first technical hurdle: *how to sample the cover-free set Δ asynchronously?* One could consider either: the first server samples it for everyone, or every server samples a *portion* of Δ such that these can be jointly stitched together to obtain the 'full' set. However, none of these would work! It is essential that Δ is selected all at once, and more importantly, the choice of Δ should not be corruptible at all! Otherwise, the cover-freeness lemma fails. Our strategy for resolving this is to use an external source that can sample Δ for all authorities jointly, with the guarantee that it looks truly random and is not corruptible/bias-able. For now, we skip discussing this technical detail and continue our process of generalizing the BFE blueprint. We will circle back to it at a more appropriate time.

Moving on to the *online phase* of the dCSF protocol, we need to appropriately generalize the CSF protocol. Since the subset of user, $\mathbf{S} \subset [N]$, that have to be activated in this round must be selected randomly and asynchronously. Thus, as before, we face the same barrier which is: how to select this set with *zero* interaction between servers? Our plan is to use the same external source, to assist in sampling $\mathbf{S}$ for each round, that we used for sampling Δ for the particular round. Assuming we are able to handle the sampling $(\mathbf{S}, \Delta)$-problem, it appears that distributing the CSF framework is a feasible strategy. Moreover, by opening up the prior CSF constructions [GVW12, AV19], we realize that all the underlying cryptographic tools can be as easily generalized. Basically, we can construct our dCSF using a (weaker) variation of CorrGarb as introduced in [AV19]. Next, our hope is that such a dCSF scheme, coupled with our 1-GID MA-FE scheme, we can rely on ideas similar to the classic BFE blueprint to design a Q-GID MA-FE scheme. In light of above, let us further inspect this $(\mathbf{S}, \Delta)$-sampling problem.

The MA-FE sampling bottleneck and static security. Let us start by sketching our candidate design for Q-GID MA-FE based on adaptive 1-GID MA-FE scheme (1MAFE) and a dCSF protocol.

Authority Setup: Sample N 1MAFE instantiations.

Encryption: Generate client encodings of C and encrypt the u-th user computation circuit with the u-th client encoding hardwired using the u-th 1MAFE instantiation. Output all N ciphertexts.

Key Generation: For the id-th authority with input $\mathsf{GID}, x_{\mathsf{id}}$, select random $(\mathbf{S}, \Delta)$ and generate the id-th server encodings and keys for u-th server encoding with the u-th 1MAFE instantiation. Output $\mathbf{S}$, secret keys for $u \in \mathbf{S}$.

The above template exactly resembles the [AV19] template for single-authority FE, but with one major (underlined) distinction. How are $(\mathbf{S}, \Delta)$ selected by different key authorities for the same user (i.e., for a fixed GID). This is a significant technical hurdle, and it is a fundamental problem that does not exist in the single-authority setting! At this point, we need to deviate from prior works and need to develop new techniques specific to multi-authority regime.

Our starting point is to restrict ourselves to a very weak security model, that we refer to as *static security* for MA-FE. Roughly, an attacker in this model *must* declare all key queries it wants to make to all the key authorities at the beginning of the game. That is, the attacker statically provides everything to the challenger, and receives all secret keys as well as the challenge ciphertext, and must bypass MA-FE security.

Our core observation is that if an attacker is fully static, then we can simply use a PRF to sample $(\mathbf{S}, \Delta)$ non-interactively and asynchronously for every user. Let us elaborate.

From PRFs to static security. Suppose we have a trusted party that samples a crs for all authorities to use. In this case, our plan to sample $(\mathbf{S}, \Delta)$ is simple. Simply set it as $F_{\mathsf{crs}}(\mathsf{GID})$. That is, view crs as a PRF key, and evaluate it on GID (as an input) to sample user-specific random sets. Clearly, these will be random (per user) and can be sampled asynchronously, and crucially, they will be *consistent*. Now one might wonder having a crs is undesirable in the multi-authority setting. However, this is a mild assumption made in many prior works. Moreover, we can avoid it as well, by simpliy considering that each authority samples its own PRF key, publishes it as part of its master public key, and then all key authorities could jointly consider the XOR of all individual keys as the full system's PRF key.

At a high level, the above seems like a reasonable strategy for resolving the sampling problem, but it is unclear why/how could we use the PRF security at all. Clearly, this key is either part of the crs or $\{\mathsf{mpk}_i\}_i$ (i.e., public in both cases). For relying PRF security, we need the key to be hidden. While this might seem like a big problem, we highlight that this is exactly why we set our goal to be *static security*. Note that in static model, the attacker must declare everything (key queries, challenge messages, etc) at the beginning. Thus, we could simply rely on statistical security of PRF outputs to argue that its output can **not** be too correlated, as otherwise it would contradict pseudorandomness.

We formalize the above argument, and prove stronger variations of the two underlying statistical lemmas (small pairwise intersection and cover-freeness). We show that the desired statistical properties still hold even when $(\mathbf{S}, \Delta)$ are pseudorandomly selected, and are not truly random (unlike prior works [GVW12, AV19]). We refer to Section 7 for more details. In summary, our core idea is to generalize all underlying tools used in the single-authority setting to the multi-authority setting, and tie them together with a single PRF to assist in joint random set sampling, while proving security in a weak security model.

Next, we develop better strategies for resolving the aforementioned sampling problem, with the goal of proving full (adaptive) security.

2.3 Step 3: Towards Adaptive Security

In this section, we develop two varying approaches to design an adaptively secure MA-FE scheme with diverging features. In a few words, our idea is to fix a global random function from GID to $(\mathbf{S}, \Delta)$ as part of the joint master public keys of all authorities. That is, we plan to define $(\mathbf{S}, \Delta) := \mathsf{Func}^*(\mathsf{GID}, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$, where n is the number of authorities and Func^* is a (possibly) non-deterministic function. And, given such a function, we plan to give each authority some special "advice" to help evaluate this random function *without interacting with other authorities*. Basically, this will ensure that an attacker cannot either evaluate the function Func^*, or gain any advantage by learning this mapping on any polynomial number of GID values.

Approach 1: Non-Interactive Key Exchange

Recall that in the static construction, we required all queries to be provided statically since we couldn't rely on PRF security if the key is revealed to the adversary. *What if an attacker never learns the* PRF *key, but authorities do!?* This would ensure that the sampling problem goes away, and more importantly, we can rely on PRF pseudorandomness. The next question is do we any cryptographic object with similar guarantees. The answer is unsurprisingly yes, with the object being a non-interactive key exchange protocol[12] (niKE) for n parties [DH76b, Jou04, BS03]. A niKE scheme for n parties allows a group of n parties to sample a shared random secret by simply publishing a global public key per user, where any attacker that only learns the public keys must not be able to learn anything about the shared secret.

Given niKE, our strategy is to make each authority sample a niKE public key and store as part of its master public key (during authority setup). With this modification, each authority can first generate the shared secret key K during key generation, and then use it to sample the random sets as $(\mathbf{S}, \Delta) := F_K(\mathsf{GID})$. Clearly, an attacker cannot learn K (since it never corrupts authorities in our security model), thus we can still rely on underlying statistical lemmas for ensuring there is sufficient redundancy in the system. We provide more details in Section 8. It is well known that 2/3-party key exchange can de designed from simple algebraic assumptions [DH76b, Jou04]. Thus, this gives us an n-authority MA-FE (for $n = 2/3$) from

[12] As we do not consider authority corruptions, we only require a statically secure scheme without corruptions.

simple assumptions with full adaptive security. We view usage of niKE as a new interesting technique in the functional encryption space (beyond being used directly for all-or-nothing public key encryption), and we think that this could open interesting directions for further research in the multi-authority setting. To the best of our knowledge, this is the first time an niKE scheme is directly used in the context of bounded collusion functional encryption.

Approach 2: Random Oracles & Complexity Leveraging

Our second solution for resolving the sampling problem is to rely on (non-programmable) Random Oracles [BR93]. Basically, our plan is to use an explicit hash function $\mathcal{H}$ to sample $(\mathbf{S}, \Delta)$ as $\mathcal{H}(\mathsf{GID})$, but model it as a random oracle in the security proof. The advantage of doing this is that we can use the fact that a random oracle can simply act as a PRF that gives pseudorandomness, even when there are no secret keys. That is, it is a source of unbounded 'true' randomness. While this seems like a natural idea, and it appears that this should directly suffice for proving full security of our statically-secure MA-FE scheme, there are some very subtle barriers which prevent the proof going through seamlessly. Let us elaborate.

The issue really is although we model $\mathcal{H}$ as a random oracle, an attacker gets unbounded access to $\mathcal{H}$. That is, the attacker can learn unbounded number of mappings (say, $P \geq Q$) between GID and their corresponding $(\mathbf{S}, \Delta)$ values. This is big red flag and a major issue because, in the bounded collusion setting, we only want an attacker to learn a bounded number of $(\mathbf{S}, \Delta)$ values. Recall in the single-authority setting, $(\mathbf{S}, \Delta)$ was sampled by the key generator, thus an attacker only ever learns Q such values. (Here Q is the collusion bound.) Whereas in our construction, the attacker can learn an unbounded number of these values. One might wonder can we define a notion of bounded number of random oracle queries, but any such definition would be utterly meaningless! Thus, we really need to find a better strategy to prove security despite the fact that an attacker can learn an unbounded number of such GID-set mappings.

Looking closely, our main observation here is to rely on better concentration bounds and prove both statistical lemmas much more tightly. Crudely, we notice that if we increase the underlying parameters (used for small pairwise intersection and cover-freeness) by a factor of Q, then this is sufficient to get similar statistical guarantees. The idea is that, if in the Chernoff's bound argument of these lemmas, we increase the exponent by a factor of Q, then by a union bound on the number of subsets of P of size Q, these lemmas still hold[13]. We refer the reader to Section 4 for a detailed overview and proof of these augmented lemmas. With the above change, we have bypassed the issue due to unbounded access to the adversary of the GID-set mappings. Unfortunately, this on its own is still not enough to prove adaptive security.

Now that we do not have any PRF key K and $\mathcal{H}$ is modeled as a random oracle, there is no need for the adversary to make the queries in a static manner. But, it turns out that this *does not* satisfy adaptive security. In order to understand why, we need to open up the prior security reductions [GVW12, AV19] a bit more.

Inherently, all prior (collusion amplification) compilers require the reduction algorithm to be aware of all $\{(\mathbf{S}_j, \Delta_j)\}_j$ sets selectively (i.e., at the start of the game). If this is not available, then the entire simulation strategy breaks apart. Now in the single-authority setting, this is not a problem since an adversary only learns these values when it queries for keys. Thus, a reduction algorithm can simply sample them in advance and use later as and when needed. However, we can't do the same in the above ROM-based design. This is because an attacker can simply make a very large number of ROM queries (say Q^2) before it makes any of its Q-GID queries. Now a reduction algorithm must answer all these ROM queries, but it does not know which Q of these will actually be used for generating secret keys (i.e., which of these will actually be key queried upon).

Partial adaptive MA-FE in ROM. This technical difference prohibits an adaptive proof of security even in the ROM. However, we notice that using ROM has still a clear advantage, which is we could prove slightly stronger security than static-Q-GID security in ROM. Consider an attacker that can make all key queries adaptively, but all it has to *selectively* commit to is all the GIDs that it will every request secret keys on during the entire experiment. We refer this as the *partial adaptive* model. And, we are able to prove that the above ROM-based design does indeed provide security against all partial adaptive adversaries. The core of this argument is that, for partial adaptive attackers, we can sample the responses to all Q GID queries (as ROM outputs) at the beginning and determine the set of non-corrupted users which is required to argue the

[13] As the number of such subsets are bounded by P^Q.

security of dCSF. Interestingly, here we are able to handle any $n = \mathsf{poly}(\lambda)$ nuber of authorities, and only require the minimal assumption of public-key encryption, although in ROM. We refer the reader to Section 10 for more details.

Final step: Complexity leveraging for adaptive security. Note that partial adaptive security is quite artificial, and still very far from adaptive security. However, the only real difference is that we want the reduction to know which GIDs will be ever corrupted at the beginning. Thus, we ask the question – *Can we just guess these* GID*s and consider it as a security loss in the reduction advantage?* Since $|\mathsf{GID}| = \mathsf{poly}(\lambda)$ (i.e., its length is some polynomial), thus if we guess all the Q GID strings that we will be corrupted, then we will incur an exponential loss (in Q, λ). While this is not ideal, we should be able to rely on sub-exponential security (of the underlying primitives) and prove adaptive security by appropriately setting λ [BB04].

This is the final strategy that we employ, where we design an adaptively secure Q-GID MA-FE scheme assuming the sub-exponential security of public-key encryption in ROM. For inspecting the exact parameter setting, we refer the reader to Section 11. We leave the problem of relying only on polynomial hardness of PKE (even in the ROM) as a very interesting open problem.

We remark that although the above appears to be a simple application of complexity leveraging, there are significant differences due to which the full security analysis is highly non-trivial and requires many heavy techniques. Briefly, the issue is that as a reduction algorithm, we cannot just use a fully adaptive adversary to break the underlying partial adaptive security. This will be an incorrect argument, as the adversary's behavior could be correlated with the random selections made by a reduction algorithm at the beginning. Thus, when the attacker's choice of GID queries match the reduction algorithm's guess, then there could be negative correlation between the reduction's success and the adversary's success. Similar (negative association) issues were first observed by Waters [Wat05] in the context of Identity-Based Encryption. For which, Waters developed a novel "artificial abort" technique to handle such negative correlations. In this work, we borrow ideas from the *artificial abort* technique, and use it to prove adaptive security of our construction. Our exact proof template is inspired by the *advantage counting variation of the artificial abort technique* developed in [FGH+17]. While artificial abort has been used numerous times in proving fully collusion resistant security, our application of artificial aborts in the context of bounded collusion security and multi-authority functional encryption appears to be a first (to the best of our knowledge). We believe future works might also benefit from our analysis and application of artificial abort.

2.4 Step 4: Bootstrapping Compiler for Q-GID MA-FE

Finally, we also develop a new compiler for bootstrapping two Q-GID MA-FE schemes (generically) into an MA-FE scheme supporting a larger number of authorities. While we believe this compiler could be of independent interest beyond our work, we show that this can be readily used to bootstrap our 2/3-party niKE-based MA-FE supporting 2/3 number of authorities to an MA-FE scheme that supports any constant number of authorities. That is, under DDH/BDDH, we obtain an adaptively-secure $O(1)$-authority Q-GID MA-FE scheme for P/Poly circuits (without using random oracles). Comparing this with our ROM-based MA-FE scheme which requires sub-exponential hardness, this only requires polynomial hardness of the underlying assumptions and is in the standard model. We leave designing an MA-FE scheme in the standard model beyond $O(1)$ authorities as a very interesting open problem. Let us next describe our bootstrapping compiler.

Circuit composition for bootstrapping MA-FE. The main idea of the compiler can be understood as follows. Given a $2n$-ary circuit C, one can construct an n-ary circuit F_C such that $F_C(x_{n+1}, \ldots, x_{2n}) = C(\cdot, \ldots, \cdot, x_{n+1}, \ldots, x_{2n})$. That is, F_C is an n-ary P/Poly circuit that takes the last n inputs meant for C and outputs the description of an n-ary circuit which is nothing but C with the last n inputs hardwired inside it. So, given a $2n$-ary circuit, we can split it "recursively" into two n-ary circuits. We can simply encrypt F_C with the second instantiation of n-authority MA-FE. At a more technical level, we modify F_C so that it outputs the encryption of the n-ary version of C under the first n-authority MA-FE instantiation. We refer the reader to Section 9 for more details. In summary, we transform C into F_C and encrypt F_C under the second n-authority instantiation where F_C recursively calls the first n-authority instantiation's encryption algorithm. The adaptive simulation security of the resulting scheme can be appropriately reduced to the adaptive simulation security of the underlying two instantiations. This is because once we rely on the security of the second instantiation, we can internally rely on the security of the first instantiation.

While at first, it might appear that this can be used to design an n-authority MA-FE scheme for any $n = \mathsf{poly}(\lambda)$, by running the bootstrapping compiler $\log n$ times. Unfortunately, this is not the case! The issue is that with d layers of recursive construction, size of F_C could grow potentially as large as $\mathsf{poly}(\lambda)^d$. This is because size of F_C is at least the size of the encryption circuit of first n-authority MA-FE scheme. We are encrypting F_C again. Hence, any non-linear circuit size will result in exponential blow-up in encryption circuit's size making it too inefficient to use it for more than constant many layers of combination.

For more details, please refer to Section 9. We conclude by stating that, to the best of our knowledge, this is the first bootstrapping compiler for any type of multi-authority functional encryption system. We think our bootstrapping compiler might be of independent interest, and could be useful for future research on MA-FE. Lastly, we leave the problem of designing a compiler that can bootstrap to support a super-constant number of authorities as a very interesting open question. This will readily lead to $\mathsf{poly}(\lambda)$-authority Q-GID MA-FE scheme for P/Poly circuits under polynomially hard standard assumptions by relying on our results.

3 Preliminaries

In this section, we provide the notation and definitions for all the primitives we use to construct multi-authority functional encryption schemes.

3.1 Notation

We denote the security parameter by λ. Let PPT denote probabilistic polynomial time. For any $n \in \mathbb{N}$, we denote by $[n]$, the set of all positive integers up to n, that is, $\{1, \ldots, n\}$. For any $a, b \in \mathbb{N}, a \leq b$, 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 $x \xleftarrow{\$} \mathbf{X}$, the process of sampling an element x from the set $\mathbf{X}$, 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 $2^{\mathbf{A}}\big|_a$, we denote the set of subsets of $\mathbf{A}$ with size a. By $\mathsf{negl}(\lambda)$, we define negligible functions. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is a negligible function if for every $c \in \mathbb{N}$ and for large enough λ, $\mathsf{negl}(\lambda) < n^{-c}$.

We say that two probability distributions $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$ with support Ω are ϵ-computationally indistinguishable if for any polynomial-size distinguisher family $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$, and for large enough $\lambda \in \mathbb{N}$,

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

We write $\mathcal{X} \approx_c^\epsilon \mathcal{Y}$ to denote that distributions $\mathcal{X}$ and $\mathcal{Y}$ are ϵ-computationally indistinguishable. If $\epsilon = \mathsf{negl}(\lambda)$, we drop the superscript and write $\mathcal{X} \approx_c \mathcal{Y}$. Similarly, by $\approx_s^\epsilon$, we denote ϵ-statistical indistinguishability. If $\epsilon = \mathsf{negl}(\lambda)$ we write $\mathcal{X} \approx_s \mathcal{Y}$. Concretely, $\mathcal{X} \approx_s^\epsilon \mathcal{Y}$ if for large enough $\lambda \in \mathbb{N}$,

$$\frac{1}{2} \sum_{\alpha \in \Omega} \left| \Pr\left[\alpha \xleftarrow{\$} \mathcal{X}_\lambda\right] - \Pr\left[\alpha \xleftarrow{\$} \mathcal{Y}_\lambda\right] \right| \leq \epsilon(\lambda)$$

3.2 Pseudorandom Functions

A secure pseudorandom function (PRF) satisfies the following properties.

Definition 3.1. *A pseudorandom function* PRF *is a function that takes inputs from the domain* $\mathbf{X} = \{\mathbf{X}_\lambda\}_{\lambda \in \mathbb{N}}$ *and samples a* PRF *seed* K *such that* $K \in \{0,1\}^\lambda$ *and outputs values in the range* $\mathbf{R} = \{\mathbf{R}_\lambda\}_{\lambda \in \mathbb{N}}$. *We say that the* PRF *is a secure pseudorandom function if it satisfies the following properties.*

1. ***Efficiency:*** *For any* $K \in \{0,1\}^\lambda, x \in \mathbf{X}_\lambda$, $\mathsf{PRF}(K, x)$ *runs in deterministic polynomial time.*
2. ***Pseudorandomness:*** *We say that* PRF *is secure if for any stateful PPT adversary* $\mathcal{A}$,

$$\Pr\left[b = b' : \begin{array}{c} b \xleftarrow{\$} \{0,1\}, K \xleftarrow{\$} \{0,1\}^\lambda, \\ b' \leftarrow \mathcal{A}^{\mathcal{O}_b(\cdot)}(1^\lambda) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

where $\mathcal{O}_0(\cdot) = \mathsf{PRF}(K, \cdot)$ *and* $\mathcal{O}_1(\cdot)$ *generates random strings in* $\mathbf{R}_\lambda$.

3.3 Statically Secure n-Party Non-Interactive Key Exchange Scheme

A statically secure n-party non-interactive key exchange scheme (niKE) allows for n independent parties to derive a common key K in the crs model. The common key depends on the public values of the parties and secret value of a certain party. Any adversary without a valid secret key, cannot distinguish between a common key and a randomly generated key. A niKE scheme consists of the following algorithms.

$\underline{\mathsf{Setup}(1^\lambda) \to \mathsf{crs}}$: This is a randomized algorithm that on input the security parameter λ outputs the common random string crs. The following algorithms take crs as an implicit input.

$\underline{\mathsf{Publish}(\mathsf{id}) \to (\mathsf{pv}, \mathsf{sv})}$: This is a randomized algorithm that on input the authority identifier id, output the public and secret values for the authority, $(\mathsf{pv}, \mathsf{sv})$.

$\underline{\mathsf{KeyGen}(\{\mathsf{pv}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \mathsf{id}^*, \mathsf{sv}_{\mathsf{id}^*}) \to K}$: This is a polynomial time algorithm that on input the public values for all the authorities $\{\mathsf{pv}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$, an authority identifier id^*, and the corresponding secret value for the authority $\mathsf{sv}_{\mathsf{id}^*}$, outputs the key K.

Definition 3.2. *The scheme* $\mathsf{niKE} = (\mathsf{Setup}, \mathsf{Publish}, \mathsf{KeyGen})$ *is said to be a statically secure* n*-party non-interactive key exchange scheme if it satisfies the following properties.*

1. ***Correctness:*** *We say that* niKE *is correct if for any* $\lambda \in \mathbb{N}$,

$$\Pr\left[K^* = K' : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^n), \\ \forall\ \mathsf{id} \in [n], (\mathsf{pv}_{\mathsf{id}}, \mathsf{sv}_{\mathsf{id}}) \leftarrow \mathsf{Publish}(\mathsf{id}), \\ \forall\ \mathsf{id}^*, \mathsf{id}' \in [n], \mathsf{id}^* \neq \mathsf{id}', \\ K^* = \mathsf{KeyGen}(\{\mathsf{pv}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \mathsf{id}^*, \mathsf{sv}_{\mathsf{id}^*}), \\ K' = \mathsf{KeyGen}(\{\mathsf{pv}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \mathsf{id}', \mathsf{sv}_{\mathsf{id}'}), \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

2. ***Static Security:*** *We say that* niKE *is statically secure if for any PPT adversary* $\mathcal{A}$,

$$\Pr\left[b = b' : \begin{array}{c} b \xleftarrow{\$} \{0,1\}, \\ \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda), \mathsf{id}^* \xleftarrow{\$} [n], \\ \forall\ \mathsf{id} \in [n], (\mathsf{pv}_{\mathsf{id}}, \mathsf{sv}_{\mathsf{id}}) \leftarrow \mathsf{Publish}(\mathsf{id}), \\ K^{(0)} = \mathsf{KeyGen}(\{\mathsf{pv}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \mathsf{id}^*, \mathsf{sv}_{\mathsf{id}^*}), \\ K^{(1)} \leftarrow \{0,1\}^\lambda, \\ b' \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}, \{\mathsf{pv}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, K^{(b)}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

3.4 Non-Committing Encryption

A non-committing encryption scheme (NCE) is a public-key encryption scheme with an additional property that we can later reveal a ciphertext to be the encryption of any message by tailoring the secret key for the scheme accordingly. A NCE scheme consists of the following algorithms.

$\underline{\mathsf{Setup}(1^\lambda, 1^l) \to (\mathsf{MPK}, \mathsf{MSK})}$: This is a randomized algorithm that on input the security parameter λ and the maximum length of the messages l, outputs the master public and secret key pair $(\mathsf{MPK}, \mathsf{MSK})$.

$\underline{\mathsf{KeyGen}(\mathsf{MSK}) \to \mathsf{SK}}$: This is a randomized algorithm that on input the master secret key MSK, outputs a secret key SK.

$\underline{\mathsf{Enc}(\mathsf{MPK}, m) \to \mathsf{CT}}$: This is a randomized algorithm that on input the master public key MPK and a message $m \in \{0,1\}^l$, outputs the corresponding ciphertext CT.

$\underline{\mathsf{Dec}(\mathsf{SK}, \mathsf{CT}) \to m'}$: This is a polynomial time algorithm that on input the secret key SK and a ciphertext ct, outputs m'.

$\underline{\mathsf{Fake}(\mathsf{MPK}) \to (\widetilde{\mathsf{CT}}, \mathsf{aux})}$: This is a randomized algorithm that on input the master public key MPK, outputs a fake ciphertext $\widetilde{\mathsf{CT}}$ and auxiliary information aux.

$\underline{\mathsf{Reveal}(\mathsf{MSK}, \mathsf{aux}, m) \to \widetilde{\mathsf{SK}}}$: This is a polynomial time algorithm that on input the master secret key MSK, the auxiliary information aux, and the message m, outputs a secret key $\widetilde{\mathsf{SK}}$.

Definition 3.3. *The scheme* $\mathsf{NCE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Fake}, \mathsf{Reveal})$ *is said to be an adaptively-secure non-committing encryption scheme if it satisfies the following properties.*

1. ***Correctness:*** *We say that* NCE *is correct if for any* $\lambda \in \mathbb{N}, l = l(\lambda)$*, and any* $m \in \{0,1\}^l$*,*

$$\Pr\left[m = m' : \begin{array}{c}(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^l), \\ \mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{MPK}, m), \mathsf{SK} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}), \\ m' \leftarrow \mathsf{Dec}(\mathsf{SK}, \mathsf{CT})\end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

2. ***Adaptive Security:*** *We say that* NCE *is secure if for any stateful PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$*,*

$$\Pr\left[b = b' : \begin{array}{c}b \xleftarrow{\$} \{0,1\}, \\ (\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{Setup}(1^\lambda, 1^l), \\ (m, \mathsf{state}) \leftarrow \mathcal{A}_0(1^\lambda, \mathsf{MPK}), \\ \mathsf{CT}^{(0)} \leftarrow \mathsf{Enc}(\mathsf{MPK}, m), (\mathsf{CT}^{(1)}, \mathsf{aux}) \leftarrow \mathsf{Fake}(\mathsf{MPK}), \\ \mathsf{SK}^{(0)} \leftarrow \mathsf{KeyGen}(\mathsf{MSK}), \mathsf{SK}^{(1)} \leftarrow \mathsf{Reveal}(\mathsf{MSK}, \mathsf{aux}, m), \\ b' \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{CT}^{(b)}, \mathsf{SK}^{(b)})\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

A non-committing encryption which obeys the above definitions can be constructed using a public-key encryption scheme as seen in [GVW12, HMNY22, HKM+23].

3.5 Correlated Garbling

A correlated garbling scheme ($\mathsf{CorrGarb}$) is a deterministic reusable garbling scheme. Leveraging the randomness from cover-free sets, $\mathsf{CorrGarb}$ can be reused for an apriori bounded number of times. A key property of $\mathsf{CorrGarb}$ is that the output of the garbling algorithm is an NC^0 circuit with locality 4. A $\mathsf{CorrGarb}$ scheme consists of the following algorithms.

$\underline{\mathsf{Setup}(1^\lambda, 1^Q, 1^s) \to \mathsf{MSK}}$: This is a randomized algorithm that on input the security parameter λ, the query bound Q, and the maximum size of the circuit s, outputs the master secret key MSK.

$\underline{\mathsf{Garb}(\mathsf{MSK}, \Delta, C, x) \to (\mathsf{GC}, \mathsf{K})}$: This is a polynomial time algorithm that on input the master secret key MSK, cover-free set Δ, a $\mathsf{P/Poly}$ circuit C, and an input x, outputs the garbled circuit GC and the wire encodings K.

$\underline{\mathsf{Eval}(\mathsf{GC}, \mathsf{K}) \to y}$: This is a polynomial time algorithm that on input the garbled circuit GC and the wire encodings K, output y.

Definition 3.4. *The scheme* $\mathsf{CorrGarb} = (\mathsf{Setup}, \mathsf{Garb}, \mathsf{Eval})$ *is said to be secure correlated garbling scheme if it satisfies the following properties.*

1. ***Correctness:*** *We say that* $\mathsf{CorrGarb}$ *is correct if for any* $\lambda \in \mathbb{N}, Q = Q(\lambda), s = s(\lambda)$*, and* $\mathsf{P/Poly}$ *circuit* C*,*

$$\Pr\left[y = C(x) : \begin{array}{c}\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda), \\ (\mathsf{GC}, \mathsf{K}) = \mathsf{Garb}(\mathsf{MSK}, \Delta, C, x), \\ y = \mathsf{Eval}(\mathsf{GC}, \mathsf{K})\end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

2. ***Security:*** *We say that the* $\mathsf{CorrGarb}$ *scheme is secure if for any admissible adversary* $\mathcal{A}$*,*

$$\left\{\mathsf{Expt}_0^{\mathsf{CorrGarb},\mathcal{A},\mathcal{C}}(1^\lambda)\right\} \approx_c \left\{\mathsf{Expt}_1^{\mathsf{CorrGarb},\mathcal{A},\mathsf{sim}_{\mathsf{CorrGarb}}}(1^\lambda)\right\}$$

where the definitions of an admissible adversary, $\mathsf{Expt}_0^{\mathsf{CorrGarb},\mathcal{A},\mathcal{C}}(1^\lambda)$*, and* $\mathsf{Expt}_1^{\mathsf{CorrGarb},\mathcal{A},\mathsf{sim}_{\mathsf{CorrGarb}}}(1^\lambda)$ *are provided in Figure 1.*

Admissible Adversary: An admissible adversary is a PPT machine which submits the circuit C and proceeds to make at most Q queries to challenger for garbled circuits and wire encodings.

$\mathsf{Expt}_0^{\mathsf{CorrGarb},\mathcal{A},\mathcal{C}}(1^\lambda)$: This is the Real experiment parameterized by adversary $\mathcal{A}$ and honest challenger $\mathcal{C}$.

1. **Setup Phase:** The adversary ($\mathcal{A}$) sends the query bound Q, and the maximum size of the circuit s to the challenger ($\mathcal{C}$). $\mathcal{C}$ computes $\mathsf{MSK} \leftarrow \mathsf{Setup}(1^\lambda, 1^Q, 1^s)$.
2. **Query Phase:** $\mathcal{A}$ submits the circuit C and queries up to Q queries of the form (Δ_q, j_q^*, x_q). For every $q \in [Q]$,$\mathcal{C}$ runs $\mathsf{GC}_q, \mathsf{K}_q \leftarrow \mathsf{Garb}(\mathsf{MSK}, \Delta_q, C, x_q)$. Send $\mathsf{GC}_q, \mathsf{K}_q$ to $\mathcal{A}$.
3. **Check Phase:** $\mathcal{C}$ check if for each $q, q' \in [Q]$, $j_q^* \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'}\right)$. If the check fails, output $\perp$.
4. **Guess Phase:** $\mathcal{A}$ outputs guess b'. Output b'.

$\mathsf{Expt}_1^{\mathsf{CorrGarb},\mathcal{A},\mathsf{sim}_{\mathsf{CorrGarb}}}(1^\lambda)$: This is the Ideal experiment parameterized by $\mathcal{A}$ and stateful $\mathsf{sim}_{\mathsf{CorrGarb}}$.

1. **Setup Phase:** $\mathcal{A}$ sends the query bound Q, and the maximum size of the circuit s. Compute $\mathsf{MSK} \leftarrow \mathsf{sim}_{\mathsf{CorrGarb}}(1^\lambda, 1^Q, 1^s)$.
2. **Query Phase:** $\mathcal{A}$ submits the circuit C and queries up to Q queries of the form (Δ_q, j_q^*, x_q). For every $q \in [Q]$, run $\mathsf{GC}_q, \mathsf{K}_q \leftarrow \mathsf{sim}_{\mathsf{CorrGarb}}(1^{|C|}, \mathsf{MSK}, \Delta_q, j_q^*, x_q, C(x_q))$. Send $\mathsf{GC}_q, \mathsf{K}_q$ to $\mathcal{A}$.
3. **Check Phase:** Check if for each $q, q' \in [Q]$, $j_q^* \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'}\right)$. If the check fails, output $\perp$.
4. **Guess Phase:** $\mathcal{A}$ outputs guess b'. Output b'.

Figure 1: Real and Ideal experiments for $\mathsf{CorrGarb}$.

Note that such a correlated garbling scheme can be constructed from the correlated garbling scheme present in [AV19] by using the circuits $C_q := \mathcal{U}(x_q, \cdot)$ and input $x := C$.

3.6 Q-GID MA-FE Scheme for P/Poly

The adaptively secure Q-GID MA-FE scheme consists of the following algorithms.

$\mathsf{GlobalSetup}(1^\lambda, 1^Q, 1^n, 1^s) \to \mathsf{crs}$: This is a possibly randomized algorithm that on input the security parameter λ, the query bound Q, and the number of authorities n, and the maximum size of the encryption circuit s, outputs the common random string crs. The following algorithms take the crs as an implicit input.

$\mathsf{AuthSetup}(\mathsf{id}) \to (\mathsf{MPK}, \mathsf{MSK})$: This is a randomized algorithm that on input the authority identifier id, outputs the master public and secret key pair $(\mathsf{MPK}, \mathsf{MSK})$.

$\underline{\mathsf{KeyGen}(\mathsf{id}, \mathsf{MSK}_{\mathsf{id}}, \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x) \to \mathsf{SK}_{\mathsf{GID},\mathsf{id},x}}$: This is a possibly randomized algorithm that on input the authority identifier id, master secret key for the authority $\mathsf{MSK}_{\mathsf{id}}$, master public keys for all the authorities $\{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}$, the global identifier GID, and an attribute $x \in \mathbf{X}_{\mathsf{id}}$, outputs the secret key $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x}$.

$\underline{\mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C) \to \mathsf{CT}}$: This is a randomized algorithm that on input the public keys for all the authorities and an n-ary $\mathsf{P/Poly}$ circuit C as input, outputs the ciphertext CT.

$\underline{\mathsf{Dec}(\{\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{GID},\mathsf{id}}}\}_{\mathsf{id}\in[n]}, \mathsf{CT})}$: This is a polynomial time algorithm that on input the set of secret keys corresponding to a user with global identifier GID from all authorities $\{\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{GID},\mathsf{id}}}\}_{\mathsf{id}\in[n]}$ and a ciphertext CT, outputs y.

Definition 3.5. *The scheme* $\mathsf{MA\text{-}FE} = (\mathsf{GlobalSetup}, \mathsf{AuthSetup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be an adaptively secure* Q*-*GID $\mathsf{MA\text{-}FE}$ *scheme if it satisfies the following properties.*

1. ***Correctness:*** *We say that* $\mathsf{MA\text{-}FE}$ *is correct if for any* $\lambda \in \mathbb{N}, Q = Q(\lambda), n = n(\lambda), s = s(\lambda)$*, and* $\mathsf{P/Poly}$ *circuit* C*.*

$$\Pr\left[y = C(x_1,\ldots,x_n) : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{GlobalSetup}(1^\lambda, 1^Q, 1^n, 1^s) \\ \forall\ \mathsf{id} \in [n], (\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) \leftarrow \mathsf{AuthSetup}(\mathsf{id}), \\ \forall\ \mathsf{id} \in [n], x_{\mathsf{id}} \in \mathbf{X}_{\mathsf{id}}, \mathsf{GID} \in \mathcal{GID}, \\ \mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{GID},\mathsf{id}}} \leftarrow \mathsf{KeyGen}(\mathsf{id}, \mathsf{MSK}_{\mathsf{id}}, \\ \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{GID},\mathsf{id}}), \\ \mathsf{CT} \leftarrow \mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C), \\ y = \mathsf{MA\text{-}FE.Dec}(\{\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{GID},\mathsf{id}}}\}_{\mathsf{id}\in[n]}, \mathsf{CT}) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda)$$

2. (t,ϵ)-***Adaptive Security:*** *We say that* $\mathsf{MA\text{-}FE}$ *is* (t,ϵ)*-adaptively secure if for any admissible adversary* $\mathcal{A}$*, if we have that*

$$\left\{\mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)\right\} \approx_c^\epsilon \left\{\mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{MA\text{-}FE}}}(1^\lambda)\right\}$$

where the definitions of an admissible adversary, $\mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)$*, and* $\mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{MA\text{-}FE}}}(1^\lambda)$ *are provided in Figure 2.*

In Figure 2, we implicitly use a predicate ψ and a function Ψ. We use ψ as a flag to simulate a query for a specific GID in the post-challenge query phase and Ψ as an aggregator function to simulate all the GID instantiations till challenge query phase. The formal descriptions of ψ and Ψ are as follows.

- Predicate ψ is parameterized by a relation $\mathbf{R} \subseteq \mathbf{A} \times \mathbf{B}$ for some sets $\mathbf{A}$ and $\mathbf{B}$ and an integer n. $\psi_{\mathbf{R},n}(a^*, b^*) = 1$ if and only if there exists $n-1$ $b_i^* \in \mathbf{B}$ such that $\{(a^*, b_1^*), \ldots, (a^*, b_{n-1}^*)\} \subseteq \mathbf{R}$.
- Function Ψ is parameterized by an integer n and takes as input a relation $\mathbf{R} \subseteq \mathbf{A} \times \mathbf{B}$. $\Psi_n(\mathbf{R})$ outputs the set $\{a^* : \forall\ i \in [n], \exists\ b_i^*, (a^*, b_i^*) \in \mathbf{R}\}$ if $\{(a^*, b_i^*) : i \in [n]\} \subseteq \mathbf{R}$.

4 Augmented Statistical Lemmas

In this section, we state and prove the lemmas which we use in our constructions that are in the random oracle model (ROM). These lemmas are a stronger variations of the small-pairwise intersection lemma and cover-freeness lemmas used in [GVW12, AV19]. These works crucially rely on sampling the cover-free sets Δ_q's and the subset of users $\mathbf{S}_q$'s for $q \in [Q]$ at the beginning of the security game. However, while working in the random oracle model, the adversary has access to the random oracle, $\mathcal{H} : \mathcal{GID} \to 2^{[N]}\big|_D \times 2^{[T]}\big|_v$, and can make an unbounded number ($P \geq Q$) of queries to $\mathcal{H}$ before the start of the security game. However, for the parameter regime in employed in [GVW12], the small pairwise intersection lemma and the cover-freeness lemma will not hold if the adversary gets hold of more than Q Δ_q's and $\mathbf{S}_q$'s. In particular for large enough

Admissible Adversary: An admissible adversary is a PPT machine that runs in at most $t(\lambda)$ time and makes at most Q queries for secret keys across all n authorities such that only one query per GID, per authority, is made. In addition, the adversary will make at most one challenge query with the circuit C. The order of these queries can be in any order.

$\mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)$: This is the Real experiment parameterized by adversary $\mathcal{A}$ and honest challenger $\mathcal{C}$.

1. **Setup:** The adversary $\mathcal{A}$ provides the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. The challenger $\mathcal{C}$ runs $\mathsf{crs} \leftarrow \mathsf{GlobalSetup}(1^\lambda, 1^Q, 1^n, 1^s)$ and $\forall\ \mathsf{id} \in [n], (\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) \leftarrow \mathsf{AuthSetup}(\mathsf{id})$. $\mathcal{C}$ sends $\left(\mathsf{crs}, (\mathsf{MPK}_{\mathsf{id}})_{\mathsf{id}\in[n]}\right)$ to $\mathcal{A}$.
2. **Pre-Challenge Query Phase:** $\mathcal{A}$ sends $q \in [Q_1]$ key queries of the form $(\mathsf{GID}_q,\ \mathsf{id}_q,\ x_{\mathsf{GID}_q,\mathsf{id}_q})$ such that $\mathsf{id}_q \in [n]$, $x_{\mathsf{GID}_q,\mathsf{id}_q} \in \mathbf{X}_{\mathsf{id}_q}$. $\mathcal{C}$ runs $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} \leftarrow \mathsf{KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$ and sends $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}$ to $\mathcal{A}$.
3. **Challenge Phase:** $\mathcal{A}$ sends an n-ary $\mathsf{P/Poly}$ circuit C of maximum size s. $\mathcal{C}$ samples $\mathsf{CT} \leftarrow \mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$ and sends CT to $\mathcal{A}$.
4. **Post-Challenge Query Phase:** This is a similar to Pre-Challenge Query Phase for $q \in [Q_1 + 1, Q]$.
5. **Guess Phase:** $\mathcal{A}$ outputs guess b'. Output b'.

$\mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{MA\text{-}FE}}}(1^\lambda)$: This is the Ideal experiment parameterized by $\mathcal{A}$ and stateful $\mathsf{sim}_{\mathsf{MA\text{-}FE}}$.

1. **Setup:** The adversary $\mathcal{A}$ provides the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. Sample $\mathsf{crs} \leftarrow \mathsf{sim}_{\mathsf{MA\text{-}FE}}(1^\lambda, 1^Q, 1^n, 1^s)$ and $\mathsf{MPK}_{\mathsf{id}} \leftarrow \mathsf{sim}_{\mathsf{MA\text{-}FE}}(\mathsf{id})\ \forall\ \mathsf{id} \in [n]$. Initiate a set $\mathbf{Q}$ to be empty. Send $\left(\mathsf{crs}, (\mathsf{MPK}_{\mathsf{id}})_{\mathsf{id}\in[n]}\right)$ to $\mathcal{A}$.
2. **Pre-Challenge Query Phase:** $\mathcal{A}$ sends $q \in [Q_1]$ key queries of the form $(\mathsf{GID}_q,\ \mathsf{id}_q,\ x_{\mathsf{GID}_q,\mathsf{id}_q})$ such that $\mathsf{id}_q \in [n]$, $x_{\mathsf{GID}_q,\mathsf{id}_q} \in \mathbf{X}_{\mathsf{id}_q}$. Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$. Run $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} \leftarrow \mathsf{sim}_{\mathsf{MA\text{-}FE}}(\mathsf{id}_q, \mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$ and send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}$ to $\mathcal{A}$.
3. **Challenge Phase:** $\mathcal{A}$ sends an n-ary $\mathsf{P/Poly}$ circuit C of maximum size s. Sample $\mathsf{CT} \leftarrow \mathsf{sim}_{\mathsf{MA\text{-}FE}}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, 1^{|C|}, \mathbf{V})$ and send CT to $\mathcal{A}$ where $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
4. **Post-Challenge Query Phase:** This is a similar to Pre-Challenge Query Phase for $q \in [Q_1 + 1, Q]$. In addition, $\mathsf{sim}_{\mathsf{MA\text{-}FE}}$ takes as input $(X, C(X))$ where $X = (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n})$ if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$.
5. **Guess Phase:** $\mathcal{A}$ outputs guess b'. Output b'.

Figure 2: Real and Ideal experiments for $\mathsf{MA\text{-}FE}$.

P, the adversary can identify a subset of GID's such that these lemmas will not hold and corrupt the secret keys for these set of GIDs.

In this section, we overcome this issue by augmenting the parameters in such a way that for any $P \geq Q$, for any subset of the set $[P]$ of size Q, small pairwise intersection lemma and cover-freeness lemma holds. The key idea is to multiply the parameters by Q in such a way that the exponent in the Chernoff's bound is a factor of Q. Then, by a union bound over all possible subsets of $[P]$ of size Q, we argue that the required probability will still be negligible in the security parameter.

Remark 4.1. *Our parameter regime subsumes [GVW12]'s parameter regime and provides the same guarantees that they provide (i.e, when $P = Q$).*

4.1 Small Pairwise Intersection

Lemma 4.2. *Let $P = P(\lambda)$ and $Q = Q(\lambda)$ be such that $P \geq Q$. Let $\mathbf{S}_1, \ldots, \mathbf{S}_P$ be independently and uniformly randomly drawn sets from $[N]$, each of size D. Let $t = \Theta(Q\lambda), D = \Theta(t), N = \Theta(Q^2 t)$. Then for any subset $\mathbf{R} \subseteq [P]$ such that $|\mathbf{R}| = Q$, with all but negligible probability,*

$$\left| \bigcup_{\substack{i,j \in \mathbf{R} \\ i \neq j}} \mathbf{S}_i \cap \mathbf{S}_j \right| \leq t$$

Proof. Consider one such $\mathbf{R}^*$ and without loss of generality, assume $\mathbf{R}^* = [Q]$ (we can always rename $\mathbf{S}_i$'s as they are independently and uniformly randomly drawn sets). For every $i, j \in \mathbf{R}^* = [Q], i \neq j$, let X_{ij} be the random variable that gives $X_{ij} = |\mathbf{S}_i \cap \mathbf{S}_j|$ and let

$$X = \sum_{\substack{i,j \in [Q] \\ i<j}} X_{ij} \Rightarrow \mathbb{E}[X] = \sum_{\substack{i,j \in [Q] \\ i<j}} \mathbb{E}[X_{ij}]$$

For any fixed $i \in [Q]$ and $j \neq i$, X_{ij} is a hypergeometric random variable where we draw D elements (corresponding to $\mathbf{S}_j$) from a population of size N which consists of D successes (corresponding to $\mathbf{S}_i$). Let $D = ct$ for some $c \in \mathbb{N}$. Hence,

$$\mathbb{E}[X_{ij}] = \frac{(ct)^2}{N} \Rightarrow \mathbb{E}[X] = \frac{Q(Q-1)(ct)^2}{2N} \leq \frac{10Q^2t^2c^2}{N}$$

By the chernoff bound, for any $\sigma \geq 0$,

$$\Pr[X > (1+\sigma)\mathbb{E}[X]] < \exp\left(\frac{-\sigma^2}{2+\sigma}\mathbb{E}[X]\right)$$

By setting $t = \Theta(Q\lambda), D = \Theta(t), N = \Theta(c^2Q^2t) = \Theta(Q^2t)$, we get that $\mathbb{E}[X] = \Theta(t) = \Theta(Q\lambda)$. By the chernoff bound,

$$\Pr[X > t] = 2^{-\Omega(Q\lambda)} = \left(2^{-\Omega(\lambda)}\right)^Q = (\mathsf{negl}(\lambda))^Q$$

We will apply a union bound for any possible subset $\mathbf{R} \subseteq [P]$ as the maximum number of subsets possible is bounded by P^Q,

$$P^Q(\mathsf{negl}(\lambda))^Q = (P(\lambda)\mathsf{negl}(\lambda))^Q = (\mathsf{negl}(\lambda))^Q = \mathsf{negl}(\lambda)$$

□

4.2 Cover-Freeness

Lemma 4.3. *Let $P = P(\lambda)$ and $Q = Q(\lambda)$ be such that $P \geq Q$. Then $\Delta_1, \ldots, \Delta_P$ be independently and uniformly randomly drawn sets from $[T]$, each of size v. For $T = vQ, v = \Theta(Q^2\lambda)$, for any subset $\mathbf{R} \subseteq [P]$ such that $|\mathbf{R}| = Q$, for every $i \in \mathbf{R}$, with all but negligible probability,*

$$\Delta_i \setminus \bigcup_{\substack{j \neq i \\ j \in \mathbf{R}}} \Delta_j \neq \emptyset$$

where $\emptyset$ denotes the empty set $\{\}$.

Proof. Consider one such $\mathbf{R}^*$ and without loss of generality, assume $\mathbf{R}^* = [Q]$ (we can always rename Δ's as they are independently and uniformly randomly drawn sets). Let $G = \bigcup_{\substack{j \neq i \\ j \in \mathcal{R}^*}} \Delta_j$ for some $i \in \mathcal{R}^*$. Clearly, $|G| \leq (Q-1)v$. Let

$$X = \left| \Delta_i \setminus \bigcup_{j \neq i} \Delta_j \right| = v - |\Delta_i \cap G|$$

Hence, $\mathbb{E}[X] = v - \mathbb{E}[|\Delta_i \cap G|]$. The random variable $|\Delta_i \cap G|$ follows a hypergeometric distribution where we draw v items (corresponding to Δ_i) from a population of size T which consists of at most $(Q-1)v$ successes (corresponding to G). Hence,

$$\mathbb{E}[X] \geq v - \frac{v^2(Q-1)}{T}$$

By setting $T = vQ$ and $v = \Theta(Q^2\lambda)$, we have that $\mathbb{E}[X] = \Omega(Q\lambda)$. By Chernoff's bound for $0 < \sigma < 1$,

$$\Pr[X \leq (1-\sigma)\mathbb{E}[X]] < \exp\left(-\tfrac{\sigma^2}{2}\mathbb{E}[X]\right)$$

$$\Pr\left[\Delta \setminus \bigcup_{j \neq i} \Delta_j = \emptyset\right] = \Pr[X = 0] \leq \Pr[X \leq (1-\sigma)\mathbb{E}[X]] < 2^{-\Omega(Q\lambda)} \leq \left(2^{-\Omega(\lambda)}\right)^Q = (\mathsf{negl}(\lambda))^Q$$

We will apply a union bound for any possible subset $\mathbf{R} \subseteq [P]$ as the maximum number of subsets possible in bounded by P^Q,

$$P^Q(\mathsf{negl}(\lambda))^Q = (P(\lambda)\mathsf{negl}(\lambda))^Q = (\mathsf{negl}(\lambda))^Q = \mathsf{negl}(\lambda)$$

□

5 Adaptive 1-GID MA-FE for P/Poly

In this section we provide the formal definition, construction, and security analysis of our adaptively secure 1-GID MA-FE scheme (1MAFE). Recall from Section 2.1 that we take the one-key NA-SIM FE from Sahai-Seyalioglu [SS10] based on garbled circuits to an adaptively secure 1-GID MA-FE scheme for P/Poly circuits. We re-define the [SS10] construction as a NA-SIM secure version of 1-GID MA-FE scheme, na1MAFE, and provide its construction, *without security analysis* in Section 5.1.

We leverage the power of hybrid encryption, additive secret sharing, and non-committing encryption to construct an adaptively secure 1-GID MA-FE scheme. Recall that we sample $R_1, \ldots R_n$ and set $\mathsf{ct} = (\{\mathsf{NCE.Enc}(\mathsf{NCE.mpk}_{\mathsf{id}}, R_{\mathsf{id}})\}_{\mathsf{id}\in[n]}, \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus \mathsf{na1MAFE.ct})$. Now, each authority sends a secret key for NCE as part of the key generation algorithm and this key decrypts one component of ct. It is only after we receive keys from all the authorities that na1MAFE.ct is revealed in the clear and we can use $\{\mathsf{na1MAFE.sk}_{\mathsf{id},x_{\mathsf{id}}}\}_{\mathsf{id}}$ to to decrypt this. Note that an additional advantage of doing things this way is that for any party that does not possess secret keys from *all* the authorities, there is no information about the circuit C is revealed from the ciphertext. It is because of this, we can rely on non-adaptive security of na1MAFE and obtain adaptive security for 1MAFE. The definition of 1MAFE is provided in Section 5.2 The construction, correctness, and security analysis of 1MAFE are provided in Section 5.3.

5.1 Non-Adaptive 1-GID MA-FE for P/Poly

For completeness, we provide the construction of na1MAFE scheme. We assume that for each $\mathsf{id} \in [n]$, the attribute $x_{\mathsf{id}} \in \{0,1\}^{l(\lambda)}$.

GlobalSetup$(1^\lambda, 1^n)$: Output $\mathsf{crs} = (\lambda, n)$.

AuthSetup(id) :

- Sample $(\mathsf{PKE.pk}_{i,b}, \mathsf{PKE.sk}_{i,b}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ for $i \in [l], b \in \{0,1\}$.
- Output $\mathsf{MPK} = (\mathsf{PKE.pk}_{i,b})_{i\in[l],b\in\{0,1\}}$ and $\mathsf{MSK} = (\mathsf{PKE.sk}_{i,b})_{i\in[l],b\in\{0,1\}}$.

$\underline{\mathsf{KeyGen}(\mathsf{id}, \mathsf{MSK}_{\mathsf{id}}, \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x)}$:

- Parse $\mathsf{MSK}_{\mathsf{id}}$ as $(\mathsf{PKE.sk}_{\mathsf{id},i,b})_{i\in[l],b\in\{0,1\}}$.
- Output $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x} = (\mathsf{PKE.sk}_{\mathsf{id},i,x_i})_{i\in[l]}$.

$\underline{\mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)}$:

- Parse $\mathsf{MPK}_{\mathsf{id}}$ as $(\mathsf{PKE.pk}_{\mathsf{id},i,b})_{i\in[l],b\in\{0,1\}}$ for each $\mathsf{id}\in[n]$.
- Compute $\mathsf{GC}, \{w_{\mathsf{id},i,b}\}_{\mathsf{id}\in[n],i\in[l],b\in\{0,1\}} \leftarrow \mathsf{Garble.Garb}(1^\lambda, C)$.
- For each $\mathsf{id}\in[n], i\in[l], b\in\{0,1\}$, compute $\mathsf{PKE.ct}_{\mathsf{id},i,b} \leftarrow \mathsf{PKE.Enc}(\mathsf{PKE.pk}_{\mathsf{id},i,b}, w_{\mathsf{id},i,b})$.
- Set $\mathsf{ct}_{\mathsf{id}} = (\mathsf{GC}, \{\mathsf{PKE.ct}_{\mathsf{id},i,b}\}_{i\in[l],b\in\{0,1\}})$.
- Output $\mathsf{CT} = (\mathsf{ct}_{\mathsf{id}})_{\mathsf{id}\in[n]}$.

$\underline{\mathsf{Dec}(\{\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{GID},\mathsf{id}}}, \mathsf{CT}\})}$:

- Parse $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{GID},\mathsf{id}}}$ as $(\mathsf{PKE.sk}_{\mathsf{id},i,x_{\mathsf{id},i}})_{i\in[l]}$ for each $\mathsf{id}\in[n]$.
- Parse CT as $(\mathsf{GC}, \{\mathsf{PKE.ct}_{\mathsf{id},i,b}\}_{\mathsf{id}\in[n],i\in[l],b\in\{0,1\}})$.
- For each $\mathsf{id}\in[n], i\in[l]$, compute $w_{\mathsf{id},i,x_{\mathsf{id},i}} \leftarrow \mathsf{PKE.Dec}(\mathsf{PKE.sk}_{\mathsf{id},i,x_{\mathsf{id},i}}, \mathsf{PKE.ct}_{\mathsf{id},i,x_{\mathsf{id},i}})$.
- Output $y = \mathsf{Garble.Eval}(\widetilde{C}, w_{\mathsf{id},i,x_{\mathsf{id},i}})$.

Correctness and Security. The correctness and security of the scheme are analogous to the correctness and security of [SS10]. We omit this for brevity.

5.2 Definition

Definition 5.1. *The scheme* $\mathsf{1MAFE} = (\mathsf{GlobalSetup}, \mathsf{AuthSetup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be an adaptively-secure* 1-GID MA-FE *scheme if it satisfies the following properties.*

1. ***Correctness:*** *We say that* 1MAFE *is correct if it satisfies the correctness for an* MA-FE *scheme.*
2. ***Adaptive Security:*** *We say that the* 1MAFE *scheme satisfies adaptive security if for any admissible adversary* $\mathcal{A}$*,*

$$\left\{\mathsf{Expt}_0^{\mathsf{1MAFE},\mathcal{A},\mathcal{C}}(1^\lambda)\right\} \approx_c \left\{\mathsf{Expt}_1^{\mathsf{1MAFE},\mathcal{A},\mathsf{sim}_{\mathsf{1MAFE}}}(1^\lambda)\right\}$$

where an admissible adversary is an admissible adversary for MA-FE *scheme and only uses one* GID *for secret key queries. Hence, an admissible adversary can make at most* n *queries for secret keys.* $\mathsf{Expt}_0^{\mathsf{1MAFE},\mathcal{A},\mathcal{C}}(1^\lambda)$ *and* $\mathsf{Expt}_1^{\mathsf{1MAFE},\mathcal{A},\mathsf{sim}_{\mathsf{1MAFE}}}(1^\lambda)$ *are defined similarly for this admissibility criterion.*

5.3 Construction

The construction of the 1MAFE scheme is as follows. As mentioned in Section 2.1, we use a matrix of NCE ciphertexts.

$\underline{\mathsf{GlobalSetup}(1^\lambda, 1^n)}$: Sample $\mathsf{na1MAFE.crs} \leftarrow \mathsf{na1MAFE.Setup}(1^\lambda, 1^n)$. Output $\mathsf{crs} = \mathsf{na1MAFE.crs}$.

$\underline{\mathsf{AuthSetup}(\mathsf{id}\in[n])}$:

- Sample $(\mathsf{na1MAFE.mpk}_{\mathsf{id}}, \mathsf{na1MAFE.msk}_{\mathsf{id}}) \leftarrow \mathsf{na1MAFE.AuthSetup}(\mathsf{id})$.
- $(\mathsf{NCE.mpk}, \mathsf{NCE.msk}) \leftarrow \mathsf{NCE.Setup}(1^\lambda, 1^\kappa)$ where $\kappa = |\mathsf{na1MAFE.ct}|$.

- Output $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{na1MAFE.mpk}_{\mathsf{id}}, \mathsf{NCE.mpk})$ and $\mathsf{MSK}_{\mathsf{id}} = (\mathsf{na1MAFE.msk}_{\mathsf{id}}, \mathsf{NCE.msk})$.

$\underline{\mathsf{KeyGen}(\mathsf{id}, \mathsf{MSK}_{\mathsf{id}}, \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x)}$:

- Parse $\mathsf{MSK}_{\mathsf{id}}$ as $(\mathsf{na1MAFE.msk}_{\mathsf{id}}, \mathsf{NCE.msk}_{\mathsf{id}})$.
- Parse $\mathsf{MPK}_{\mathsf{idx}}$ as $(\mathsf{na1MAFE.mpk}_{\mathsf{idx}}, \mathsf{NCE.mpk}_{\mathsf{idx}}))$ for each $\mathsf{idx} \in [n]$.
- Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id},x} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}, \mathsf{na1MAFE.msk}_{\mathsf{id}}, \{\mathsf{na1MAFE.mpk}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x)$.
- $\mathsf{NCE.sk}_{\mathsf{id}} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}})$.
- Output $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id},x}, \mathsf{NCE.sk}_{\mathsf{id}})$.

$\underline{\mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)}$:

- For each $\mathsf{id} \in [n]$, parse $\mathsf{MPK}_{\mathsf{id}}$ as $(\mathsf{na1MAFE.mpk}_{\mathsf{id}}, \mathsf{NCE.mpk}_{\mathsf{id}})$.
- Sample $\mathsf{na1MAFE.ct} \leftarrow \mathsf{na1MAFE.Enc}(\{\mathsf{na1MAFE.mpk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$.
- For each $\mathsf{id} \in [n]$, sample a random string $R_{\mathsf{id}} \leftarrow \{0,1\}^{\kappa}$ where $\kappa = |\mathsf{na1MAFE.ct}|$.
- For each $\mathsf{id} \in [n]$, $\mathsf{NCE.ct}_{\mathsf{id}} \leftarrow \mathsf{NCE.Enc}(\mathsf{NCE.mpk}_{\mathsf{id}}, R_{\mathsf{id}})$.
- Set $\widetilde{R} = \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus \mathsf{na1MAFE.ct}$.
- Output $\mathsf{CT} = (\{\mathsf{NCE.ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \widetilde{R})$.

$\underline{\mathsf{Dec}(\{\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}\}_{\mathsf{id}\in[n]}, \mathsf{CT})}$:

- For each $\mathsf{id} \in [n]$, parse $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}$ as $(\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}, \mathsf{NCE.sk}_{\mathsf{id}})$.
- Parse CT as $(\{\mathsf{NCE.ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \widetilde{R})$.
- For each $\mathsf{id} \in [n]$, let $\rho_{\mathsf{id}} = \mathsf{NCE.Dec}(\mathsf{NCE.sk}_{\mathsf{id}}, \mathsf{NCE.ct}_{\mathsf{id}})$.
- Set $\rho = \bigoplus_{\mathsf{id}\in[n]} \rho_{\mathsf{id}}$.
- Compute $\widetilde{\mathsf{ct}} = \widetilde{R} \oplus \rho$.
- Output $y = \mathsf{na1MAFE.Dec}\left(\{\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}\}_{\mathsf{id}\in[n]}, \widetilde{\mathsf{ct}}\right)$.

Theorem 5.2. *If* na1MAFE *is a non-adaptively secure* 1-GID MA-FE *scheme for* P/Poly *circuits and* NCE *is an adaptively secure non-committing encryption scheme (Definition 3.3), the above construction is an adaptively secure* 1-GID MA-FE *scheme (Definition 5.1) for* P/Poly *circuits.*

Proof.

Correctness. The correctness of the scheme follows from the correctness of na1MAFE scheme and NCE scheme. For every $\mathsf{id} \in [n]$, We have that,

$$\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}} = \left(\mathsf{na1MAFE.KeyGen}(\mathsf{id}, \mathsf{na1MAFE.msk}_{\mathsf{id}}, \{\mathsf{na1MAFE.mpk}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}}), \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}})\right),$$

$$\mathsf{CT} = \left(\{\mathsf{NCE.Enc}(\mathsf{NCE.mpk}_{\mathsf{id}}, R_{\mathsf{id}})\}_{\mathsf{id}\in[n]}, \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus (\mathsf{na1MAFE.Enc}(\{\mathsf{na1MAFE.mpk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C))\right)$$

Hence, by the correctness of NCE,

$$\begin{aligned}\rho_{\mathsf{id}} &= \mathsf{NCE.Dec}(\mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}}), \mathsf{NCE.Enc}(\mathsf{NCE.mpk}_{\mathsf{id}}, R_{id})) \\ &= R_{\mathsf{id}} \\ \Rightarrow \rho &= \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}}\end{aligned}$$

Hence, $\widetilde{\mathsf{ct}} = \mathsf{na1MAFE.Enc}(\{\mathsf{na1MAFE.mpk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$. From the correctness of na1MAFE, we have that $y = C(x_1, \ldots, x_n)$.

Security. We prove the security of 1MAFE using a series of hybrids and claims.

$\mathsf{Hyb}_0(1^\lambda)$: This is $\mathsf{Expt}_0^{\mathsf{1MAFE},\mathcal{A},\mathcal{C}}$ from Definition 5.1.

$\mathsf{Hyb}_{1,j}(1^\lambda)$: for $j \in [n+1]$, we will fake the first $j-1$ NCE instantiations if they are queried adaptively.

$\mathsf{Hyb}_2(1^\lambda)$: In this hybrid, all the NCE instantiations are faked accordingly. We will change the way $\widetilde{R}$ is generated as a uniform random string and reveal it accordingly in post-challenge query phases.

$\mathsf{Hyb}_2(1^\lambda)$: In this hybrid, we will simulate the na1MAFE instantiation. This is $\mathsf{Expt}_1^{\mathsf{1MAFE},\mathcal{A},\mathsf{sim}_{\mathsf{1MAFE}}}$ from Definition 5.1.

Claim 5.3. *The hybrids* Hyb_0 *and* $\mathsf{Hyb}_{1,1}$ *are identically distributed.*

Proof. In the hybrid $\mathsf{Hyb}_{1,1}$, even if the first authority is not queried in the pre-challenge phase, we do not fake the NCE instantiation. Hence, the hybrids are identical. □

Claim 5.4. *Assuming the security of non-committing encryption,* $\mathsf{Hyb}_{1,j}$ *and* $\mathsf{Hyb}_{1,j+1}$ *for any* $j \in [n]$ *are computationally indistinguishable.*

Claim 5.5. *The hybrids* $\mathsf{Hyb}_{1,n+1}$ *and* Hyb_2 *are identically distributed.*

Proof. The distribution of $\widetilde{R}$ in both of these hybrids are identical and the revealed string is also distributed identically. Hence, the hybrids are identical. □

Claim 5.6. *Assuming the non-adaptive security of* na1MAFE, Hyb_2 *and* Hyb_3 *are computationally indistinguishable.*

For completeness, we provide full descriptions of these hybrids and proofs of claims in Appendix B.

6 Distributed Client-Server Framework

In this section, we formally define, construct, and analyze the security for our distributed client-server framework (dCSF). Recall from Section 2.2 that our framework is tailored for adaptive Q-GID MA-FE constructions similar to how CSF was tailored for BFE.

6.1 Definition

A dCSF scheme consists of the following algorithms.

Server Side Algorithms:

$\mathsf{ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}, \mathsf{id}, x, \Delta) \to \left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]}$: This is a polynomial time algorithm that on input the security parameter λ, number of servers n, query bound Q, the maximum size of client's circuit s, global identifier GID, server identifier id, attribute $x \in \mathbf{X}_{\mathsf{id}}$, and the cover-free set Δ, outputs the server-side encoding for x, $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]}$.

Client Side Algorithms:

$\mathsf{ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C) \to \{\widehat{C}^u\}_{u\in[N]}$: This is a randomized algorithm that on input the security parameter λ, number of servers n, query bound Q, the maximum size of the circuit s, and the circuit description C, outputs the client-side encoding for C, $\{\widehat{C}^u\}_{u\in[N]}$.

User Algorithms:

$\mathsf{UserComp}(\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\}_{\mathsf{id}\in[n]}, \widehat{C}^u) \to \widehat{y}^u_{\mathsf{GID}}$: This is a polynomial time algorithm that on input the server-side encodings for all the servers for a specific GID and the u-th user, $\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\}_{\mathsf{id}\in[n]}$, and the client-side encoding $\widehat{C}^u$ for the u-th user, outputs the encoding $\widehat{y}^u_{\mathsf{GID}}$.

Other:

$\mathsf{Decode}(\{\widehat{y}^u_{\mathsf{GID}}\}_{u\in\mathbf{S}}, \mathbf{S}) \to y$: This is a polynomial time algorithm that on input the output encodings for a subset of users $\{\widehat{y}^u_{\mathsf{GID}}\}_{u\in\mathbf{S}}$, and the set of users $\mathbf{S}$, outputs y_{GID}.

Definition 6.1. *The scheme* $\mathsf{dCSF} = (\mathsf{ServEnc}, \mathsf{ClientEnc}, \mathsf{UserComp}, \mathsf{Decode})$ *is said to be an adaptively-secure distributed-client server framework if it satisfies the following properties.*

1. ***Correctness:*** *We say that* dCSF *is correct if for any* $\lambda \in \mathbb{N}$, $n = n(\lambda), s = s(\lambda), Q = Q(\lambda)$, *and* P/Poly *circuit* C,

$$\Pr\left[y = C(x_1, \ldots, x_n) : \begin{array}{c} \{\widehat{C}^u\}_{u\in[N]} \leftarrow \mathsf{ClientEnc}(1^\lambda, 1^n 1^Q, 1^s, C), \\ \forall\ \mathsf{id} \in [n], \{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\}_{u\in[N]} = \mathsf{ServEnc}(1^\lambda, 1^n, 1^Q, 1^s, \\ \mathsf{GID}, \mathsf{id}, x_{\mathsf{GID},\mathsf{id}}, \Delta), \\ \forall\ u \in \mathbf{S}, \widehat{y}^u_{\mathsf{GID}} = \mathsf{UserComp}(\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\}_{\mathsf{id}\in[n]}, \widehat{C}^u), \\ y = \mathsf{Decode}(\{\widehat{y}^u_{\mathsf{GID}}\}_{u\in\mathbf{S}}, \mathbf{S}) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda)$$

2. ***Adaptive Security:*** *We say that* dCSF *is adaptively-secure if for any admissible PPT adversary* $\mathcal{A}$,

$$\left\{\mathsf{Expt}_0^{\mathsf{dCSF},\mathcal{A},\mathcal{C}}(1^\lambda)\right\} \approx_c \left\{\mathsf{Expt}_1^{\mathsf{dCSF},\mathcal{A},\mathsf{sim}_{\mathsf{dCSF}}}(1^\lambda)\right\}$$

where the definitions of an admissible adversary and $\mathsf{Expt}_0^{\mathsf{dCSF},\mathcal{A},\mathcal{C}}(1^\lambda)$ *are provided in Figure 3. Definition for* $\mathsf{Expt}_1^{\mathsf{dCSF},\mathcal{A},\mathsf{sim}_{\mathsf{dCSF}}}(1^\lambda)$ *is provided in Figure 4.* ψ *and* Ψ *are from Section 3.6.*

6.2 Construction

The construction of dCSF is as follows. For the construction, we use an auxiliary algorithm MsgEnc defined as follows.

Admissible Adversary: An admissible adversary is PPT machine that queries for at most Q server encodings across all n servers and queries for client encoding with at most one circuit C in any order.

$\underline{\mathsf{Expt}_0^{\mathsf{dCSF},\mathcal{A},\mathcal{C}}(1^\lambda)}$: This is the Real experiment parameterized by adversary $\mathcal{A}$ and honest challenger $\mathcal{C}$.

1. **Setup Phase:** $\mathcal{A}$ sends the number of authorities n, the query bound Q, the maximum size of the circuit s, and the set of non-corrupted users $\widetilde{\mathbf{S}}$ to the challenger ($\mathcal{C}$). $\mathcal{C}$ initiates a set $\mathbf{Q}$ to be empty.
2. **Pre-Circuit Server Query Phase:** $\mathcal{A}$ sends $q \in [Q_1]$ queries for sever-side encodings of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$ such that $|\mathbf{S}_{\mathsf{GID}_q}| = D$ and $|\Delta_q| = v$. Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to set $\mathbf{Q}$. $\mathcal{C}$ runs $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{ServEnc}(1^\lambda, 1^n, 1^Q, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$ and sends $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.
3. **Circuit Query Phase:** $\mathcal{A}$ sends the n-ary circuit C and $\{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})}$ to $\mathcal{C}$. $\mathcal{C}$ does the following
   - Sample $\{\widehat{C}^u\}_{u\in[N]} \leftarrow \mathsf{ClientEnc}(1^\lambda, 1^n, 1^Q, 1^s, C)$.
   - For each $\mathsf{GID} \in \Psi_n(\mathbf{Q})$ sample $\{\widehat{y}^u_{\mathsf{GID}}\} \leftarrow \mathsf{UserComp}(\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\}_{\mathsf{id}\in[n]}, j^*_{\mathsf{GID}}, \widehat{C}^u)$ for each $u \in \mathbf{S}_{\mathsf{GID}}$.

   Send $\{\widehat{C}^u\}_{u\in[N]\setminus\widetilde{\mathbf{S}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q}),u\in\mathbf{S}_{\mathsf{GID}}}$ to $\mathcal{A}$.
4. **Post-Circuit Query Phase:** $\mathcal{A}$ sends $q \in [Q_1+1, Q]$ queries for sever-side encodings of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$ such that $|\mathbf{S}_{\mathsf{GID}_q}| = D$ and $|\Delta_q| = v$. $\mathcal{A}$ also sends $j^*_{\mathsf{GID}_q}$ if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$. $\mathcal{C}$ runs
   - $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{ServEnc}(1^\lambda, 1^n, 1^Q, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.
   - If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, sample $\{\widehat{y}^u_{\mathsf{GID}_q}\} \leftarrow \mathsf{UserComp}(\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}}\}_{\mathsf{id}\in[n]}, j^*_{\mathsf{GID}_q}, \widehat{C}^u)$ for each $u \in \mathbf{S}_{\mathsf{GID}_q}$.
   - Otherwise, set $\{\widehat{y}^u_{\mathsf{GID}_q}\} = \perp$.

   Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to set $\mathbf{Q}$. Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}, \{\widehat{y}^u_{\mathsf{GID}_q}\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.
5. **Check Phase:** Let $Q^* \le Q$ be the number of unique $\mathbf{S}_q$'s and Δ_q's. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Abort and output $\perp$ if either of these checks fail.
6. **Guess Phase:** $\mathcal{A}$ outputs b'. Output b'.

Figure 3: Valid adversary and Real experiment for dCSF.

$\mathsf{MsgEnc}(1^\lambda, 1^n, 1^Q, 1^s, m, d)$: For each $h \in [|m|]$, let m_h denote the h-th bit of m. Sample a degree-d polynomial μ_h such that $\mu_h(0) = m_h$. Set $\mathsf{E}^u_{m,h} = \mu_h(u)$ and $\mathsf{E}^u_m = \{\mathsf{E}^u_{m,h}\}_{h\in[|m|]}$. Output $\{\mathsf{E}^u_m\}_{u\in[N]}$.

Here, N, D, t denote the parameters from Lemma 4.2 and v, T denote the parameters from Lemma 4.3.

$\underline{\mathsf{ServEnc}(1^\lambda, 1^n, 1^Q, 1^s, \mathsf{GID}, \mathsf{id}, x, \Delta)}$: For each $u \in [N]$, set $\widehat{x}^u_{\mathsf{GID},\mathsf{id}} = (\mathsf{GID}, x, \Delta)$. Output $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]}$.

$\underline{\mathsf{ClientEnc}(1^\lambda, 1^n, 1^Q, 1^s, C)}$:

$\mathsf{Expt}_1^{\mathsf{dCSF},\mathcal{A},\mathsf{sim}_{\mathsf{dCSF}}}(1^\lambda)$: This is the Ideal experiment parameterized by $\mathcal{A}$ and a stateful simulator $\mathsf{sim}_{\mathsf{dCSF}}$.

1. **Setup Phase:** $\mathcal{A}$ sends the number of authorities n, the query bound Q, the maximum size of the circuit s, and the set of non-corrupted users $\widetilde{\mathbf{S}}$. Initiate a set $\mathbf{Q}$ to be empty.
2. **Pre-Circuit Server Query Phase:** $\mathcal{A}$ sends $q \in [Q_1]$ queries for sever-side encodings of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$ such that $|\mathbf{S}_{\mathsf{GID}_q}| = D$ and $|\Delta_q| = v$. Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to set $\mathbf{Q}$. Run $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}(1^\lambda, 1^n, 1^Q, 1^s,\ \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$ and send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.
3. **Circuit Query Phase:** $\mathcal{A}$ sends the n-ary circuit C and $\{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})}$.
   - Create a set $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Sample $\{\widehat{C}^u\}_{u\in[N]\setminus\widetilde{\mathbf{S}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q}),u\in\mathbf{S}_{\mathsf{GID}}} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}(1^{|C|}, \mathbf{V}, \{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})})$.

   Send $\{\widehat{C}^u\}_{u\in[N]\setminus\widetilde{\mathbf{S}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q}),u\in\mathbf{S}_{\mathsf{GID}}}$ to $\mathcal{A}$.
4. **Post-Circuit Query Phase:** $\mathcal{A}$ sends $q \in [Q_1+1, Q]$ queries for sever-side encodings of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$ such that $|\mathbf{S}_{\mathsf{GID}_q}| = D$ and $|\Delta_q| = v$. $\mathcal{A}$ also sends $j^*_{\mathsf{GID}_q}$ if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$.
   - If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, let $V_q = (\mathsf{GID}_q, j^*_{\mathsf{GID}_q}, (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}), C(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
   - Otherwise, $V_q = \perp$.

   Run $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}, \{\widehat{y}^u_{\mathsf{GID}_q}\}_{u\in\mathbf{S}_{\mathsf{GID}_q}} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}(1^\lambda,\ 1^n, 1^Q, 1^s, \mathsf{GID}_q, \mathsf{id}_q,\ x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q, \mathbf{S}_{\mathsf{GID}_q} V_q)$.
   Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to set $\mathbf{Q}$. Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}, \{\widehat{y}^u_{\mathsf{GID}_q}\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.
5. **Check Phase:** Let $Q^* \le Q$ be the number of unique $\mathbf{S}_q$'s and Δ_q's. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Abort and output $\perp$ if either of these checks fail.
6. **Guess Phase:** $\mathcal{A}$ outputs b'. Output b'.

Figure 4: Ideal experiment for dCSF.

- Sample $\mathsf{CorrGarb.msk} \leftarrow \mathsf{CorrGarb.Setup}(1^\lambda, 1^Q, 1^s)$.
- Sample $\{\mathsf{E}^u_0\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^n, 1^Q, 1^s, 0^n, t), \{\mathsf{E}^u_1\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^n, 1^Q, 1^s, 1^n, t)$.
- Sample $\{\mathsf{E}^u_{\mathsf{CorrGarb.msk}}\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^n, 1^Q, 1^s, \mathsf{CorrGarb.msk}, t)$.
- Sample $\{\mathsf{E}^u_C\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^n, 1^Q, 1^s, C, t)$.
- For each $h \in [s'']$, let $z_h = 0^T$ and $\left\{\mathsf{E}^u_{z,h}\right\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^n, 1^Q, 1^s, z_h, D-1)$ where $s'' = |\mathsf{GC}|+|\mathsf{K}|$ from correlated garbling.
- Set $\widehat{C}^u = \left(\mathsf{E}^u_0, \mathsf{E}^u_1, \mathsf{E}^u_{\mathsf{msk}}, \mathsf{E}^u_C, \left\{\mathsf{E}^u_{z,h}\right\}_{h\in[s'']}\right)$ and output $\left\{\widehat{C}^u\right\}_{u\in[N]}$.

$\mathsf{UserComp}\left(\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{\mathsf{id}\in[n]}, \widehat{C}^u\right)$:

- For each $\mathsf{id} \in [n]$, parse $\widehat{x}^u_{\mathsf{GID},\mathsf{id}}$ as $(\mathsf{GID}_{\mathsf{id}}, x_{\mathsf{GID},\mathsf{id}}, \Delta_{\mathsf{id}})$.
- If for each $\mathsf{id} \in [n]$, $\mathsf{GID}_{\mathsf{id}}$ are not the same or Δ_{id} are not the same, output $\perp$. Let $\Delta = \Delta_1$.
- Parse $\widehat{C}^u$ as $\left(\mathsf{E}^u_0, \mathsf{E}^u_1, \mathsf{E}^u_{\mathsf{msk}}, \mathsf{E}^u_C, \left\{\mathsf{E}^u_{z,h}\right\}_{h\in[s'']}\right)$.
- Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID},\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
- For each $h \in [s'']$, parse $\mathsf{E}^u_{z,h}$ as $\{\zeta^u_{h,h'}\}_{h'\in[T]}$. let $Z^u_h = \sum_{j\in\Delta} \zeta^u_{h,j}$.
- Compute for each $h \in [s'']$, $\widehat{y}^u_{\mathsf{GID},h} = (\mathsf{CorrGarb.Garb}\,(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta, \mathsf{E}^u_C, \mathsf{E}^u_x))_h + Z^u_h$ where $(\cdot)_h$ denotes the h-th bit of the string.
- Output $\widehat{y}^u_{\mathsf{GID}} = \widehat{y}^u_{\mathsf{GID},1} \,\|\, \ldots \,\|\, \widehat{y}^u_{\mathsf{GID},s''}$ where $\mathsf{GID} = \mathsf{GID}_1$.

$\underline{\mathsf{Decode}(\{\widehat{y}^u\}_{u\in\mathbf{S}}, \mathbf{S}) :}$

- For each $u \in \mathbf{S}$, parse $\widehat{y}^u$ as $(\widehat{y}^u_1, \ldots, \widehat{y}^u_{s''})$ where $s'' = |\mathsf{GC}| + |\mathsf{K}|$ from correlated garbling.
- For each $h \in [s'']$, construct a degree-$(D-1)$ polynomial η_h such that $\eta_h(u) = \widehat{y}^u_h$.
- Set $(\mathsf{GC}, \mathsf{K}) = \eta_1(0) \,\|\, \ldots \,\|\, \eta_{s''}(0)$.
- Compute $y = \mathsf{CorrGarb.Eval}(\mathsf{GC}, \mathsf{K})$. Output y.

Lemma 6.2. *If* CorrGarb *is an adaptively secure correlated garbling scheme for* P/Poly *circuits (Definition 3.4), then the above construction is an adaptively secure distributed client-server framework (Definition 6.1) for* P/Poly *circuits.*

Proof.

Correctness. The correctness of the scheme follows from the correctness of CorrGarb and Shamir's secret sharing scheme. From Lemma 4.2, we have that $|\mathbf{S}| = D$. When $\{\widehat{C}^u\}_{u\in[N]} \leftarrow \mathsf{ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$, we have for each $u \in [N], \widehat{C}^u = \left(\mathsf{E}^u_0, \mathsf{E}^u_1, \mathsf{E}^u_{\mathsf{msk}}, \mathsf{E}^u_C, \left\{\mathsf{E}^u_{z,h}\right\}_{h\in[s'']}\right)$. By the correctness of Shamir's secret sharing,

$$\begin{aligned}
\mathsf{E}^0_{\mathsf{CorrGarb.msk}} &= \mathsf{CorrGarb.msk} \\
\mathsf{E}^0_C &= C \\
\mathsf{E}^0_x &= x = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n}) \\
Z^0_h &= 0
\end{aligned}$$

Hence, $\eta_h(0) = \widehat{y}^0_{\mathsf{GID},h} = (\mathsf{CorrGarb.Garb}\,(\mathsf{CorrGarb.msk}, \Delta, C, x))_h$. As a result, $(\mathsf{GC}, \mathsf{K}) = \eta_1(0) \| \ldots \| \eta_{s''}(0) = \mathsf{CorrGarb.Garb}\,(\mathsf{CorrGarb.msk}, \Delta, C, x)$. By the correctness of CorrGarb, we have that $y = \mathsf{CorrGarb.Eval}\,(\mathsf{GC}, \mathsf{K}) = C(x)$.

Security. We argue the security of the distributed correlated garbling scheme using the following series of hybrids and claims.

$\mathsf{Hyb}_0(1^\lambda)$: This is $\mathsf{Expt}_0^{\mathsf{dCSF},\mathcal{A},\mathcal{C}}$ in Definition 6.1.

$\mathsf{Hyb}_1(1^\lambda)$: In this hybrid, we change the way $\widehat{C}^u$ is sampled. Specifically, we will use the output of $C(x_{\mathsf{GID}})$ to sample the random input polynomials η, ζ.

$\mathsf{Hyb}_2(1^\lambda)$: In this hybrid, we will simulate the CorrGarb instantiation.

$\mathsf{Hyb}_3(1^\lambda)$: In this hybrid, we will use $0^{|C|}$ instead of C. This is $\mathsf{Expt}_1^{\mathsf{dCSF},\mathcal{A},\mathsf{sim}_{\mathsf{dCSF}}}$ from Definition 6.1.

Claim 6.3. *The hybrids* Hyb_0 *and* Hyb_1 *are identically distributed.*

Claim 6.4. *Assuming the security of the* CorrGarb *scheme,* Hyb_1 *and* Hyb_2 *are computationally indistinguishable.*

Claim 6.5. *The hybrids* Hyb_2 *and* Hyb_3 *are identically distributed.*

For completeness, we provide full descriptions of these hybrids and proofs of claims in Appendix C.

7 Static-Q-GID MA-FE for P/Poly

In this section, we will provide the definitions and construction for static-Q-GID MA-FE scheme (stMA-FE). The only difference between a stMA-FE scheme and an MA-FE scheme as mentioned in Section 2.2 is that the security of the stMA-FE scheme holds against admissible adversary *that query all Q secret-key queries across n authorities at the beginning of the security game, even before obtaining* crs *from the challenger.* The adversary in addition receives the secret keys for all the queries at once and then makes the challenge query with the circuit C. Recall that we are using a PRF scheme to solve the MA-FE sampling bottleneck in stMA-FE and this restriction is placed to utilize the security of PRF as the PRF key is embedded in MA-FE crs. Our scheme uses 1MAFE (Section 5) and dCSF (Section 6) scheme and proceeds as follows.

Authority Setup. For each $u \in [N]$, sample $(\mathsf{1MAFE.mpk}_u, \mathsf{1MAFE.msk}_u) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{id})$. Output $(\mathsf{MPK}, \mathsf{MSK}) = (\{\mathsf{1MAFE.mpk}_u\}_{u\in[N]}, \{\mathsf{1MAFE.msk}_u\}_{u\in[N]})$.

Key Generation. Sample pseudorandom $(\mathbf{S}, \Delta)$ using key K from crs. Compute id-th server encodings for $(\mathsf{GID}, x, \Delta)$ from dCSF, $\{\widehat{x}^u_{\mathsf{id}}\}$. Output $\mathsf{SK}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}, \mathsf{1MAFE.msk}_{\mathsf{id},u}, \widehat{x}^u_{\mathsf{id}})$ for $u \in \mathbf{S}$.

Encryption. Compute client encodings for C using dCSF, $\{\widehat{C}^u\}$. For each $u \in [N]$, $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, \mathsf{UserComp}(\cdot, \widehat{C}^u))$.

Note the similarities with BFE blueprint. Another requirement for BFE blueprint is that we need to generate all the non-corrupted users $\widetilde{\mathbf{S}}$ before the adversary sees the crs. If all the queries are explicitly provided in a static manner, we can sample the sets $(\mathbf{S}_q, \Delta_q)$ and compute the set of non-corrupted users $\widetilde{\mathbf{S}}$. Apart from this, we also address the technical subtlety of lemmas 4.2 and 4.3 for pseudorandom strings. The definition of stMA-FE is provided in Section 7.1, construction and security analysis in 7.2.

7.1 Definition

Definition 7.1. *The scheme* $\mathsf{stMA\text{-}FE} = (\mathsf{GlobalSetup}, \mathsf{AuthSetup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be a static-Q-GID* MA-FE *scheme if it satisfies the following properties.*

1. ***Correctness:*** *We say that* stMA-FE *is correct if it satisfies the correctness for an* MA-FE *scheme.*
2. ***Static Security:*** *We say that an* stMA-FE *scheme satisfies static security if for any admissible adversary $\mathcal{A}$, if we have that*

$$\left\{\mathsf{Expt}_0^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)\right\} \approx_c \left\{\mathsf{Expt}_1^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{stMA\text{-}FE}}}(1^\lambda)\right\}$$

where the definitions of an admissible adversary, $\mathsf{Expt}_0^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)$*, and* $\mathsf{Expt}_1^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{stMA\text{-}FE}}}(1^\lambda)$ *are provided in Figure 5.*

Admissible Adversary: An admissible adversary is a PPT machine which is an admissible adversary for an MA-FE scheme and makes the Q queries for secret keys before receiving crs, master public keys, and responses to these queries. In addition, the adversary will not make any query after the challenge query with the circuit C.

$\underline{\mathsf{Expt}_0^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda):}$ This is the Real experiment parameterized by adversary $\mathcal{A}$ and honest challenger $\mathcal{C}$.

1. **Static Query Phase:** $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum size of the challenge circuit s, and the secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$ for each $q \in [Q]$. The challenger ($\mathcal{C}$) runs
   - $\mathsf{crs} \leftarrow \mathsf{GlobalSetup}(1^\lambda, 1^Q, 1^n, 1^s)$.
   - For each $\mathsf{id} \in [n], (\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) \leftarrow \mathsf{AuthSetup}(\mathsf{id})$.
   - For each $q \in [Q], \mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} \leftarrow \mathsf{KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$, and $\{\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}\}_{q\in[Q]}$ to $\mathcal{A}$.
2. **Challenge Phase:** $\mathcal{A}$ sends an n-ary P/Poly circuit C of maximum size s. $\mathcal{C}$ samples $\mathsf{CT} \leftarrow \mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$ and sends CT to $\mathcal{A}$.
3. **Guess Phase:** $\mathcal{A}$ outputs guess b'. Output b'.

$\underline{\mathsf{Expt}_1^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{stMA\text{-}FE}}}(1^\lambda):}$ This is the Ideal experiment parameterized by $\mathcal{A}$ and stateful $\mathsf{sim}_{\mathsf{stMA\text{-}FE}}$.

1. **Static Query Phase:** $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum size of the challenge circuit s, and the secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$ for each $q \in [Q]$. Run:
   - $\mathsf{crs} \leftarrow \mathsf{sim}_{\mathsf{stMA\text{-}FE}}(1^\lambda, 1^Q, 1^n, 1^s)$.
   - For each $\mathsf{id} \in [n], \mathsf{MPK}_{\mathsf{id}} \leftarrow \mathsf{sim}_{\mathsf{stMA\text{-}FE}}(\mathsf{id})$.
   - For each $q \in [Q], \mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} \leftarrow \mathsf{sim}_{\mathsf{stMA\text{-}FE}}(\mathsf{id}_q, \mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$, and $\{\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}\}_{q\in[Q]}$ to $\mathcal{A}$.
2. **Challenge Phase:** $\mathcal{A}$ sends an n-ary P/Poly circuit C of maximum size s. Sample $\mathsf{CT} \leftarrow \mathsf{sim}_{\mathsf{stMA\text{-}FE}}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$ and send CT to $\mathcal{A}$.
3. **Guess Phase:** $\mathcal{A}$ outputs guess b'. Output b'.

Figure 5: Real and Ideal experiments for stMA-FE.

7.2 Construction

We provide the construction of stMA-FE scheme in the crs model using the pseudorandom function $\mathsf{PRF} : \{0,1\}^\lambda \times \mathcal{GID} \to 2^{[N]}\big|_D \times 2^{[T]}\big|_v$. The parameters N, t, D, T, v are from Lemmas 4.2, 4.3.

$\underline{\mathsf{GlobalSetup}(1^\lambda, 1^n, 1^Q, 1^s):}$ Sample a PRF key $K \xleftarrow{\$} \{0,1\}^\lambda$. For each $u \in [N], \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$ and output $\mathsf{crs} = (n, Q, s, (\mathsf{1MAFE.crs}_u)_{u\in[N]}, K)$.

$\underline{\mathsf{AuthSetup}(\mathsf{id}):}$

- For every $u \in [N], (\mathsf{1MAFE.mpk}_u, \mathsf{1MAFE.msk}_u) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{id})$.
- Output $\mathsf{MPK} = (\mathsf{1MAFE.mpk}_u)_{u\in[N]}$ and $\mathsf{MSK} = (\mathsf{1MAFE.msk}_u)_{u\in[N]}$.

$\underline{\mathsf{KeyGen}(\mathsf{id}, \mathsf{MSK}_{\mathsf{id}}, \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x):}$

- Parse $\mathsf{MSK}_{\mathsf{id}}$ as $(\mathsf{1MAFE.msk}_u)_{u\in[N]}$.
- For each $\mathsf{idx} \in [n]$, parse $\mathsf{MPK}_{\mathsf{idx}}$ as $(\mathsf{1MAFE.mpk}_{\mathsf{idx},u})_{u\in[N]}$.
- Deterministically sample $(\mathbf{S}, \Delta) = \mathsf{PRF.Eval}(K, \mathsf{GID})$.
- Compute $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}, \mathsf{id}, x, \Delta)$.
- For each $u \in \mathbf{S}$, calculate $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}, \mathsf{1MAFE.msk}_u, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID},\mathsf{id}})$.
- Output $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x} = (\mathbf{S}, \{\mathsf{1MAFE.sk}_u\}_{u\in S})$.

$\underline{\mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C):}$

- For each $\mathsf{id} \in [n]$, parse $\mathsf{MPK}_{\mathsf{id}}$ as $(\mathsf{1MAFE.mpk}_{\mathsf{id},u})_{u\in[N]}$.
- Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
- For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
- Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
- Output $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$.

$\underline{\mathsf{Dec}(\{\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}\}_{\mathsf{id}\in[n]}, \mathsf{CT}):}$

- Parse CT as $(\mathsf{ct}_u)_{u\in[N]}$ and for each $\mathsf{id} \in [n], \mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}$ as $(\mathbf{S}_{\mathsf{id}}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}}})$.
- If all $\mathbf{S}_{\mathsf{id}}$ are not the same, output $\perp$.
- Let $\mathbf{S} = \mathbf{S}_1$. For each $u \in \mathbf{S}, \widehat{y}^u_{\mathsf{GID}} \leftarrow \mathsf{1MAFE.Dec}(\{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, \mathsf{ct}_u)$.
- Output $y = \mathsf{dCSF.Decode}(\{\widehat{y}^u_{\mathsf{GID}}\}_{u\in\mathbf{S}}, \mathbf{S})$.

Theorem 7.2. *If* PRF *is a secure pseudorandom function (Definition 3.1),* 1MAFE *is adaptively secure* 1-GID MA-FE *for* P/Poly *circuits (Definition 5.1), and* dCSF *is an adaptively secure distributed client server framework for* P/Poly *circuits (Definition 6.1), then the above construction is a static-Q-*GID MA-FE *scheme (Definition 7.1) for* P/Poly *circuits.*

Proof.

Correctness. The correctness of the scheme follows from the correctness of dCSF, 1MAFE, and PRF. By PRF correctness, we have that all $\mathbf{S}_{\mathsf{id}}$ are the same and Δ_{id} are also the same. We have that,

$$\widehat{x}^u_{\mathsf{GID},\mathsf{id}} = (x_{\mathsf{id}}, \Delta) \text{ and}$$
$$\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}} = \left(\mathbf{S}, \left\{\mathsf{1MAFE.KeyGen}(\mathsf{id}, \mathsf{1MAFE.msk}_{\mathsf{id},u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, (x_{\mathsf{id}}, \Delta))\right\}_{u\in\mathbf{S}}\right).$$
$$\mathsf{ct}_u = \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, \mathsf{dCSF.UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)).$$

Hence for $u \in \mathbf{S}$, by the correctness of 1MAFE, $\widehat{y}^u_{\mathsf{GID}} = \mathsf{dCSF.UserComp}(\{(x_{\mathsf{id}}, \Delta)\}_{\mathsf{id}\in[n]}, \widehat{C}^u)$. Hence, $y = \mathsf{dCSF.Decode}(\{\widehat{y}^u_{\mathsf{GID}}\}_{u\in\mathbf{S}}, \mathbf{S})$. From the correctness of dCSF, we can see that $y = C(x_1,\ldots,x_n)$.

Security. We argue the security of the scheme using a series of hybrids and claims.

$\mathsf{Hyb}_0(1^\lambda)$: This is $\mathsf{Expt}_0^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathcal{C}}$ from Definition 7.1.

$\mathsf{Hyb}_1(1^\lambda)$: In this hybrid, we will check if Lemma 4.2 holds for pseudorandom sets $\mathbf{S}_q$.

$\mathsf{Hyb}_2(1^\lambda)$: In this hybrid, we will check if Lemma 4.3 holds for pseudorandom sets Δ_q.

$\mathsf{Hyb}_{3,j}$: for $j \in [N+1]$, we will simulate the first $j-1$ instantiation of 1MAFE using $\mathsf{sim}^u_{\mathsf{1MAFE}}$.

$\mathsf{Hyb}_4(1^\lambda)$: In this hybrid, we will simulate the dCSF instantiation. This is the $\mathsf{Expt}_1^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{stMA\text{-}FE}}}$ from Definition 7.1.

Claim 7.3. *Assuming the security of* PRF *scheme,* Hyb_0 *and* Hyb_1 *are computationally indistinguishable.*

Claim 7.4. *Assuming the security of* PRF *scheme,* Hyb_0 *and* Hyb_1 *are computationally indistinguishable.*

Claim 7.5. *The hybrids* Hyb_2 *and* $\mathsf{Hyb}_{3,1}$ *are identically distributed.*

Proof. In $\mathsf{Hyb}_{3,1}$, we do not simulate any 1MAFE instantiations. Hence, Hyb_2 and $\mathsf{Hyb}_{3,1}$ are identical. □

Claim 7.6. *Assuming the adaptive security of* 1MAFE, $\mathsf{Hyb}_{3,j}$ *and* $\mathsf{Hyb}_{3,j+1}$ *for any* $j \in [N-1]$ *are computationally indistinguishable.*

Claim 7.7. *Assuming the adaptive security of* 1MAFE, $\mathsf{Hyb}_{3,N}$ *and* $\mathsf{Hyb}_{3,N+1}$ *are computationally indistinguishable.*

Proof. The proof of this claim is similar to proof of Claim 7.6. □

Claim 7.8. *Assuming the security of* dCSF, $\mathsf{Hyb}_{3,N+1}$ *and* Hyb_4 *are computationally indistinguishable.*

For completeness, we provide full descriptions of these hybrids and proofs of the claims are in Section D.

8 Adaptive Q-GID MA-FE for P/Poly Using niKE

In the previous section, we saw how [AV19]'s blueprint is insufficient to realize a Q-GID MA-FE scheme. If the n independent authorities cannot agree upon $(\mathbf{S}, \Delta)$ for a specific GID, we cannot rely on the correctness and security of 1MAFE and dCSF. One remedy for this is to use a programmable random oracle $\mathcal{H}$. The GlobalSetup algorithm embeds the description of a hash function in the crs and the KeyGen algorithm can use this to sample $(\mathbf{S}, \Delta)$ deterministically using GID. However, this will only be NA-SIM secure. The reason is that an adaptive adversary can query for a GID such that a user $u^* \in [N]$ can be corrupted and the challenger might not know about this during the encryption. In this case, the challenger cannot efficiently simulate the u^*-th instantiation of 1MAFE.

As mentioned in the technical overview, this issue can be solved if we substitute the random oracle with a scheme that can sample a random PRF key K such that only the authorities are privy to this information and any adversary has negligible advantage in guessing K. This is exactly what a statically secure non-interactive key exchange scheme (niKE, Definition 3.2) provides. So, we can rely on the PRF security even in the adaptive MA-FE security game. We leverage BFE blueprint and a niKE scheme to construct an adaptively secure Q-GID MA-FE scheme in this section and prove its security.

The construction of the scheme uses niKE (Definition 3.2), 1MAFE (Definition 5.1), PRF (Definition 3.1), and dCSF (Definition 6.1). It proceeds as follows.

$\underline{\mathsf{GlobalSetup}(1^\lambda, 1^Q, 1^n, 1^s)}$:

- For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
- $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
- Output $\mathsf{crs} = (\lambda, Q, n, s, (\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs})$.

$\underline{\mathsf{AuthSetup}(\mathsf{id}) :}$

- For each $u \in [N]$, $(\mathsf{1MAFE.mpk}_u, \mathsf{1MAFE.msk}_u) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{id})$.
- Sample $(\mathsf{niKE.pv}, \mathsf{niKE.sv}) \leftarrow \mathsf{niKE.Publish}(\mathsf{id})$.
- Output $\mathsf{MPK} = (\{\mathsf{1MAFE.mpk}_u\}_{u\in[N]}, \mathsf{niKE.pv})$ and $\mathsf{MSK} = (\{\mathsf{1MAFE.msk}_u\}_{u\in[N]}, \mathsf{niKE.sv})$.

$\underline{\mathsf{KeyGen}(\mathsf{id}, \mathsf{MSK}_{\mathsf{id}}, \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x) :}$

- Parse $\mathsf{MSK}_{\mathsf{id}}$ as $(\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.sv}_{\mathsf{id}})$.
- Parse $\mathsf{MPK}_{\mathsf{idx}}$ as $(\{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{idx}})$ for each $\mathsf{idx} \in [n]$.
- Compute $K \leftarrow \mathsf{niKE.KeyGen}(\{\mathsf{niKE.pv}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{id}, \mathsf{niKE.sv}_{\mathsf{id}})$.
- Deterministically sample $(\mathbf{S}, \Delta) = \mathsf{PRF.Eval}(K, \mathsf{GID})$.
- Compute $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}, \mathsf{id}, x, \Delta)$.
- For each $u \in \mathbf{S}$, calculate $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}, \mathsf{1MAFE.msk}_{\mathsf{id},u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID},\mathsf{id}})$.
- Output $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x} = (\mathbf{S}, \{\mathsf{1MAFE.sk}_u\}_{u\in\mathbf{S}})$.

$\underline{\mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)}$

- For each $\mathsf{id} \in [n]$, parse $\mathsf{MPK}_{\mathsf{id}}$ as $(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$.
- Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
- For each $u \in [N]$, let $F^u(\cdot, \ldots, \cdot) = \mathsf{UserComp}(\cdot, \ldots, \cdot, \widehat{C}^u)$.
- Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
- Output $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$.

$\underline{\mathsf{Dec}(\{\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}\}_{\mathsf{id}\in[n]}, \mathsf{CT}) :}$

- Parse CT as $(\mathsf{ct}_u)_{u\in[N]}$ and for each $\mathsf{id} \in [n]$, $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}$ as $(\mathbf{S}_{\mathsf{id}}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}}})$.
- If all $\mathbf{S}_{\mathsf{id}}$ are not the same, output $\perp$.
- Let $\mathbf{S} = \mathbf{S}_1$. For each $u \in \mathbf{S}$, $\widehat{y}^u_{\mathsf{GID}} \leftarrow \mathsf{1MAFE.Dec}(\{\mathsf{1MAFE.sk}_{id,u}\}_{\mathsf{id}\in[n]}, \mathsf{ct}_u)$.
- Output $y = \mathsf{dCSF.Decode}(\{\widehat{y}^u_{\mathsf{GID}}\}_{u\in\mathbf{S}}, \mathbf{S})$.

Theorem 8.1. *If* niKE *is a statically secure non-interactive key exchange scheme for n parties (Definition 3.2),* 1MAFE *is an adaptively secure* 1-GID MA-FE *for* P/Poly *circuits (Definition 5.1),* dCSF *is an adaptively secure distributed client-server framework for* P/Poly *circuits (Definition 6.1), and* PRF *is a secure pseudorandom function (Definition 3.1), then the above construction is an adaptively secure Q-*GID MA-FE *scheme (Definition 3.5) for* P/Poly *circuits.*

Proof.

Correctness. The correctness of the scheme follows from the correctness of 1MAFE, dCSF, and niKE.

Security. We argue the security of the scheme using the following series of hybrids and claims.

$\mathsf{Hyb}_0(1^\lambda)$: This is $\mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)$ from Definition 3.5.

$\mathsf{Hyb}_1(1^\lambda)$: In this hybrid, we utilize the correctness of niKE and sample the key used for the PRF instantiation early for a randomly chosen id^*.

$\mathsf{Hyb}_2(1^\lambda)$: In this hybrid, we sample the key K uniformly randomly.

$\mathsf{Hyb}_3(1^\lambda)$: In this hybrid, we sample $\mathbf{S}_q$'s and Δ_q's uniformly randomly.

$\mathsf{Hyb}_4(1^\lambda)$: In this hybrid, we sample $\mathbf{S}_q$'s and Δ_q's early and check if Lemmas 4.2 and 4.3 hold.

$\mathsf{Hyb}_{5,j}(1^\lambda)$: for $j \in [N+1]$, we simulate the first $j-1$ instantiations of $\mathsf{1MAFE}$ using $\mathsf{sim}^u_{\mathsf{1MAFE}}$.

$\mathsf{Hyb}_6(1^\lambda)$: In this hybrid, we will simulate the dCSF instantiation. This is $\mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{MA\text{-}FE}}}(1^\lambda)$ from Definition 3.5.

Claim 8.2. *Assuming the correctness of* niKE *scheme,* Hyb_0 *and* Hyb_1 *are statistically indistinguishable.*

Proof. The proof of this claim is immediate from Definition 3.2. □

Claim 8.3. *Assuming the security of* niKE *scheme,* Hyb_1 *and* Hyb_2 *are computationally indistinguishable.*

Claim 8.4. *Assuming the security of* PRF *scheme,* Hyb_2 *and* Hyb_3 *are computationally indistinguishable.*

Claim 8.5. *Hybrids* Hyb_3 *and* Hyb_4 *are statistically indistinguishable.*

Proof. The proof of this claim in immediate by Lemmas 4.2 and 4.3. □

Claim 8.6. *The hybrids* Hyb_4 *and* $\mathsf{Hyb}_{5,1}$ *are identically distributed.*

Proof. Note that in $\mathsf{Hyb}_{5,1}$, we do not simulate any instantiations of $\mathsf{1MAFE}$. Hence, Hyb_4 and $\mathsf{Hyb}_{5,1}$ are identically distributed. □

Claim 8.7. *Assuming the adaptive security of* $\mathsf{1MAFE}$, $\mathsf{Hyb}_{5,j}$ *and* $\mathsf{Hyb}_{5,j+1}$ *for any* $j \in [N-1]$ *are computationally indistinguishable.*

Claim 8.8. *Assuming the security of* dCSF, $\mathsf{Hyb}_{5,N+1}$ *and* Hyb_6 *are computationally indistinguishable.*

For completeness, we provide full descriptions of these hybrids and proofs of claims in Appendix E.

9 Bootstrapping MA-FE for P/Poly

In this section, we present our bootstrapping compiler to compose an n_1-authority $\mathsf{MA\text{-}FE}$ scheme for $\mathsf{P/Poly}$ and an n_2-authority functional encryption scheme for $\mathsf{P/Poly}$ into an (n_1+n_2)-authority $\mathsf{MA\text{-}FE}$ scheme for $\mathsf{P/Poly}$. In Section 2.4, we explained our compiler for $n_1 = n_2 = n$. However, the same approach flows naturally to any constant n_1, n_2. Recall from Section 2.4, the main idea of our compiler is as follows: if we have a $\mathsf{P/Poly}$ (n_1+n_2)-ary circuit C inputs, $x_1, \ldots, x_{n_1+n_2}$, we can create an n_2-ary $\mathsf{P/Poly}$ circuit F_C that takes as input, $x_{n_1+1}, \ldots, x_{n_1+n_2}$ and outputs a description of an n_1-ary $\mathsf{P/Poly}$ $\widetilde{C}$. This can be achieved if we simply set $\widetilde{C} = C(\cdot, \ldots, \cdot, x_{n_1+1}, \ldots, x_{n_1+n_2})$. Now, if we have instantiations of n_1-authority and n_2-authority $\mathsf{MA\text{-}FE}$ schemes for $\mathsf{P/Poly}$ circuits, denoted by $\mathsf{n_1MAFE}$ and $\mathsf{n_2MAFE}$ respectively, we can make the authorities of $\mathsf{n_1MAFE}$ responsible for inputs $x_1, \ldots, x_{n_1+1}$ and $\mathsf{n_2MAFE}$ responsible for inputs $x_{n_1+1}, \ldots, x_{n_1+n_2}$. In the encryption algorithm, we can generate the circuit F_C that given the description of C, $x_{n_1+1}, \ldots, x_{n_1+n_2}$, outputs $\mathsf{n_1MAFE.Enc}(\widetilde{C})$, and encrypt F_C with $\mathsf{n_2MAFE}$.

One thing we abstracted in the technical overview is that if $\mathsf{n_2MAFE}$ does not take randomness, $\mathsf{n_1MAFE.Enc}(\widetilde{C})$ will be deterministic and render the scheme vulnerable to polynomial time adversaries. There is a simple fix to this problem, use a pseudorandom function. While encrypting F_C, we can sample a PRF key K and hardwire it in F_C. Inside F_C, we will use the pseudorandomness $R = \mathsf{PRF}(K, (\mathsf{GID}, \{x_{\mathsf{id}}\}_{\mathsf{id} \in [n_1+1, n_1+n_2]}))$

to compute $\mathsf{n}_1\mathsf{MAFE.Enc}(\widetilde{C}; R)$. Why is this secure? Can't the adversary query the same $\{x_{\mathsf{id}}\}_{\mathsf{id}\in[1,n_1+n_2]}$ and break the security when the same R is used in another encryption? Note that this goes against the security definition of Q-GID MA-FE (Definition 3.5). Concretely, per GID, $x_1, \ldots, x_{n_1+n_2}$ are uniquely defined from adversary's queries. Then, this (pseudo)randomness will not be reused.

The security the scheme flows quite naturally from the security of n_2MAFE, PRF, and n_1MAFE schemes. Once we simulate n_2MAFE, we can internally rely on the security of PRF and n_1MAFE schemes to argue that no non-trivial information about C is revealed to the adversary.

We emphasize once more, that this tree-esque bootstrapping can only be done constant-many times. So, for a single composition using this bootstrapping, the size of the resulting MA-FE scheme's Enc circuit will be at least $s^{(2)}(s^{(1)}(\lambda))$ where $s^{(1)}$ and $s^{(2)}$ are the sizes of $\mathsf{n}_1\mathsf{MAFE.Enc}$ and $\mathsf{n}_2\mathsf{MAFE.Enc}$ respectively. Formal construction, correctness, efficiency, and security are provided below.

Notation. For the ease of exposition, throughout this section we assume that the input x to the key generation algorithm implicitly contains the information of the global user identifier GID. However, in a slight abuse of notation, we sometimes write the GID explicitly. In such cases, we assume that x is free of the information of GID. Furthermore, we assume that the P/Poly challenge circuit C checks the GID of all the inputs before evaluation and trivially outputs $\perp$ if all the inputs do not contain the same GID. We also relax the notation for KeyGen and assume that input $\{\mathsf{MPK}_{\mathsf{id}x}\}_{\mathsf{id}x\in[n]}$ is provided implicitly.

We provide the construction for the $(n_1 + n_2)$-authority MA-FE scheme using a secure PRF scheme $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{L(\lambda)}$ (Definition 3.1). Here, $\ell(\lambda) = \mathsf{poly}(\lambda, |\mathsf{GID}|, n_2)$ and $L(\lambda) = \mathsf{poly}(\lambda, |\mathsf{GID}|, n_1, n_2)$. Let n_1MAFE denote the n_1-authority MA-FE scheme and n_2MAFE denote the n_1-authority MA-FE scheme.

$\underline{\mathsf{GlobalSetup}(1^\lambda, 1^Q, 1^{n_1}, 1^{n_2}, 1^s)}$:

- $\mathsf{n}_1\mathsf{MAFE.crs} \leftarrow \mathsf{n}_1\mathsf{MAFE.GlobalSetup}(1^\lambda, 1^Q, 1^{n_1}, 1^s)$.
- $\mathsf{n}_2\mathsf{MAFE.crs} \leftarrow \mathsf{n}_2\mathsf{MAFE.GlobalSetup}(1^\lambda, 1^Q, 1^{n_2}, 1^s)$.
- Output $\mathsf{crs} = (\mathsf{n}_1\mathsf{MAFE.crs}, \mathsf{n}_2\mathsf{MAFE.crs})$.

$\underline{\mathsf{AuthSetup}(\mathsf{id} \in [n_1 + n_2])}$:

- Parse crs as $(\mathsf{n}_1\mathsf{MAFE.crs}, \mathsf{n}_2\mathsf{MAFE.crs})$.
- If $\mathsf{id} \in [n_1]$, sample $(\mathsf{n}_1\mathsf{MAFE.mpk}, \mathsf{n}_1\mathsf{MAFE.msk}) \leftarrow \mathsf{n}_1\mathsf{MAFE.AuthSetup}(\mathsf{id})$. Set $(\mathsf{MPK}, \mathsf{MSK}) = (\mathsf{n}_1\mathsf{MAFE.mpk}, \mathsf{n}_1\mathsf{MAFE.msk})$.
- If $\mathsf{id} \in [n_1 + 1, n_1 + n_2]$, sample $(\mathsf{n}_2\mathsf{MAFE.mpk}, \mathsf{n}_2\mathsf{MAFE.msk}) \leftarrow \mathsf{n}_2\mathsf{MAFE.AuthSetup}(\mathsf{id} - n_1)$. Set $(\mathsf{MPK}, \mathsf{MSK}) = (\mathsf{n}_2\mathsf{MAFE.mpk}, \mathsf{n}_2\mathsf{MAFE.msk})$.
- Output $(\mathsf{MPK}, \mathsf{MSK})$.

$\underline{\mathsf{KeyGen}(\mathsf{id}, \mathsf{MSK}_{\mathsf{id}}, x)}$:

- If $\mathsf{id} \in [n_1]$, parse $\mathsf{MSK}_{\mathsf{id}}$ as $\mathsf{n}_1\mathsf{MAFE.msk}_{\mathsf{id}}$. Sample $\mathsf{n}_1\mathsf{MAFE.sk}_{\mathsf{id},x} \leftarrow \mathsf{n}_1\mathsf{MAFE.KeyGen}(\mathsf{id}, \mathsf{n}_1\mathsf{MAFE.msk}_{\mathsf{id}}, x)$. Set $\mathsf{SK}_{\mathsf{id},x} = \mathsf{n}_1\mathsf{MAFE.sk}_{\mathsf{id},x}$.
- If $\mathsf{id} \in [n_1 + 1, n_1 + n_2]$, parse $\mathsf{MSK}_{\mathsf{id}}$ as $\mathsf{n}_2\mathsf{MAFE.msk}_{\mathsf{id}}$. Sample $\mathsf{n}_2\mathsf{MAFE.sk}_{\mathsf{id},x} \leftarrow \mathsf{n}_2\mathsf{MAFE.KeyGen}(\mathsf{id} - n_1, \mathsf{n}_2\mathsf{MAFE.msk}_{\mathsf{id}}, x)$. Set $\mathsf{SK}_{\mathsf{id},x} = \mathsf{n}_2\mathsf{MAFE.sk}_{\mathsf{id},x}$.
- Output $\mathsf{SK}_{\mathsf{id},x}$.

$\underline{\mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1+n_2]}, C)}$:

- For each $\mathsf{id} \in [n_1]$, parse $\mathsf{MPK}_{\mathsf{id}}$ as $\mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}}$ and for each $\mathsf{id} \in [n_1 + 1, n_1 + n_2]$ as $\mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}}$.
- Sample a PRF key $K \xleftarrow{\$} \{0,1\}^\lambda$.

- Sample $\mathsf{n_2MAFE.ct} \leftarrow \mathsf{n_2MAFE.Enc}(\{\mathsf{n_2MAFE.mpk_{id}}\}_{\mathsf{id}\in[n_1+1,n_1+n_2]}, F(\cdot,\ldots,\cdot,\{\mathsf{n_1MAFE.mpk_{id}}\}_{\mathsf{id}\in[n_1]}, K, C))$ where the description of the circuit F is provided in Figure 6.
- Output $\mathsf{CT} = \mathsf{n_2MAFE.ct}$.

$\underline{\mathsf{Dec}(\{\mathsf{SK}_{\mathsf{id},x_{\mathsf{id}}}\}_{\mathsf{id}\in[n_1+n_2]}, \mathsf{CT}) :}$

- For each $\mathsf{id} \in [n_1]$, parse $\mathsf{SK}_{\mathsf{id},x_{\mathsf{id}}}$ as $\mathsf{n_1MAFE.sk}_{\mathsf{id},x_{\mathsf{id}}}$ and for each $\mathsf{id} \in [n_1+1, n_1+n_2]$ as $\mathsf{n_2MAFE.sk}_{\mathsf{id},x_{\mathsf{id}}}$. Parse CT as $\mathsf{n_2MAFE.ct}$.
- Let $\mathsf{ct}^* = \mathsf{n_2MAFE.Dec}(\{\mathsf{n_2MAFE.sk}_{\mathsf{id},x_{\mathsf{id}}}\}_{\mathsf{id}\in[n_1+1,n_1+n_2]}, \mathsf{n_2MAFE.ct})$.
- Output $y = \mathsf{n_1MAFE.Dec}(\{\mathsf{n_1MAFE.sk}_{\mathsf{id},x_{\mathsf{id}}}\}_{\mathsf{id}\in[n_1]}, \mathsf{ct}^*)$.

$\underline{F(x_1,\ldots,x_{n_2}, \{\mathsf{MPK_{id}}\}_{\mathsf{id}\in[n_1]}, K, C) :}$

- Evaluate the pseudorandom string $R \leftarrow \mathsf{PRF}(K, (\mathsf{GID}, (\mathsf{id}, x_{\mathsf{id}})_{\mathsf{id}\in[n_2]}))$.
- Output $\mathsf{n_1MAFE.Enc}(\{\mathsf{MPK_{id}}\}_{\mathsf{id}\in[n_1]}, C(\cdot,\ldots,\cdot,x_1,\ldots,x_{n_2}); R)$

Figure 6: Description of the circuit F.

Efficiency. Let $s^{(1)}_{\mathsf{Enc}} = s^{(1)}_{\mathsf{Enc}}(\lambda, Q, n_1, |C|)$ denote the size of the encryption circuit for $\mathsf{n_1MAFE}$. Similarly, let $s^{(2)}_{\mathsf{Enc}}$ denote the size of the encryption circuit for $\mathsf{n_1MAFE}$. Let s_{PRF} denote the size of the PRF circuit. The size of the encryption circuit of our bootstrapping compiler is given by

$$s_{\mathsf{Enc}} \leq s^{(2)}_{\mathsf{Enc}}\left(\lambda, Q, n_2, s_{\mathsf{PRF}} + s^{(1)}_{\mathsf{Enc}}(\lambda, Q, n_1, |C|)\right)$$

We can see that for d layers of bootstrapping, we get $s_{\mathsf{Enc}} \leq O(Q^d\ 2^{d^2}\ \mathsf{poly}(|C|)^d)$. Hence, any super-constant layers of bootstrapping will result in a non-efficient encryption circuit.

Theorem 9.1. *If* PRF *is a secure pseudorandom function (Definition 3.1),* $\mathsf{n_1MAFE}$ *is an adaptively secure* n_1*-authority* Q*-*GID MA-FE *scheme and* $\mathsf{n_2MAFE}$ *is an adaptively secure* n_2*-authority* Q*-*GID MA-FE *scheme for* P/Poly *circuits (Definition 3.5) where* $n_1, n_2 = O(1)$ *and* $n_1, n_2 > 1$*, then the above construction is an adaptively secure* (n_1+n_2)*-authority* Q*-*GID MA-FE *scheme for* P/Poly *circuits (Definition 3.5).*

Proof.

Correctness. The correctness of the scheme follows from the correctness of $\mathsf{n_1MAFE}$ and $\mathsf{n_2MAFE}$.

Security. We prove the security of the construction, using the following hybrids and claims.

$\mathsf{Hyb}_0(1^\lambda)$: This is $\mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)$ from Definition 3.5.

$\mathsf{Hyb}_1(1^\lambda)$: In this hybrid, we will simulate $\mathsf{n_2MAFE}$ using the simulator sim_2.

$\mathsf{Hyb}_2(1^\lambda)$: In this hybrid, we will change the circuit F (Figure 6) that is simulated in sim_2 so that it uses a hardwired pseudorandom string instead of evaluating the PRF as part of it's evaluation.

$\mathsf{Hyb}_3(1^\lambda)$: In this hybrid, we will replace all PRF evaluations with uniformly random strings.

$\mathsf{Hyb}_4(1^\lambda)$: In this hybrid, we will simulate $\mathsf{n_1MAFE}$ which is evaluated inside sim_2 using sim_1. This is $\mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{MA\text{-}FE}}}$ from Definition 3.5.

Claim 9.2. *Assuming the adaptive security of* $\mathsf{n_2MAFE}$*,* $\mathsf{Hyb}_0(1^\lambda)$ *and* $\mathsf{Hyb}_1(1^\lambda)$ *are computationally indistinguishable.*

Claim 9.3. $\mathsf{Hyb}_1(1^\lambda)$ *and* $\mathsf{Hyb}_2(1^\lambda)$ *are identically distributed.*

Proof. As the only difference between these hybrids is that the PRF evaluation is deferred from inside the circuit to being used as an input, the output distribution of the circuit remains identical. Hence, by the simulation security, both of these hybrids are identical. □

Claim 9.4. *Assuming the security of the* PRF *scheme,* $\mathsf{Hyb}_2(1^\lambda)$ *and* $\mathsf{Hyb}_3(1^\lambda)$ *are computationally indistinguishable.*

Claim 9.5. *Assuming the adaptive security of* $\mathsf{n}_1\mathsf{MAFE}$*,* $\mathsf{Hyb}_3(1^\lambda)$ *and* $\mathsf{Hyb}_4(1^\lambda)$ *are computationally indistinguishable.*

For completeness, we provide full descriptions of these hybrids and proofs of claims in Appendix F. We also state the following corollary of Theorem 8.1 and Theorem 9.1.

Corollary 9.6. *Assuming the hardness of Decisional Diffie-Hellman assumption,* DDH *or Bilinear Decisional Diffie-Hellman assumption,* BDDH*, there exists a* n*-authority* Q-GID MA-FE *scheme (Definition 3.5) for* P/Poly *circuits with* $n = O(1), n > 1$ *authorities in the* crs *model.*

Proof. Assuming DDH, we can instantiate a $n = 2$ authority MA-FE scheme using Diffie-Hellman key exchange protocol [DH76a] or for $n = 3$ authority from the three-party key exchange protocol in [Jou04]. Using Theorem 9.1, we can obtain MA-FE for any $n = O(1)$. □

10 Partial Adaptive Q-GID MA-FE for P/Poly in ROM

In this section, we provide the formal definition, construction, and security analysis for our partial adaptive Q-GID MA-FE scheme for P/Poly circuits, parMA-FE. As mentioned in the technical overview, we use a random oracle $\mathcal{H}$ to consistently sample the sets $(\mathbf{S}, \Delta)$ among n authorities. In addition, as mentioned in Section 8, to realize adaptive security in ROM, we need to know the set of non-corrupted users $\widetilde{\mathbf{S}}$ and the unique indices $(j_1^*, \ldots, j_Q^*)$ apriori from the adversary before the beginning of the security game[14]. If the adversary *statically* handed the information about the string $(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$, we can rely on $\mathcal{H}$ to realize construct adaptively secure Q-GID MA-FE scheme. This is the main idea behind our partially adaptive MA-FE construction. Moreover, we need to make sure that adversary behaves as promised at the end of the security game by using "Check Phase" as seen in security definitions of dCSF (Definition 6.1) and CorrGarb (Definition 3.4).

The construction of parMA-FE is almost similar to Section 8 except that we replace niKE and PRF with a random oracle $\mathcal{H}$. We concede that this is not a standard notion of security for Q-GID MA-FE schemes. We define the security definition for parMA-FE scheme this way so that we can use it as an intermediate step in constructing Q-GID MA-FE. In particular, we show how to convert a sub-exponentially secure parMA-FE scheme into an MA-FE scheme in Section 11. The definition of parMA-FE is provided in Section 10.1, the construction, correctness, and security are provided in Section 10.2

10.1 Definition

Definition 10.1. *The scheme* $\mathsf{parMA\text{-}FE} = (\mathsf{GlobalSetup}, \mathsf{AuthSetup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be partial adaptively secure* Q-GID MA-FE *scheme if it satisfies the following properties.*

1. ***Correctness:*** *We say that the scheme is correct if it satisfies the correctness for an* MA-FE *scheme.*
2. (t, ϵ)-***Partial Adaptive Security:*** *We say that an* parMA-FE *scheme is partial adaptively secure if for any admissible adversary* $\mathcal{A}$*, if we have that*

$$\left\{\mathsf{Expt}_0^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)\right\} \approx_c^\epsilon \left\{\mathsf{Expt}_1^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{parMA\text{-}FE}}}(1^\lambda)\right\}$$

[14] ... or at least during challenge query phase.

where the definitions of an admissible adversary, $\mathsf{Expt}_0^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)$*, and* $\mathsf{Expt}_1^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{parMA\text{-}FE}}}(1^\lambda)$ *are provided in Figure 7.* ψ *and* Ψ *are from Section 3.6.*

10.2 Construction

We provide the construction using a random oracle $\mathcal{H} : \mathcal{GID} \to 2^{[N]}\big|_D \times 2^{[T]}\big|_v$, 1MAFE, and dCSF as follows.

$\underline{\mathsf{GlobalSetup}(1^\lambda, 1^Q, 1^n, 1^s, \widetilde{\mathbf{S}}, (j_1^*, \ldots, j_Q^*)) :}$

- Let $\mathcal{H}$ be a hash function from $\mathcal{GID} \to 2^{[N]}\big|_D \times 2^{[T]}\big|_v$.
- For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
- Output $\mathsf{crs} = (\lambda, Q, n, s, \{\mathsf{1MAFE.crs}_u\}_{u\in[N]}, \widetilde{\mathbf{S}}, (j_1^*, \ldots, j_Q^*), \mathcal{H})$.

$\underline{\mathsf{AuthSetup}(\mathsf{id} \in [n]) :}$

- For each $u \in [N]$, $(\mathsf{1MAFE.mpk}_u, \mathsf{1MAFE.msk}_u) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{id})$.
- Output $\mathsf{MPK} = \{\mathsf{1MAFE.mpk}_u\}_{u\in[N]}$ and $\mathsf{MSK} = \{\mathsf{1MAFE.msk}_u\}_{u\in[N]}$.

$\underline{\mathsf{KeyGen}(\mathsf{id}, \mathsf{MSK}_{\mathsf{id}}, \{\mathsf{MPK}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x) :}$

- Parse $\mathsf{MSK}_{\mathsf{id}}$ as $\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]}$.
- For each $\mathsf{idx} \in [n]$, parse $\mathsf{MPK}_{\mathsf{idx}}$ as $\{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{u\in[N]}$.
- Deterministically sample $(\mathbf{S}, \Delta) = \mathcal{H}(\mathsf{GID})$.
- Compute $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}, \mathsf{id}, x, \Delta)$.
- For each $u \in \mathbf{S}$, calculate $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}, \mathsf{1MAFE.msk}_{\mathsf{id},u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID},\mathsf{id}})$.
- Output $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x} = (\mathbf{S}, \{\mathsf{1MAFE.sk}_u\}_{u\in\mathbf{S}})$.

$\underline{\mathsf{Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C) :}$

- For each $\mathsf{id} \in [n]$, parse $\mathsf{MPK}_{\mathsf{id}}$ as $\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}$.
- Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
- For each $u \in [N]$, let $F^u(\cdot, \ldots, \cdot) = \mathsf{UserComp}(\cdot, \ldots, \cdot, \widehat{C}^u)$.
- Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
- Output $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$.

$\underline{\mathsf{Dec}(\{\mathsf{SK}_{\mathsf{id},\mathsf{GID},x_{\mathsf{GID},\mathsf{id}}}\}_{\mathsf{id}\in[n]}, \mathsf{CT}) :}$

- Parse CT as $(\mathsf{ct}_u)_{u\in[N]}$ and for each $\mathsf{id} \in [n]$, $\mathsf{SK}_{\mathsf{GID},\mathsf{id},x_{\mathsf{id}}}$ as $(\mathbf{S}_{\mathsf{id}}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}}})$.
- If all $\mathbf{S}_{\mathsf{id}}$ are not the same, output $\perp$.
- Let $\mathbf{S} = \mathbf{S}_1$. For each $u \in \mathbf{S}$, $\widehat{y}^u_{\mathsf{GID}} \leftarrow \mathsf{1MAFE.Dec}(\{\mathsf{1MAFE.sk}_{id,u}\}_{\mathsf{id}\in[n]}, \mathsf{ct}_u)$.
- Output $y = \mathsf{dCSF.Decode}(\{\widehat{y}^u_{\mathsf{GID}}\}_{u\in\mathbf{S}}, \mathbf{S})$.

Lemma 10.2. *If* 1MAFE *is an adaptively secure* 1-GID MA-FE *scheme (Definition 5.1) and* dCSF *is an adaptively secure distributed client-server framework for* P/Poly *circuits (Definition 6.1), then the above construction is a partial adaptively secure* Q-GID MA-FE *for* P/Poly *circuits (Definition 10.1) in the random oracle model.*

Proof.

Admissible Adversary An admissible adversary for $\mathsf{parMA\text{-}FE}$ is a valid adversary for $\mathsf{MA\text{-}FE}$ scheme (Figure 2) and the adversary should submit the set of non-corrupted users $\widetilde{\mathbf{S}}$ and the indices $(j_1^*, \ldots, j_Q^*)$ at the beginning of the experiments.

$\underline{\mathsf{Expt}_0^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)}$: This is the Real experiment parameterized by adversary $\mathcal{A}$ and honest challenger $\mathcal{C}$.

1. **Setup:** The adversary $\mathcal{A}$ provides the query bound Q, the number of authorities n, the maximum size of the challenge circuit s, the set of non-corrupted users, $\widetilde{\mathbf{S}}$, and the set of unique indices $(j_1^*, \ldots, j_Q^*)$. The challenger $\mathcal{C}$ runs $\mathsf{crs} \leftarrow \mathsf{GlobalSetup}(1^\lambda, 1^Q, 1^n, 1^s, \widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$ and $\forall\ \mathsf{id} \in [n], (\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) \leftarrow \mathsf{AuthSetup}(\mathsf{id})$. $\mathcal{C}$ sends $\left(\mathsf{crs}, (\mathsf{MPK}_{\mathsf{id}})_{\mathsf{id}\in[n]}\right)$ to $\mathcal{A}$.

2. **Pre-Challenge Query Phase:** $\mathcal{A}$ sends $q \in [Q_1]$ key queries of the form $(\mathsf{GID}_q,\ \mathsf{id}_q,\ x_{\mathsf{GID}_q,\mathsf{id}_q})$ such that $\mathsf{id}_q \in [n]$. $\mathcal{C}$ runs $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} \leftarrow \mathsf{KeyGen}\ (\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, \mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$ and sends $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}$ to $\mathcal{A}$.

3. **Challenge Phase:** $\mathcal{A}$ sends an n-ary poly-size circuit C of maximum size s. $\mathcal{C}$ samples $\mathsf{CT} \leftarrow \mathsf{Enc}\ (\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$ and sends CT to $\mathcal{A}$.

4. **Post-Challenge Query Phase:** This is similar to Pre-Challenge Query Phase for $q \in [Q_1{+}1, Q]$.

5. **Check Phase:** Let $Q^* \le Q$ be the number of unique $\mathbf{S}_q$'s and Δ_q's. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ and $j_q^* \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Abort and output $\perp$ if either of these checks fail.

6. **Guess Phase:** $\mathcal{A}$ outputs guess b'. Output b'.

$\underline{\mathsf{Expt}_1^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{parMA\text{-}FE}}}(1^\lambda)}$: This is the Ideal experiment parameterized by $\mathcal{A}$ and stateful $\mathsf{sim}_{\mathsf{parMA\text{-}FE}}$.

1. **Setup:** The adversary $\mathcal{A}$ provides the query bound Q, the number of authorities n, the maximum size of the challenge circuit s, the set of non-corrupted users, $\widetilde{\mathbf{S}}$, and the set of unique indices $(j_1^*, \ldots, j_Q^*)$. Sample $\mathsf{crs} \leftarrow \mathsf{Sim}_{\mathsf{par}}\mathsf{MA\text{-}FE}(1^\lambda, 1^Q, 1^n, 1^s, \widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$ and $\mathsf{MPK}_{\mathsf{id}} \leftarrow \mathsf{sim}_{\mathsf{parMA\text{-}FE}}(\mathsf{id})$ for every $\mathsf{id} \in [n]$. Initiate a set $\mathbf{Q}$ to be empty. Send $\left(\mathsf{crs}, (\mathsf{MPK}_{\mathsf{id}})_{\mathsf{id}\in[n]}\right)$ to $\mathcal{A}$.

2. **Pre-Challenge Query Phase:** $\mathcal{A}$ sends $q \in [Q_1]$ key queries of the form $(\mathsf{GID}_q,\ \mathsf{id}_q,\ x_{\mathsf{GID}_q,\mathsf{id}_q})$ such that $\mathsf{id}_q \in [n]$. Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$. Run $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} \leftarrow \mathsf{sim}_{\mathsf{parMA\text{-}FE}}(\mathsf{id}_q, \mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$ and sends $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}$ to $\mathcal{A}$.

3. **Challenge Phase:** $\mathcal{A}$ sends an n-ary poly-size circuit C of maximum size s. Sample $\mathsf{CT} \leftarrow \mathsf{sim}_{\mathsf{parMA\text{-}FE}}\ (\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, 1^{|F|}, \mathbf{V})$ and sends CT to $\mathcal{A}$ where $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) \mid \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.

4. **Post-Challenge Query Phase:** This is similar to Pre-Challenge Query Phase for $q \in [Q_1 + 1, Q]$. In addition, $\mathsf{sim}_{\mathsf{parMA\text{-}FE}}$ takes as input $(X, C(X))$ where $X = (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n})$ if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$. Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$.

5. **Check Phase:** Let $Q^* \le Q$ be the number of unique $\mathbf{S}_q$'s and Δ_q's. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ and $j_q^* \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Abort and output $\perp$ if either of these checks fail.

6. **Guess Phase:** $\mathcal{A}$ outputs guess b'. Output b'.

Figure 7: Real and Ideal experiments for $\mathsf{parMA\text{-}FE}$.

Correctness. The correctness of the scheme follows from the correctness of $\mathsf{1MAFE}$, dCSF, and the $\mathcal{H}$ scheme.

Security. We show the security of the scheme through a series of hybrids and claims by using $\mathcal{H}$ as a programmable random oracle.

$\mathsf{Hyb}_0(1^\lambda)$: This is $\mathsf{Expt}_0^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathcal{C}}$ from Definition 10.1.

$\mathsf{Hyb}_1(1^\lambda)$: In this hybrid, we program $\mathcal{H}$ to output uniformly random strings and maintain consistency using dictionary $\mathbf{H}$. Thus all $\mathbf{S}$ and Δ are truly random strings.

$\mathsf{Hyb}_2(1^\lambda)$: In this hybrid, we check if $\mathbf{S}_q$'s and Δ_q's obey Lemmas 4.2 and 4.3. If not, we will abort.

$\mathsf{Hyb}_{3,j}(1^\lambda)$: for $j \in [N+1]$, we simulate the first $j-1$ instantiations of $\mathsf{1MAFE}$ using $\mathsf{sim}^u_{\mathsf{1MAFE}}$.

$\mathsf{Hyb}_4(1^\lambda)$: In this hybrid, we will simulate dCSF. This is $\mathsf{Expt}_1^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{parMA\text{-}FE}}}$ from Definition 10.1.

Claim 10.3. *Hybrids* Hyb_0 *and* Hyb_1 *are identically distributed.*

Proof. The proof of this claim follows from the definition of $\mathcal{H}$ which acts as a random oracle. □

Claim 10.4. *Hybrids* Hyb_1 *and* Hyb_2 *are statistically indistinguishable.*

Proof. The proof of this claim follows from Lemmas 4.2 and 4.3. □

Claim 10.5. *Hybrids* Hyb_2 *and* $\mathsf{Hyb}_{3,1}$ *are identically distributed.*

Proof. Note that in $\mathsf{Hyb}_{3,1}$, we do not simulate any instantiations of $\mathsf{1MAFE}$. Hence, the hybrids are identically distributed as there is no change between these hyrbids. □

Claim 10.6. *Assuming the adaptive security of* $\mathsf{1MAFE}$, $\mathsf{Hyb}_{3,j}$ *and* $\mathsf{Hyb}_{3,j+1}$ *for any* $j \in [N-1]$ *are computationally indistinguishable.*

Proof. The proof of this claim is similar to the proof of Claim 8.7. □

Claim 10.7. *Assuming the adaptive security of* $\mathsf{1MAFE}$, $\mathsf{Hyb}_{3,N}$ *and* $\mathsf{Hyb}_{3,N+1}$ *are computationally indistinguishable.*

Proof. The proof of this claim is similar to the proof of Claim 10.6. □

Claim 10.8. *Assuming the security of* dCSF, $\mathsf{Hyb}_{3,N+1}$ *and* Hyb_4 *are computationally indistinguishable.*

Proof. The proof of this claim is similar to the proof of Claim 8.8. □

For completeness, we provide full descriptions of these hybrids in Appendix G.

11 Adaptive Q-GID MA-FE for P/Poly in ROM

In this section, we provide the construction of adaptively secure Q-GID MA-FE scheme from sub-exponentially secure parMA-FE scheme. In the parMA-FE scheme, the security game requires the adversary to submit the set of non-corrupted users $\widetilde{\mathbf{S}}$ and the unique indices $(j_1^*, \ldots, j_Q^*)$. As mentioned previously, we created this scheme to demonstrate the feasibility of MA-FE by relying on sub-exponential security of parMA-FE. The size of the string $(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$ is polynomial in Q and λ and we can guess the string correctly with $1/2^{O(Q\lambda \log(Q\lambda))}$ probability. Hence, if we rely on sub-exponentially secure parMA-FE, we can set the security parameter such that parMA-FE is $\left(O(2^{Q^2\lambda^2}), O(2^{-(Q^2\lambda^2)})\right)$-partial adaptively secure. In this case, the reduction algorithm can guess the string $(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$ and be correct with probability $2^{O(Q\lambda \log(Q\lambda))} > 2^{-Q^2\lambda^2}$. The construction of the adaptively secure MA-FE scheme is similar to parMA-FE except we set the security parameter to $1^{(Q^2\lambda^2)^{1/\alpha}}$ where α is a specific constant.

The reduction algorithm ($\mathcal{B}$) between an adversary $\mathcal{A} = \mathcal{A}_{\mathsf{MA\text{-}FE}}$ with advantage ϵ and challenger $\mathcal{C} = \mathcal{C}_{\mathsf{parMA\text{-}FE}}$ is as follows.

- Uniformly sample $(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$ and submit it to $\mathcal{C}$. Receive crs from $\mathcal{C}$ and feed it to $\mathcal{A}$.
- Any query made by $\mathcal{A}$ to $\mathcal{C}$ or random oracle $\mathcal{H}$, pass it on to $\mathcal{C}$.
- $\mathcal{A}$ outputs guess b', output b'.

It looks like $\mathcal{B}$ will win with advantage at least $\epsilon/2^{O(Q\lambda \log(Q\lambda))} > 1/2^{Q^2\lambda^2}$. However, there is a subtle issue here. The string $(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$ guessed by $\mathcal{B}$ might not the consistent with the GID queries made by $\mathcal{A}$. That is, the set of non-corrupted users resulting from the GID queries made by $\mathcal{A}$ might not be the same as $\widetilde{\mathbf{S}}$ or the set of unique indices might not be same or both. If the string is indeed wrong, then even if the $\mathcal{A}$ wins with non-negligible probability, $\mathcal{C}$ aborts and renders the advantage of $\mathcal{B}$ to be 0. So, although $\mathcal{A}$ can break the MA-FE scheme, we cannot rely on $\mathcal{A}$ to break the parMA-FE scheme.

A similar issue also arises in construction of IBE schemes from paring-based groups as seen in [Wat05]. We utilize the Artificial Abort technique introduced by Waters [Wat05] and in particular, the advantage counting variant proposed by Freitag et al [FGH+17] to solve this issue. Using artificial abort, we can reduce the dependence between the events where $\mathcal{A}$'s wins and the $\mathcal{C}$ aborts. Artificial abort technique, as used in the context of this work, can be summarized as follows: if we have "Good" adversary, then it means that for a significant portion of random guesses of $(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$, $\mathcal{B}$ wins with 'large' probability. Hence, $\mathcal{B}$ can readily use it's initial guess against $\mathcal{C}$. On the other hand, if we have a "Bad" adversary, then $\mathcal{B}$ is better off guessing b' randomly. For the specific definitions of Good and Bad, we refer the reader to the proof of Theorem 11.1.

How does one determine whether $\mathcal{A}$ is Good or Bad? In order to determine this, we use the counting argument used in [FGH+17]. In this counting argument, we will run $\mathcal{A}$ independently for Λ-many times acting as a challenger for MA-FE and count the number of times $\mathcal{A}$ is successful. By setting Λ as a function of Q, λ, ϵ, we can rely on Chernoff's bound to determine whether $\mathcal{A}$ is Good or Bad. At this point, if we determine that $\mathcal{A}$ is Bad, then we artificially abort and randomly guess b'. Once we determine that $\mathcal{A}$ is Good, we can readily use $\mathcal{A}$ in the reduction algorithm as defined above. Note that depending on ϵ, the running time of this procedure might not be polynomial in Q, λ. However, all we require is that running time of $\mathcal{B}$ is sub-exponential in $Q^2\lambda^2$, which is satisfied by our reduction algorithm.

Construction. The construction of the MA-FE scheme closely follows the construction of parMA-FE. The only addition in MA-FE scheme is that in GlobalSetup, we use parMA-FE.GlobalSetup $\left(1^{(Q^2\lambda^2)^{1/\alpha}}, 1^Q, 1^n, 1^s\right)$ where α is a constant which is defined in the security analysis. The remaining algorithms, AuthSetup, KeyGen, Enc, Dec utilize parMA-FE identically.

Theorem 11.1. *If* parMA-FE *is a sub-exponentially secure partial adaptively secure* Q-GID MA-FE *scheme (Definition 10.1) for* P/Poly *circuits in the random oracle model, then the construction described above is* $(2^{O(Q\lambda \log(Q\lambda))}, 2^{-O(Q\lambda \log(Q\lambda))})$*-adaptively secure* Q-GID MA-FE *scheme (Definition 3.5) for* P/Poly *circuits in the random oracle model.*

Proof.

Correctness. The correctness of the scheme follows from the correctness of parMA-FE.

Security. Assume that there exists a valid adversary $\mathcal{A}$ for MA-FE. We we will construct an adversary that can break the security of parMA-FE. More formally, if $\mathcal{A}$'s advantage is such that

$$\mathsf{Adv}_{\mathcal{A}}^{\text{MA-FE}} = \left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\text{MA-FE},\mathcal{A},\mathcal{C}}\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\text{MA-FE},\mathcal{A},\mathsf{sim}_{\text{MA-FE}}}\right]\right| > \epsilon$$

for some ϵ. We will construct a reduction algorithm that can distinguish between Expt_0 and Expt_1 for the parMA-FE scheme. In other words,

$$\mathsf{Adv}_{\mathcal{B}}^{\text{parMA-FE}} = \left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\text{parMA-FE},\mathcal{B},\mathcal{C}}\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\text{parMA-FE},\mathcal{B},\mathsf{sim}_{\text{parMA-FE}}}\right]\right| > \frac{\epsilon}{2^{Q^2\lambda^2}}$$

Note that $\mathcal{B}$ cannot utilize the output of $\mathcal{A}$ directly. As per Definition 10.1 and Figure 7, $\mathcal{B}$ needs to submit the non-corrupted users $(\widetilde{\mathbf{S}})$ and the unique indices $(j_1^*, \ldots, j_Q^*)$. If these indices do not match the $\mathcal{A}$'s query values, $\mathcal{C}$ will abort. Hence, we rely on the artificial abort technique. The description of $\mathcal{B}$ is as follows. In the description, $\Lambda = O(Q^2\lambda^2/\epsilon^2)$.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda):}$

1. Uniformly sample $(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$.
2. Initiate counters x, y, initially to be 0.
3. For $i \in [\Lambda]$:
   - Sample $\beta \xleftarrow{\$} \{0,1\}$.
   - Run $\mathsf{AbortExpt}_\beta^{\mathcal{A},\mathcal{B}}(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$ as defined in Figure 8. If the experiment does not abort, increment the counter y.
   - If the experiment doesn't abort, let $\beta' \leftarrow \mathsf{AbortExpt}_\beta^{\mathcal{A},\mathcal{B}}(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$. Increment the counter x if $\beta = \beta'$.
4. Let $\gamma = x/\Lambda - 1/2$. If $\gamma < \epsilon/4$, abort and output $b' \xleftarrow{\$} \{0,1\}$.
5. $\mathcal{A}$ sends Q, n, s to $\mathcal{B}$. Submit $(Q^2\lambda^2)^{1/\alpha}, n, s, (\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)$ to $\mathcal{O}$ where α is a constant such that parMA-FE is $\left(O(2^{Q^2\lambda^2}), O(2^{-(Q^2\lambda^2)})\right)$-partial adaptively secure.
6. $\mathcal{B}$ receives $\mathsf{parMA\text{-}FE.crs}, (\mathsf{parMA\text{-}FE.mpk_{id}})_{\mathsf{id}\in[n]}$ from $\mathcal{O}$. Set $\mathsf{crs} = \mathsf{parMA\text{-}FE.crs}$, $\mathsf{MPK_{id}} = \mathsf{parMA\text{-}FE.mpk_{id}}$ for each $\mathsf{id} \in [n]$ and send $\mathsf{crs}, (\mathsf{MPK_{id}})_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
7. For any GID queries to $\mathcal{H}$, send the query to $\mathsf{parMA\text{-}FE}.\mathcal{H}$ and pass the result along to $\mathcal{A}$.
8. The remaining pre-challenge queries, challenge query, and post-challenge queries are passed as it is to $\mathcal{O}$ and the responses from $\mathcal{O}$ are passed in accordance to the construction to $\mathcal{A}$.
9. For each $q \in [Q^*]$, let $(\mathbf{S}_q, \Delta_q) = \mathsf{parMA\text{-}FE}.\mathcal{H}(\mathsf{GID}_q)$. If $\widetilde{\mathbf{S}} \neq [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ or for any $q \in [Q^*], j_q^* \notin \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$, abort and output $b' \xleftarrow{\$} \{0,1\}$. Otherwise, $\mathcal{A}$ outputs guess β'. Output $b' = \beta'$.

Note that the running time of $\mathcal{B}$ is still sub-exponential in the security parameter for $\mathcal{O}$, $(Q^2\lambda^2)^{1/\alpha}$. Hence, $\mathcal{B}$ is a valid adversary for parMA-FE as per Definition 10.1. To analyze the algorithm $\mathcal{B}$, we define the following events.

- Abort : $\gamma < \epsilon/4$ (This is the artificial abort in step 4).

$\underline{\mathsf{AbortExpt}_{\beta}^{\mathcal{A},\mathcal{B}}(\widetilde{\mathbf{S}}, j_1^*, \ldots, j_Q^*)}$

1. If $\beta = 0$, run $\mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{A},\mathcal{B}}$. Otherwise run $\mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{parMA\text{-}FE}}}$.
2. Let $\mathsf{GID}_1, \ldots, \mathsf{GID}_{Q^*}$ be the unique GID queries made by $\mathcal{A}$ in the previous experiment. Let $(\mathbf{S}_1, \Delta_1), \ldots, (\mathbf{S}_{Q^*}, \Delta_{Q^*})$ be the responses.
3. If for each $\widetilde{\mathbf{S}} \neq [N] \setminus \left(\bigcup_{q \neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'} \right)$ or for any $q \in [Q^*], j_q^* \notin \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'} \right)$, abort. Output $\perp$.
4. Otherwise, let β' be $\mathcal{A}$'s guess. Output β'.

Figure 8: Artificial Abort Experiment for $\mathcal{B}$.

- Good : $\Pr[\mathcal{A} \text{ wins}] \geq 1/2 + \epsilon/2$
- Bad : $\Pr[\mathcal{A} \text{ wins}] < 1/2$
- Mid : $1/2 \leq \Pr[\mathcal{A} \text{ wins}] < 1/2 + \epsilon/2$.
- Guess : $\widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q \neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'} \right)$ and for each $q \in [Q], j_q^* \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'} \right)$.

Using a combinatorial argument, we can show that $\Pr[\mathsf{Guess}] = 1/2^{O(Q\lambda \log(Q\lambda))}$ for the values of N, D, t, v, T from Lemmas 4.2 and 4.3. Let us denote $p = \Pr[\mathsf{Guess}]$. Now, $\Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}}] \geq 1/2 \cdot (1-p) + \Pr[\mathcal{A} \text{ wins}] \cdot p$. Thus,

- $\Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}} \wedge \mathsf{Good}] \geq 1/2 + \epsilon p/2$.
- $\Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}} \wedge \mathsf{Mid}] \geq 1/2$.

$$
\begin{aligned}
\Pr[\mathcal{B} \text{ wins}] &= \Pr[\mathcal{B} \text{ wins} \mid \mathsf{Abort}] \Pr[\mathsf{Abort}] + \Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}}] \Pr[\overline{\mathsf{Abort}}] \\
&\geq \frac{1}{2} \Pr[\mathsf{Abort}] + \Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}} \wedge \mathsf{Good}] \Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Good}] \\
&+ \Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}} \wedge \mathsf{Mid}] \Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Mid}] + \Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}} \wedge \mathsf{Bad}] \Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Bad}] \\
&\geq \frac{1}{2} \Pr[\mathsf{Abort}] + \Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}} \wedge \mathsf{Good}] \Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Good}] + \\
&+ \Pr[\mathcal{B} \text{ wins} \mid \overline{\mathsf{Abort}} \wedge \mathsf{Mid}] \left(\Pr[\overline{\mathsf{Abort}}] - \Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Good}] - \Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Bad}] \right) \\
&= \frac{1}{2} + \frac{\epsilon p}{2} \Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Good}] - \frac{1}{2} \Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Bad}]
\end{aligned}
$$

Claim 11.2. $\Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Good}] \geq 2^{-O(Q^2\lambda^2)}$

Proof. Consider the value $\Pr[\mathsf{Abort} \mid \mathsf{Good}] = \Pr[\gamma < \epsilon/4 \mid \mathsf{Good}] = \Pr[x < \Lambda(1/2 + \epsilon/4) \mid \mathsf{Good}]$. In the event Good, $\Pr[\mathcal{A} \text{ wins}] \geq 1/2 + \epsilon/2$. Using a Chernoff's bound argument, we can see that

$$\Pr[x < \Lambda(1/2 + \epsilon/4) \mid \mathsf{Good}] \leq 2^{-O(Q^2\lambda^2)}$$

Hence, $\Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Good}] = (1 - \Pr[\mathsf{Abort} \mid \mathsf{Good}]) \Pr[\mathsf{Good}] \geq 2^{-O(Q^2\lambda^2)}$. □

Claim 11.3. $\Pr[\overline{\mathsf{Abort}} \wedge \mathsf{Bad}] \leq 2^{-O(Q^2\lambda^2)}$

Proof. The proof of this claim also utilizes Chernoff's bound and is similar to proof of Claim 11.2. □

Using Claims 11.2 and 11.3, we can see that $\Pr[\mathcal{B} \text{ wins}] > 1/2 + \epsilon/2^{O(Q^2\lambda^2)}$. Hence, $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{parMA\text{-}FE}} > \epsilon/2^{O(Q^2\lambda^2)}$.

12 Result Statements

In this section, we provide restatements of all our theorems and provide the sufficient assumptions that yield MA-FE schemes we constructed in this paper.

Theorem 12.1. *Assuming the existence of* IND-CPA*-secure public key encryption (Definition A.1), there exists an adaptive, simulation secure multi-authority functional encryption for* P/Poly *circuits secure against corruption of secret keys for* one GID *(Definition 5.1).*

Proof. This follows from Theorem 5.2. Note that na1MAFE for P/Poly circuits is constructed from IND-CPA-secure PKE and Garb for P/Poly circuits which are implied by one-way functions [Yao86, BHR12]. The NCE required for our work can be constructed from IND-CPA-secure PKE [GVW12, HMNY22, HKM+23]. □

Theorem 12.2. *Assuming the existence of* IND-CPA*-secure public key encryption (Definition A.1), there exists a simulation secure multi-authority functional encryption for* P/Poly *circuits secure against bounded Q queries to n authorities queried statically (Definition 7.1) where Q, n are polynomials in the security parameter.*

Proof. This follows from Theorem 7.2 and as PRFs (Definition 3.1) can be constructed from one-way functions [GGM86]. □

Corollary 12.3. *Assuming the hardness of Decisional Diffie-Hellman assumption (*DDH*) or Bilinear Decisional Diffie-Hellman assumption (*BDDH*), there exists an adaptive simulation secure multi-authority functional encryption scheme for* P/Poly *circuits (Definition 3.5) with $n = O(1)$ authorities respectively.*

Proof. This follows from Theorems 8.1 and 9.1. We instantiate 2 or 3 authority MA-FE for P/Poly circuits using the niKE scheme based on DDH [DH76b] or BDDH [Jou04] respectively. We then use our bootstrapping compiler from Section 9 to obtain MA-FE for P/Poly circuits that can handle any $O(1)$ authorities. □

Theorem 12.4. *Assuming the existence of sub-exponentially* IND-CPA*-secure public key encryption and sub-exponentially secure one-way functions, there exists an adaptive simulation secure multi-authority functional encryption scheme for* P/Poly *circuits (Definition 3.5) secure against a-priori bounded Q queries for secret keys across n authorities and secure against non-uniform $2^{O(Q\lambda \log(Q\lambda))}$-size circuits.*

Proof. This follows from Lemma 10.2 and Theorem 11.1. Note that we require sub-exponential PRGs to sub-exponentially secure CorrGarb scheme. □

Remark 12.5. *We stress that all of our Q-*GID *constructions from Sections 7, 8, 9, 10, and 11, can handle input sizes of arbitrary lengths. That is for any* $\mathsf{id} \in [n], x_{\mathsf{id}} \in \{0,1\}^{\ell_{\mathsf{id}}(\lambda)}$.

References

[AG21] Miguel Ambrona and Romain Gay. Multi-authority abe, revisited. *Cryptology ePrint Archive*, 2021.

[AGT21] Shweta Agrawal, Rishab Goyal, and Junichi Tomida. Multi-party functional encryption. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II*, pages 224–255. Springer, 2021.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 500–518. Springer, 2013.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *computational complexity*, 15(2):115–162, 2006.

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

[AJL+19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: new paradigms via low degree weak pseudorandomness and security amplification. In *Annual International Cryptology Conference*, pages 284–332. Springer, 2019.

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

[AMVY21] Shweta Agrawal, Monosij Maitra, Narasimha Sai Vempati, and Shota Yamada. Functional encryption for turing machines with dynamic bounded collusion from lwe. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part IV 41*, pages 239–269. Springer, 2021.

[AR17] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In *Theory of Cryptography Conference*, pages 173–205. Springer, 2017.

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

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, pages 223–238. Springer, 2004.

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In *Annual International Cryptology Conference*, pages 67–98. Springer, 2017.

[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.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[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.

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6, 2012.

[BGW88] Michael BenOr, S Goldwasser, and Avi Wigderson. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, pages 1–10, 1988.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 784–796, 2012.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

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

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324(1):71–90, 2003.

[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.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *34rd Annual International Cryptology Conference, CRYPTO 2014*, pages 480–499. Springer Verlag, 2014.

[CC09] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *ACM Conference on Computer and Communications Security*, pages 121–130, 2009.

[Cha07] Melissa Chase. Multi-authority attribute based encryption. In *Theory of Cryptography: 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007. Proceedings 4*, pages 515–534. Springer, 2007.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding: 8th IMA International Conference Cirencester, UK, December 17–19, 2001 Proceedings 8*, pages 360–363. Springer, 2001.

[DH76a] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[DH76b] Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, pages 109–112, 1976.

[DKW21] Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority abe for dnf s from lwe. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 177–209. Springer, 2021.

[DKW23] Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority abe for nc 1 from bdh. *Journal of Cryptology*, 36(2):6, 2023.

[DKXY02] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, 2002.

[DP23] Pratish Datta and Tapas Pal. Decentralized multi-authority attribute-based inner-product fe: Large universe and unbounded. In *IACR International Conference on Public-Key Cryptography*, pages 587–621. Springer, 2023.

[FGH+17] Cody Freitag, Rishab Goyal, Susan Hohenberger, Venkata Koppula, Eysa Lee, Tatsuaki Okamoto, Jordan Tran, and Brent Waters. Signature schemes with randomized verification. In *International Conference on Applied Cryptography and Network Security*, pages 373–389. Springer, 2017.

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[GGL24] Rachit Garg, Rishab Goyal, and George Lu. Dynamic collusion functional encryption and multi-authority attribute-based encryption. In *IACR International Conference on Public-Key Cryptography*, pages 69–104. Springer, 2024.

[GGLW22] Rachit Garg, Rishab Goyal, George Lu, and Brent Waters. Dynamic collusion bounded functional encryption from identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 736–763. Springer, 2022.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

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

[GLW12] Shafi Goldwasser, Allison Lewko, and David A Wilson. Bounded-collusion ibe from key homomorphism. In *Theory of Cryptography: 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings 9*, pages 564–581. Springer, 2012.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.

[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.

[HKM+23] Taiga Hiroka, Fuyuki Kitagawa, Tomoyuki Morimae, Ryo Nishimaki, Tapas Pal, and Takashi Yamakawa. Certified everlasting secure collusion-resistant functional encryption. Technical report, and more. Cryptology ePrint Archive, Report 2023/236, 2023.

[HMNY22] Taiga Hiroka, Tomoyuki Morimae, Ryo Nishimaki, and Takashi Yamakawa. Certified everlasting functional encryption. *arXiv preprint arXiv:2207.13878*, 2022.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from lpn over f p, dlin, and prgs in nc 0. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 670–699. Springer, 2022.

[Jou04] Antoine Joux. A one round protocol for tripartite diffie–hellman. *Journal of cryptology*, 17(4):263–276, 2004.

[Kim19] Sam Kim. Multi-authority attribute-based encryption from lwe in the ot model. *Cryptology ePrint Archive*, 2019.

[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.

[Lin17] Huijia Lin. Indistinguishability obfuscation from sxdh on 5-linear maps and locality-5 prgs. In *Annual International Cryptology Conference*, pages 599–629. Springer, 2017.

[LW11] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.

[MJ18] Yan Michalevsky and Marc Joye. Decentralized policy-hiding abe with receiver privacy. In *Computer Security: 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II 23*, pages 548–567. Springer, 2018.

[OT20] Tatsuaki Okamoto and Katsuyuki Takashima. Decentralized attribute-based encryption and signatures. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 103(1):41–73, 2020.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology: Proceedings of CRYPTO 84 4*, pages 47–53. Springer, 1985.

[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.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 475–484, 2014.

[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.

[WFL19] Zhedong Wang, Xiong Fan, and Feng-Hao Liu. Fe for inner products and its application to decentralized abe. In *IACR international workshop on public key cryptography*, pages 97–127. Springer, 2019.

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

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pages 162–167. IEEE, 1986.

A Additional Preliminaries

A.1 Public Key Encryption

A public key encryption (PKE) scheme with message space $\mathbf{M} = \{\mathbf{M}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of the following algorithms.

$\underline{\mathsf{Setup}(1^\lambda) \to (\mathsf{PK}, \mathsf{SK})}$: This is a randomized algorithm that on input the security parameter λ, outputs the public and secret key pair $(\mathsf{PK}, \mathsf{SK})$.

$\underline{\mathsf{Enc}(\mathsf{PK}, m) \to \mathsf{CT}}$: This is a randomized algorithm that on input the public key PK and the message $m \in \mathbf{M}_\lambda$, outputs the corresponding ciphertext CT.

$\underline{\mathsf{Dec}(\mathsf{SK}, \mathsf{CT}) \to m'}$: This is a polynomial time algorithm that on input the secret key SK and the ciphertext CT, outputs m'.

Definition A.1. *The scheme* $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *is said to be IND-CPA secure public key encryption scheme if it satisfies the following properties.*

1. ***Correctness:*** *We say that* PKE *is correct if for any* $\lambda \in \mathbb{N}$, $m \in \mathbf{M}_\lambda$,

$$\Pr\left[m' = m : \begin{array}{c}(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Setup}(1^\lambda),\\ \mathsf{CT} \leftarrow \mathsf{Enc}(\mathsf{PK}, m),\\ m' \leftarrow \mathsf{Dec}(\mathsf{SK}, \mathsf{CT})\end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

2. ***IND-CPA Security:*** *We say that* PKE *is IND-CPA secure if for any stateful PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$,

$$\Pr\left[b = b' : \begin{array}{c}b \xleftarrow{\$} \{0,1\},\\ (\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{Setup}(1^\lambda), (m, \mathsf{state}) \leftarrow \mathcal{A}_0(1^\lambda, \mathsf{PK}),\\ \mathsf{CT}^{(0)} \leftarrow \mathsf{Enc}(\mathsf{PK}, m), \mathsf{CT}^{(1)} \leftarrow \mathsf{Enc}(\mathsf{PK}, 0^{|m|}),\\ b' \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{CT}^{(b)})\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

A.2 Garbled Circuits

This definition of garbled circuits [Yao86] is derived from [BHR12]. For any $\mathsf{P/Poly}$ circuit C that takes n-bit inputs, a garbling scheme (Garble) consists of the following algorithms.

$\underline{\mathsf{Garb}(1^\lambda, C) \to (\mathsf{GC}, \{w_{i,b}\}_{i\in[n],b\in\{0,1\}})}$: This is a randomized algorithm that on input the security parameter λ and description of the circuit C, outputs a garbled circuit GC and $2n$ wire keys $\{w_{i,b}\}_{i\in[n],b\in\{0,1\}}$, where n is the number of input bits for C.

$\underline{\mathsf{Eval}(\mathsf{GC}, \{w_{i,x_i}\}_{i\in[n]}) \to y}$: This is a polynomial time algorithm that on input the garbled circuit GC and n wire keys $\{w_{i,x_i}\}_{i\in[n]}$, outputs the value y.

Definition A.2. *The scheme* $\mathsf{Garble} = (\mathsf{Garb}, \mathsf{Eval})$ *is said to be a secure garbling scheme if satisfies the following properties.*

- ***Correctness:*** *We say that* Garble *is correct if for any* $\lambda \in \mathbb{N}$, *n-ary* $\mathsf{P/Poly}$ *circuit* C,

$$\Pr\left[y = C(x_1, \ldots, x_n) : \begin{array}{c}(\mathsf{GC}, \{w_{i,b}\}_{i\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garb}(1^\lambda, C)\\ y = \mathsf{Eval}(\mathsf{GC}, \{w_{i,x_i}\}_{i\in[n]})\end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

- ***Security:*** *We say that* Garble *is secure if any stateful PPT adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$,

$$\Pr\left[b' = b : \begin{array}{c} b \xleftarrow{\$} \{0,1\}, \\ (\mathsf{state}, C, x) \leftarrow \mathcal{A}_0(1^\lambda), \\ (\mathsf{GC}^{(0)}, \{w^{(0)}_{i,\beta}\}_{i\in[n],\beta\in\{0,1\}}) \leftarrow \mathsf{Garb}(1^\lambda, C) \\ (\mathsf{GC}^{(1)}, \{w^{(1)}_{i,x_i}\}_{i\in[n]}) \leftarrow \mathsf{sim}_{\mathsf{Garble}}(1^\lambda, 1^n, 1^{|C|}, C(x)) \\ b' \leftarrow \mathcal{A}_1(\mathsf{state}, \mathsf{GC}^{(b)}, \{w^{(b)}_{i,x_i}\}_{i\in[n]}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

B Proofs from Section 5

In this section, we provide the missing proofs from Section 5.

B.1 Hybrid Descriptions

The description of hybrids is as follows.

$\underline{\mathsf{Hyb}_0(1^\lambda):}$ This is $\mathsf{Expt}_0^{\mathsf{1MAFE},\mathcal{A},\mathcal{C}}$ from Definition 5.1.

1. $\mathcal{A}$ sends the number of authorities n to $\mathcal{C}$. $\mathcal{C}$ does the following:
   - Sample $\mathsf{na1MAFE.crs} \leftarrow \mathsf{na1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{na1MAFE.mpk_{id}}, \mathsf{na1MAFE.msk_{id}}) \leftarrow \mathsf{na1MAFE.AuthSetup}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{NCE.mpk_{id}}, \mathsf{NCE.msk_{id}}) \leftarrow \mathsf{NCE.Setup}(1^\lambda, 1^\kappa)$.
   - Set $\mathsf{MPK_{id}} = (\mathsf{na1MAFE.mpk_{id}}, \mathsf{NCE.mpk_{id}})$ and $\mathsf{MSK_{id}} = (\mathsf{na1MAFE.msk_{id}}, \mathsf{NCE.msk_{id}})$.

   Send $\mathsf{crs} = (\mathsf{na1MAFE.crs})$ and $\{\mathsf{MPK_{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq n$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, $\mathcal{C}$ runs:
   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{na1MAFE.msk}_{\mathsf{id}_q}, \{\mathsf{na1MAFE.mpk}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.
3. $\mathcal{A}$ sends a n-ary P/Poly circuit C. $\mathcal{C}$ does the following:
   - Sample $\mathsf{na1MAFE.ct} \leftarrow \mathsf{na1MAFE.Enc}(\{\mathsf{na1MAFE.mpk_{id}}\}_{\mathsf{id}\in[n]}, C)$.
   - For each $\mathsf{id} \in [n]$, sample a random string $R_{\mathsf{id}} \xleftarrow{\$} \{0,1\}^\kappa$.
   - For each $\mathsf{id} \in [n]$, $\mathsf{NCE.ct_{id}} \leftarrow \mathsf{NCE.Enc}(\mathsf{NCE.mpk_{id}}, R_{\mathsf{id}})$.

   Send $\mathsf{CT} = \left((\mathsf{NCE}.ct_{\mathsf{id}})_{\mathsf{id}\in[n]}, \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus \mathsf{na1MAFE.ct}\right)$ to $\mathcal{A}$.
4. $\mathcal{A}$ makes at most $n - Q_1$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, $\mathcal{C}$ runs:
   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{na1MAFE.msk}_{\mathsf{id}_q}, \{\mathsf{na1MAFE.mpk}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.
5. $\mathcal{A}$ outputs b'. Output b'.

$\underline{\mathsf{Hyb}_{1,j}(1^\lambda) \text{ for } j \in [n+1]}$: In this hybrid, we will fake the first $j-1$ NCE instantiations if they are queried adaptively. The changes are highlighted in red.

1. $\mathcal{A}$ sends the number of authorities n. Do the following:
   - Sample $\mathsf{na1MAFE.crs} \leftarrow \mathsf{na1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{na1MAFE.mpk_{id}}, \mathsf{na1MAFE.msk_{id}}) \leftarrow \mathsf{na1MAFE.AuthSetup}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{NCE.mpk_{id}}, \mathsf{NCE.msk_{id}}) \leftarrow \mathsf{NCE.Setup}(1^\lambda, 1^\kappa)$.
   - Set $\mathsf{MPK_{id}} = (\mathsf{na1MAFE.mpk_{id}}, \mathsf{NCE.mpk_{id}})$ and $\mathsf{MSK_{id}} = (\mathsf{na1MAFE.msk_{id}}, \mathsf{NCE.msk_{id}})$.

   Send $\mathsf{crs} = (\mathsf{na1MAFE.crs})$ and $\{\mathsf{MPK_{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. Initiate dictionary $\mathbf{Q}$ to be initially empty.
3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq n$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:
   - Sample $\mathsf{na1MAFE.SK_{GID,id_q},x_{id_q}} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{na1MAFE.msk_{id_q}}, \{\mathsf{na1MAFE.mpk_{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - $\mathsf{NCE.sk_{id_q}} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk_{id_q}})$.

   Add $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$ to $\mathbf{Q}$. Send $\mathsf{SK_{GID,id_q},x_{id_q}} = (\mathsf{na1MAFE.SK_{GID,id_q},x_{id_q}}, \mathsf{NCE.sk_{id_q}})$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends a n-ary P/Poly circuit C. $\mathcal{C}$ does the following: For each $\mathsf{id} \in [n]$, sample $R_{\mathsf{id}} \xleftarrow{\$} \{0,1\}^\kappa$ and
   - If $\mathsf{id} < j$ and $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$, $(\mathsf{NCE.ct_{id}}, \mathsf{NCE.aux_{id}}) \leftarrow \mathsf{NCE.Fake}(\mathsf{NCE.mpk_{id}})$.
   - Otherwise, $\mathsf{NCE.ct_{id}} \leftarrow \mathsf{NCE.Enc}(\mathsf{NCE.mpk_{id}}, R_{\mathsf{id}})$.
   - Sample $\mathsf{na1MAFE.ct} \leftarrow \mathsf{na1MAFE.Enc}(\{\mathsf{na1MAFE.mpk_{id}}\}_{\mathsf{id}\in[n]}, C)$ and $\widetilde{R} = \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus \mathsf{na1MAFE.ct}$.

   Send $\mathsf{CT} = (\{\mathsf{NCE.ct_{id}}\}_{\mathsf{id}\in[n]}, \widetilde{R})$ to $\mathcal{A}$.
5. $\mathcal{A}$ makes at most $n - Q_1$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:
   - Sample $\mathsf{na1MAFE.SK_{GID,id_q},x_{id_q}} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{na1MAFE.msk_{id_q}}, \{\mathsf{na1MAFE.mpk_{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - If $\mathsf{id} < j$ and $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$, $\mathsf{NCE.sk_{id_q}} \leftarrow \mathsf{NCE.Reveal}(\mathsf{NCE.msk_{id_q}}, \mathsf{NCE.aux_{id_q}}, R_{\mathsf{id}_q})$.
   - Otherwise, $\mathsf{NCE.sk_{id_q}} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk_{id_q}})$.

   Send $\mathsf{SK_{GID,id_q},x_{id_q}} = (\mathsf{na1MAFE.SK_{GID,id_q},x_{id_q}}, \mathsf{NCE.sk_{id_q}})$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs b'. Output b'.

$\underline{\mathsf{Hyb}_2(1^\lambda)}$: In this hybrid, all the NCE instantiations are faked appropriately and we will change the way ciphertext is generated and revealed through NCE. The changes are highlighted in red.

1. $\mathcal{A}$ sends the number of authorities n. Do the following:
   - Sample $\mathsf{na1MAFE.crs} \leftarrow \mathsf{na1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{na1MAFE.mpk_{id}}, \mathsf{na1MAFE.msk_{id}}) \leftarrow \mathsf{na1MAFE.AuthSetup}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{NCE.mpk_{id}}, \mathsf{NCE.msk_{id}}) \leftarrow \mathsf{NCE.Setup}(1^\lambda, 1^\kappa)$.
   - Set $\mathsf{MPK_{id}} = (\mathsf{na1MAFE.mpk_{id}}, \mathsf{NCE.mpk_{id}})$ and $\mathsf{MSK_{id}} = (\mathsf{na1MAFE.msk_{id}}, \mathsf{NCE.msk_{id}})$.

   Send $\mathsf{crs} = (\mathsf{na1MAFE.crs})$ and $\{\mathsf{MPK_{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. Initiate dictionary $\mathbf{Q}$ to be initially empty.
3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq n$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:

- Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{na1MAFE.msk}_{\mathsf{id}_q}, \{\mathsf{na1MAFE.mpk}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}_q})$.
- $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}_q})$.

Add $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$ to $\mathbf{Q}$. Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.

4. $\mathcal{A}$ sends a n-ary P/Poly circuit C. $\mathcal{C}$ does the following: For each $\mathsf{id} \in [n]$,
   - If $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$, $(\mathsf{NCE.ct}_{\mathsf{id}}, \mathsf{NCE.aux}_{\mathsf{id}}) \leftarrow \mathsf{NCE.Fake}(\mathsf{NCE.mpk}_{\mathsf{id}})$.
   - Otherwise, $\mathsf{NCE.ct}_{\mathsf{id}} \leftarrow \mathsf{NCE.Enc}(\mathsf{NCE.mpk}_{\mathsf{id}}, R_{\mathsf{id}})$ where $R_{\mathsf{id}} \xleftarrow{\$} \{0,1\}^\kappa$.
   - Sample $\mathsf{na1MAFE.ct} \leftarrow \mathsf{na1MAFE.Enc}(\{\mathsf{na1MAFE.mpk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$. If $|\mathbf{Q}| = n$, $\widetilde{R} = \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus \mathsf{na1MAFE.ct}$. Otherwise, $\widetilde{R} \xleftarrow{\$} \{0,1\}^\kappa$.

   Send $\mathsf{CT} = (\{\mathsf{NCE.ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \widetilde{R})$ to $\mathcal{A}$.
5. $\mathcal{A}$ makes at most $n - Q_1$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:
   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{na1MAFE.msk}_{\mathsf{id}_q}, \{\mathsf{na1MAFE.mpk}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - If $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$,
     - If this is the last authority query for GID, $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.Reveal}(\mathsf{NCE.msk}_{\mathsf{id}_q}, \bigoplus_{\mathsf{id}\neq\mathsf{id}_q} R_{\mathsf{id}} \oplus \widetilde{R} \oplus \mathsf{na1MAFE.ct})$ where $\mathsf{na1MAFE.ct} \leftarrow \mathsf{na1MAFE.Enc}(\{\mathsf{na1MAFE.mpk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$.
     - Otherwise, $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.Reveal}(\mathsf{NCE.msk}_{\mathsf{id}_q}, \mathsf{NCE.aux}_{\mathsf{id}_q}, R_{\mathsf{id}_q})$ where $R_{\mathsf{id}_q} \xleftarrow{\$} \{0,1\}^\kappa$.

   Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs b'. Output b'.

$\underline{\mathsf{Hyb}_3(1^\lambda)}$: This is $\mathsf{Expt}_1^{\mathsf{1MAFE},\mathcal{A},\mathsf{sim}_{\mathsf{1MAFE}}}$ from Definition 5.1. The changes are highlighted in red.

1. $\mathcal{A}$ sends the number of authorities n. Do the following:
   - Sample $\mathsf{na1MAFE.crs} \leftarrow \mathsf{sim}_{\mathsf{na1MAFE}}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n]$, sample $\mathsf{na1MAFE.mpk}_{\mathsf{id}} \leftarrow \mathsf{na1MAFE.AuthSetup}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{NCE.mpk}_{\mathsf{id}}, \mathsf{NCE.msk}_{\mathsf{id}}) \leftarrow \mathsf{NCE.Setup}(1^\lambda, 1^\kappa)$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{na1MAFE.mpk}_{\mathsf{id}}, \mathsf{NCE.mpk}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = \mathsf{NCE.msk}_{\mathsf{id}}$.

   Send $\mathsf{crs} = (\mathsf{na1MAFE.crs})$ and $\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. Initiate dictionary $\mathbf{Q}$ to be initially empty.
3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq n$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:
   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_{\mathsf{na1MAFE}}(\mathsf{id}_q, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}_q})$.

   Add $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$ to $\mathbf{Q}$. Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends a n-ary P/Poly circuit C. $\mathcal{C}$ does the following: For each $\mathsf{id} \in [n]$,
   - If $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$, $(\mathsf{NCE.ct}_{\mathsf{id}}, \mathsf{NCE.aux}_{\mathsf{id}}) \leftarrow \mathsf{NCE.Fake}(\mathsf{NCE.mpk}_{\mathsf{id}})$.
   - Otherwise, $\mathsf{NCE.ct}_{\mathsf{id}} \leftarrow \mathsf{NCE.Enc}(\mathsf{NCE.mpk}_{\mathsf{id}}, R_{\mathsf{id}})$ where $R_{\mathsf{id}} \xleftarrow{\$} \{0,1\}^\kappa$.

- If $|\mathbf{Q}| = n$, $\widetilde{R} = \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus \mathsf{na1MAFE.ct}$ where $\mathsf{na1MAFE.ct} \leftarrow \mathsf{sim}_{\mathsf{na1MAFE}}(1^{|C|}, (x_1, \ldots, x_n), C(x_1, \ldots, x_n))$.
  Otherwise, $\widetilde{R} \xleftarrow{\$} \{0,1\}^\kappa$.

Send $\mathsf{CT} = (\{\mathsf{NCE.ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \widetilde{R})$ to $\mathcal{A}$.

5. $\mathcal{A}$ makes at most $n - Q_1$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:
   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_{\mathsf{na1MAFE}}(\mathsf{id}_q, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - If $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$,
     - If this is the last authority query for GID, $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.Reveal}(\mathsf{NCE.msk}_{\mathsf{id}_q}, \bigoplus_{\mathsf{id}\neq\mathsf{id}_q} R_{\mathsf{id}} \oplus \widetilde{R} \oplus \mathsf{na1MAFE.ct})$ where $\mathsf{na1MAFE.ct} \leftarrow \mathsf{sim}_{\mathsf{na1MAFE}}(1^{|C|}, (x_1, \ldots, x_n), C(x_1, \ldots, x_n))$.
     - Otherwise, $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.Reveal}(\mathsf{NCE.msk}_{\mathsf{id}_q}, \mathsf{NCE.aux}_{\mathsf{id}_q}, R_{\mathsf{id}_q})$ where $R_{\mathsf{id}_q} \xleftarrow{\$} \{0,1\}^\kappa$.

   Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs b'. Output b'.

B.2 Proofs of Claims

The proofs of hybrids from Section 5 are as follows.

Proof of Claim 5.4. Note that for any $j \in [n]$, the only difference between $\mathsf{Hyb}_{1,j}$ and $\mathsf{Hyb}_{1,j+1}$ is that we fake the j-th instantiation of NCE if the j-th authority is queried in the post-challenge query phase. If $\mathcal{A}$ queries the j-th authority in the pre-challenge query phase, both the hybrids remain identical. Hence, w.l.o.g, assume that the j-th authority is only queried in the post-challenge query phase. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{1,j}$ and $\mathsf{Hyb}_{1,j+1}$, we will construct an adversary $\mathcal{B}$ that can break the security of NCE. More formally, if $\mathcal{A}$'s advantage is such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_{1,j}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_{1,j+1}}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction which distinguishes between oracle access between Expt_0 and Expt_1 for the NCE scheme. More formally,

$$\left|\Pr\left[1 \leftarrow \mathcal{B}^{\mathsf{NCE},0}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathcal{B}^{\mathsf{NCE},1}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda):}$

1. $\mathcal{A}$ sends the number of authorities n. Do the following:
   - Sample $\mathsf{na1MAFE.crs} \leftarrow \mathsf{na1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{na1MAFE.mpk}_{\mathsf{id}}, \mathsf{na1MAFE.msk}_{\mathsf{id}}) \leftarrow \mathsf{na1MAFE.AuthSetup}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n] \setminus \{j\}$, sample $(\mathsf{NCE.mpk}_{\mathsf{id}}, \mathsf{NCE.msk}_{\mathsf{id}}) \leftarrow \mathsf{NCE.Setup}(1^\lambda, 1^\kappa)$ Set $\mathsf{NCE.mpk}_j \leftarrow \mathcal{O}(\kappa)$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{na1MAFE.mpk}_{\mathsf{id}}, \mathsf{NCE.mpk}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\mathsf{na1MAFE.msk}_{\mathsf{id}}, \mathsf{NCE.msk}_{\mathsf{id}})$ for $\mathsf{id} \neq j$ and $\mathsf{MSK}_j = \mathsf{na1MAFE.msk}_j$.

   Send $\mathsf{crs} = (\mathsf{na1MAFE.crs})$ and $\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. Initiate dictionary $\mathbf{Q}$ to be initially empty.
3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq n$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:
   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{na1MAFE.msk}_{\mathsf{id}_q}, \{\mathsf{na1MAFE.mpk}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}_q})$.

- $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}_q})$.

Add $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$ to $\mathbf{Q}$. Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.

4. $\mathcal{A}$ sends a n-ary P/Poly circuit C. Do the following: For each $\mathsf{id} \in [n]$, sample $R_{\mathsf{id}} \xleftarrow{\$} \{0,1\}^\kappa$, and

   - If $\mathsf{id} < j$ and $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$, $(\mathsf{NCE.ct}_{\mathsf{id}}, \mathsf{NCE.aux}_{\mathsf{id}}) \leftarrow \mathsf{NCE.Fake}(\mathsf{NCE.mpk}_{\mathsf{id}})$.
   - Otherwise, if $\mathsf{id} = j$, $(\mathsf{NCE.ct}_{\mathsf{id}}, \mathsf{NCE.sk}_{\mathsf{id}}) \leftarrow \mathcal{O}(R_{\mathsf{id}})$.
   - Otherwise, $\mathsf{NCE.ct}_{\mathsf{id}} \leftarrow \mathsf{NCE.Enc}(\mathsf{NCE.mpk}_{\mathsf{id}}, R_{\mathsf{id}})$.
   - Sample $\mathsf{na1MAFE.ct}_{\mathsf{id}} \leftarrow \mathsf{na1MAFE.Enc}(\{\mathsf{na1MAFE.mpk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, C)$ and $\widetilde{R} = \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus \mathsf{na1MAFE.ct}$.

   Send $\mathsf{CT} = (\{\mathsf{NCE.ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \widetilde{R})$ to $\mathcal{A}$.

5. $\mathcal{A}$ makes at most $n - Q_1$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:

   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathsf{na1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{na1MAFE.msk}_{\mathsf{id}_q}, \{\mathsf{na1MAFE.mpk}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - If $\mathsf{id} < j$ and $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$, $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.Reveal}(\mathsf{NCE.msk}_{\mathsf{id}_q}, \mathsf{NCE.aux}_{\mathsf{id}_q}, R_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id} > j$, $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.

6. $\mathcal{A}$ outputs b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ, n and as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is Expt_0, $\mathcal{B}$ behaves like $\mathsf{Hyb}_{1,j}$ and if $\mathcal{O}$ is Expt_1, $\mathcal{B}$ behaves like $\mathsf{Hyb}_{1,j+1}$. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between Expt_0 and Expt_1 for NCE. This contradicts our assumption of adaptively secure NCE scheme. Hence, $\mathsf{Hyb}_{1,j}$ and $\mathsf{Hyb}_{1,j+1}$ are computationally indistinguishable. □

Proof of Claim 5.6. Note that if the adversary does not query all the authorities, the na1MAFE ciphertext is perfectly hidden. w.l.o.g, assume that the adversary queries all the authorities. In this case, the only difference between Hyb_2 and Hyb_3 is that we simulate the na1MAFE instantiation in Hyb_3. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids Hyb_2 and Hyb_3, we will construct an adversary $\mathcal{B}$ that can break the security of 1MAFE. More formally, if $\mathcal{A}$'s advantage is such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_2}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_3}\right]\right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction which distinguishes between oracle access between Expt_0 and Expt_1 for the na1MAFE scheme. More formally,

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathsf{na1MAFE},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathsf{na1MAFE},\mathcal{B},\mathsf{sim}_{\mathsf{1MAFE}}}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda):}$

1. $\mathcal{A}$ sends the number of authorities n. Do the following:

   - Sample $\mathsf{na1MAFE.crs}, \{\mathsf{na1MAFE.mpk}_{\mathsf{id}}\}_{\mathsf{id}\in[n]} \leftarrow \mathcal{O}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n]$, sample $\mathsf{na1MAFE.mpk}_{\mathsf{id}} \leftarrow \mathsf{na1MAFE.AuthSetup}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, sample $(\mathsf{NCE.mpk}_{\mathsf{id}}, \mathsf{NCE.msk}_{\mathsf{id}}) \leftarrow \mathsf{NCE.Setup}(1^\lambda, 1^\kappa)$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{na1MAFE.mpk}_{\mathsf{id}}, \mathsf{NCE.mpk}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = \mathsf{NCE.msk}_{\mathsf{id}}$.

   Send $\mathsf{crs} = (\mathsf{na1MAFE.crs})$ and $\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.

2. Initiate dictionary $\mathbf{Q}$ to be initially empty.
3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq n$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:
   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathcal{O}(\mathsf{id}_q, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.KeyGen}(\mathsf{NCE.msk}_{\mathsf{id}_q})$.

   Add $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$ to $\mathbf{Q}$. Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends a n-ary $\mathsf{P/Poly}$ circuit C. $\mathcal{C}$ does the following: For each $\mathsf{id} \in [n]$,
   - If $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$, $(\mathsf{NCE.ct}_{\mathsf{id}}, \mathsf{NCE.aux}_{\mathsf{id}}) \leftarrow \mathsf{NCE.Fake}(\mathsf{NCE.mpk}_{\mathsf{id}})$.
   - Otherwise, $\mathsf{NCE.ct}_{\mathsf{id}} \leftarrow \mathsf{NCE.Enc}(\mathsf{NCE.mpk}_{\mathsf{id}}, R_{\mathsf{id}})$ where $R_{\mathsf{id}} \xleftarrow{\$} \{0,1\}^\kappa$.
   - If $|\mathbf{Q}| = n$, $\widetilde{R} = \bigoplus_{\mathsf{id}\in[n]} R_{\mathsf{id}} \oplus \mathsf{na1MAFE.ct}$ where $\mathsf{na1MAFE.ct} \leftarrow \mathcal{O}(C)$. Otherwise, $\widetilde{R} \xleftarrow{\$} \{0,1\}^\kappa$.

   Send $\mathsf{CT} = (\{\mathsf{NCE.ct}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \widetilde{R})$ to $\mathcal{A}$.
5. $\mathcal{A}$ makes at most $n - Q_1$ queries of the form $(\mathsf{GID}, \mathsf{id}_q, x_{\mathsf{id}_q})$. For each query, run:
   - Sample $\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} \leftarrow \mathcal{O}(\mathsf{id}_q, \mathsf{GID}, x_{\mathsf{id}_q})$.
   - If $(\mathsf{GID}, \mathsf{id}, \cdot) \notin \mathbf{Q}$,
     - If this is the last authority query for GID, $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.Reveal}(\mathsf{NCE.msk}_{\mathsf{id}_q}, \bigoplus_{\mathsf{id}\neq\mathsf{id}_q} R_{\mathsf{id}} \oplus \widetilde{R} \oplus \mathsf{na1MAFE.ct})$ where $\mathsf{na1MAFE.ct} \leftarrow \mathcal{O}(C)$.
     - Otherwise, $\mathsf{NCE.sk}_{\mathsf{id}_q} \leftarrow \mathsf{NCE.Reveal}(\mathsf{NCE.msk}_{\mathsf{id}_q}, \mathsf{NCE.aux}_{\mathsf{id}_q}, R_{\mathsf{id}_q})$ where $R_{\mathsf{id}_q} \xleftarrow{\$} \{0,1\}^\kappa$.

   Send $\mathsf{SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}} = (\mathsf{na1MAFE.SK}_{\mathsf{GID},\mathsf{id}_q,x_{\mathsf{id}_q}}, \mathsf{NCE.sk}_{\mathsf{id}_q})$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ, n and as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is Expt_0, $\mathcal{B}$ behaves like Hyb_2 and if $\mathcal{O}$ is Expt_1, $\mathcal{B}$ behaves like Hyb_3. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between an honest challenger and simulator for $\mathsf{na1MAFE}$. This contradicts our assumption for a secure $\mathsf{na1MAFE}$ scheme. Hence, Hyb_2 and Hyb_3 are computationally indistinguishable. □

C Proofs from Section 6

In this section, we provide the missing proofs from Section 6.

C.1 Hybrid Descriptions

The description of hybrids is as follows.

$\underline{\mathsf{Hyb}_0(1^\lambda)}$: This is $\mathsf{Expt}_0^{\mathsf{dCSF},\mathcal{A},\mathcal{C}}$ in Definition 6.1.

1. $\mathcal{A}$ sends $(Q, n, s, \widetilde{\mathbf{S}})$ to $\mathcal{C}$. $\mathcal{C}$ sets $\mathbf{Q}$ to be empty initially.
2. $\mathcal{A}$ sends $q \in [Q_1]$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$. $\mathcal{C}$ does the following:
   - Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to $\mathbf{Q}$.
   - For each $u \in \mathbf{S}_{\mathsf{GID}_q}$, set $\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q} = (\mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.

   Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.

3. $\mathcal{A}$ sends a circuit C and $\{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})}$ to $\mathcal{C}$. $\mathcal{C}$ does the following:
   - Sample $\mathsf{CorrGarb.msk} \leftarrow \mathsf{CorrGarb.Setup}(1^\lambda, 1^Q, 1^s)$.
   - Sample $\{\mathsf{E}^u_0\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^n, t), \{\mathsf{E}^u_1\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 1^n, t)$.
   - Sample $\{\mathsf{E}^u_{\mathsf{CorrGarb.msk}}\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{CorrGarb.msk}, t)$.
   - Sample $\{\mathsf{E}^u_C\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, C, t)$.
   - For each $h \in [s'']$, $\left\{\mathsf{E}^u_{z,h}\right\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^T, D-1)$.
   - Set $\widehat{C}^u = \left(\mathsf{E}^u_0, \mathsf{E}^u_1, \mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \mathsf{E}^u_C, \left\{\mathsf{E}^u_{z,h}\right\}_{h\in[s'']}\right)$.

   For each $(\mathsf{GID}, \mathsf{id}, \mathbf{S}_{\mathsf{GID}}) \in \Psi_n(\mathbf{Q}), u \in \mathbf{S}_{\mathsf{GID}}$,
   - Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID},\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
   - For each $h \in [s'']$, parse $\mathsf{E}^u_{z,h}$ as $\{\zeta^u_{h,h'}\}_{h'\in[T]}$. let $Z^u_h = \sum_{j\in\Delta_{\mathsf{GID}}} \zeta^u_{h,j}$.
   - Compute for each $h \in [s'']$, $\widehat{y}^u_{\mathsf{GID},h} = \left(\mathsf{CorrGarb.Garb}\left(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta_{\mathsf{GID}}, j^*_{\mathsf{GID}}, \mathsf{E}^u_C, \mathsf{E}^u_x\right)\right)_h + Z^u_h$.
   - Set $\widehat{y}^u_{\mathsf{GID}} = \widehat{y}^u_{\mathsf{GID},1} \| \ldots \| \widehat{y}^u_{\mathsf{GID},s''}$.

   Send $\left\{\widehat{C}^u\right\}_{u\in[N]\setminus\widetilde{\mathbf{S}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q}),u\in\mathbf{S}_{\mathsf{GID}}}$ to $\mathcal{A}$.
4. $\mathcal{A}$ sends $q \in [Q_1]$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$. $\mathcal{A}$ also sends $j^*_{\mathsf{GID}_q}$ if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$. $\mathcal{C}$ does the following:
   - For each $u \in \mathbf{S}_{\mathsf{GID}_q}$, set $\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q} = (\mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.

   If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$,
   - Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID}_q,\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
   - For each $h \in [s'']$, parse $\mathsf{E}^u_{z,h}$ as $\{\zeta^u_{h,h'}\}_{h'\in[T]}$. let $Z^u_h = \sum_{j\in\Delta_q} \zeta^u_{h,j}$.
   - Compute for each $h \in [s'']$, $\widehat{y}^u_{\mathsf{GID}_q,h} = \left(\mathsf{CorrGarb.Garb}\left(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta_q, j^*_{\mathsf{GID}_q}\mathsf{E}^u_C, \mathsf{E}^u_x\right)\right)_h + Z^u_h$.
   - Set $\widehat{y}^u_{\mathsf{GID}_q} = \widehat{y}^u_{\mathsf{GID}_q,1} \| \ldots \| \widehat{y}^u_{\mathsf{GID}_q,s''}$.

   Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to $\mathbf{Q}$. Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}, \left\{\widehat{y}^u_{\mathsf{GID}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.
5. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q^*}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$.
6. $\mathcal{A}$ outputs b'. Output b'.

$\underline{\mathsf{Hyb}_1(1^\lambda)}$: In this hybrid, we change the way $\widehat{C}^u$ is sampled. The changes are highlighted in red.

1. $\mathcal{A}$ sends $(Q, n, s, \widetilde{\mathbf{S}})$ to $\mathcal{C}$. Initial $\mathbf{Q}$ to be empty initially.
2. $\mathcal{A}$ sends $q \in [Q_1]$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$. Do the following:
   - Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to $\mathbf{Q}$.
   - For each $u \in \mathbf{S}_{\mathsf{GID}_q}$, set $\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q} = (\mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.

   Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.

3. Create constraint dictionaries $\mathbf{K}_1, \ldots, \mathbf{K}_{s''}$ which are initially empty. Use the table $\mathbf{K}_h[h', u]$ to store the randomly sampled $\zeta_{h,h'}(u)$, where $h \in [s''], h' \in [T], u \in [N]$ for the $D-1$ degree polynomial $\zeta_{h,h'}$.

4. $\mathcal{A}$ sends circuit C and $\{j^*_{\mathsf{GID}}\}_{\mathsf{GID} \in \Psi_n(\mathbf{Q})}$. Do the following: Using C:

   - Sample $\mathsf{CorrGarb.msk} \leftarrow \mathsf{CorrGarb.Setup}(1^\lambda, 1^Q, 1^s)$.
   - Sample $\{\mathsf{E}^u_0\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^n, t), \{\mathsf{E}^u_1\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 1^n, t)$.
   - Sample $\{\mathsf{E}^u_{\mathsf{CorrGarb.msk}}\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{CorrGarb.msk}, t)$.
   - Sample $\{\mathsf{E}^u_C\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, C, t)$.

   For each $(\mathsf{GID}, \mathsf{id}, \mathbf{S}_{\mathsf{GID}}) \in \Psi_n(\mathbf{Q}), u \in \mathbf{S}_{\mathsf{GID}}$,

   - Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID},\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
   - Let $y_{\mathsf{GID}} = \mathsf{CorrGarb.Garb}(\mathsf{CorrGarb.msk}, \Delta_{\mathsf{GID}}, j^*_{\mathsf{GID}} C, x = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n}))$.
   - For each $h \in [s'']$, sample a random degree $D-1$ polynomial η_h such that $\eta_h(0) = (y_{\mathsf{GID}})_h$.
   - Set $\widehat{y}^u_{\mathsf{GID}} = \eta_1(u) \,\|\, \ldots \,\|\, \eta_{s''}(u)$.
   - For each $h \in [s'']$, for the unique index $j^*_{\mathsf{GID}} \in \Delta_{\mathsf{GID}}$ (provided by the adversary), set the values of $\zeta_{h,j^*_{\mathsf{GID}}}(u)$ to follow the relation:

   $$\eta_h(u) = (\mathsf{CorrGarb.Garb}\,(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta_{\mathsf{GID}}, j^*_{\mathsf{GID}} \mathsf{E}^u_C, \mathsf{E}^u_x))_h + \zeta_{h,j^*_{\mathsf{GID}}}(u) + \sum_{j \in \Delta_{\mathsf{GID}} \setminus \{j^*_{\mathsf{GID}}\}} \zeta_{h,j}(u) \quad (1)$$

   in accordance with the constraint values in $\mathbf{K}_h$. Update $\mathbf{K}_h[j^*_{\mathsf{GID}}, u]$ with the new values.

   For the remaining required $\zeta_{h,h'}(u)$, select random values and update $\mathbf{K}_h$ accordingly. For each $h \in [s'']$ set $\mathsf{E}^u_{z,h} = \{\zeta_{h,h'}(u)\}_{h'\in[T]}$. Set $\widehat{C}^u = \left(\mathsf{E}^u_0, \mathsf{E}^u_1, \mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \mathsf{E}^u_C, \left\{\mathsf{E}^u_{z,h}\right\}_{h\in[s'']}\right)$. Send $\left\{\widehat{C}^u\right\}_{u\in[N]\setminus\widetilde{\mathbf{S}}}$, $\{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q}), u\in\mathbf{S}_{\mathsf{GID}}}$ to $\mathcal{A}$.

5. $\mathcal{A}$ sends $q \in [Q_1]$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$. $\mathcal{A}$ also sends $j^*_{\mathsf{GID}_q}$ if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$. Do the following:

   - For each $u \in \mathbf{S}_{\mathsf{GID}_q}$, set $\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q} = (\mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.

   If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$,

   - Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID}_q,\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
   - Let $y_{\mathsf{GID}_q} = \mathsf{CorrGarb.Garb}(\mathsf{CorrGarb.msk}, \Delta_q, j^*_{\mathsf{GID}_q}, C, x = (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
   - For each $h \in [s'']$, sample a random degree $D-1$ polynomial η_h such that $\eta_h(0) = (y_{\mathsf{GID}_q})_h$.
   - Set $\widehat{y}^u_{\mathsf{GID}_q} = \eta_1(u) \,\|\, \ldots \,\|\, \eta_{s''}(u)$.
   - For each $h \in [s'']$, for the unique index $j^*_{\mathsf{GID}_q} \in \Delta_q$ (provided by the adversary), set the values of $\zeta_{h,j^*_{\mathsf{GID}_q}}(u)$ to follow the relation:

   $$\eta_h(u) = \left(\mathsf{CorrGarb.Garb}\left(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta_q, j^*_{\mathsf{GID}_q}, \mathsf{E}^u_C, \mathsf{E}^u_x\right)\right)_h + \zeta_{h,j^*_{\mathsf{GID}_q}}(u) + \sum_{j\in\Delta_q\setminus\{j^*_{\mathsf{GID}_q}\}} \zeta_{h,j}(u)$$

   in accordance with the constraint values in $\mathbf{K}_h$. Update $\mathbf{K}_j[j^*_{\mathsf{GID}_q}, u]$ with new values.

   Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to $\mathbf{Q}$. Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}, \left\{\widehat{y}^u_{\mathsf{GID}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.

6. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q^*}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$.

7. $\mathcal{A}$ outputs b'. Output b'.

$\underline{\mathsf{Hyb}_2(1^\lambda)}$: In this hybrid, we simulate $\mathsf{CorrGarb}$. The changes are highlighted in red.

1. $\mathcal{A}$ sends $(Q, n, s, \widetilde{\mathbf{S}})$ to $\mathcal{C}$. Initial $\mathbf{Q}$ to be empty initially.
2. $\mathcal{A}$ sends $q \in [Q_1]$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$. Do the following:
   - Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to $\mathbf{Q}$.
   - For each $u \in \mathbf{S}_{\mathsf{GID}_q}$, set $\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q} = (\mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.

   Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.
3. Create constraint dictionaries $\mathbf{K}_1, \ldots, \mathbf{K}_{s''}$ which are initially empty. Use the table $\mathbf{K}_h[h', u]$ to store the randomly sampled $\zeta_{h,h'}(u)$, where $h \in [s''], h' \in [T], u \in [N]$ for the $D-1$ degree polynomial $\zeta_{h,h'}$.
4. $\mathcal{A}$ sends circuit C and $\{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})}$. Do the following: Using C:
   - Sample $\mathsf{MSK} \leftarrow \mathsf{sim}_{\mathsf{CorrGarb}}(1^\lambda, 1^Q, 1^s)$.
   - Sample $\{\mathsf{E}^u_0\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^n, t), \{\mathsf{E}^u_1\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 1^n, t)$.
   - Sample $\{\mathsf{E}^u_{\mathsf{CorrGarb.msk}}\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^{|\mathsf{CorrGarb.msk}|}, t)$.
   - Sample $\{\mathsf{E}^u_C\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, C, t)$.

   For each $(\mathsf{GID}, \mathsf{id}, \mathbf{S}_{\mathsf{GID}}) \in \Psi_n(\mathbf{Q}), u \in \mathbf{S}_{\mathsf{GID}}$,
   - Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID},\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
   - Let $y_{\mathsf{GID}} = \mathsf{sim}_{\mathsf{CorrGarb}}(1^{|C|}, \mathsf{MSK}, \Delta_{\mathsf{GID}}, j^*_{\mathsf{GID}}, x = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n}), C(x))$.
   - For each $h \in [s'']$, sample a random degree $D-1$ polynomial η_h such that $\eta_h(0) = (y_{\mathsf{GID}})_h$.
   - Set $\widehat{y}^u_{\mathsf{GID}} = \eta_1(u) \,\|\, \ldots \,\|\, \eta_{s''}(u)$.
   - For each $h \in [s'']$, for the unique index $j^*_{\mathsf{GID}} \in \Delta_{\mathsf{GID}}$ (provided by the adversary), set the values of $\zeta_{h,j^*_{\mathsf{GID}}}(u)$ to follow the relation:
   $$\eta_h(u) = \left(\mathsf{CorrGarb.Garb}\left(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta_{\mathsf{GID}}, j^*_{\mathsf{GID}}\mathsf{E}^u_C, \mathsf{E}^u_x\right)\right)_h + \zeta_{h,j^*_{\mathsf{GID}}}(u) + \sum_{j\in\Delta_{\mathsf{GID}}\setminus\{j^*_{\mathsf{GID}}\}} \zeta_{h,j}(u)$$
   in accordance with the constraint values in $\mathbf{K}_h$. Update $\mathbf{K}_h[j^*_{\mathsf{GID}}, u]$ with the new values.

   For the remaining required $\zeta_{h,h'}(u)$, select random values and update $\mathbf{K}_h$ accordingly. For each $h \in [s'']$ set $\mathsf{E}^u_{z,h} = \{\zeta_{h,h'}(u)\}_{h'\in[T]}$. Set $\widehat{C}^u = \left(\mathsf{E}^u_0, \mathsf{E}^u_1, \mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \mathsf{E}^u_C, \left\{\mathsf{E}^u_{z,h}\right\}_{h\in[s'']}\right)$. Send $\left\{\widehat{C}^u\right\}_{u\in[N]\setminus\widetilde{\mathbf{S}}}$, $\{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q}),u\in\mathbf{S}_{\mathsf{GID}}}$ to $\mathcal{A}$.
5. $\mathcal{A}$ sends $q \in [Q_1]$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$. $\mathcal{A}$ also sends $j^*_{\mathsf{GID}_q}$ if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$. Do the following:
   - For each $u \in \mathbf{S}_{\mathsf{GID}_q}$, set $\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q} = (\mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.

   If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$,
   - Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID}_q,\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
   - Let $y_{\mathsf{GID}_q} = \mathsf{sim}_{\mathsf{CorrGarb}}(1^{|C|}, \mathsf{MSK}, \Delta_q, j^*_{\mathsf{GID}_q}, x = (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}), C(x))$.
   - For each $h \in [s'']$, sample a random degree $D-1$ polynomial η_h such that $\eta_h(0) = (y_{\mathsf{GID}_q})_h$.
   - Set $\widehat{y}^u_{\mathsf{GID}_q} = \eta_1(u) \,\|\, \ldots \,\|\, \eta_{s''}(u)$.

- For each $h \in [s'']$, for the unique index $j^*_{\mathsf{GID}_q} \in \Delta_q$ (provided by the adversary), set the values of $\zeta_{h,j^*_{\mathsf{GID}_q}}(u)$ to follow the relation:

$$\eta_h(u) = \left(\mathsf{CorrGarb.Garb}\left(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta_q, j^*_{\mathsf{GID}_q}, \mathsf{E}^u_C, \mathsf{E}^u_x\right)\right)_h + \zeta_{h,j^*_q}(u) + \sum_{j \in \Delta_q \setminus \{j^*_{\mathsf{GID}_q}\}} \zeta_{h,j}(u)$$

in accordance with the constraint values in $\mathbf{K}_h$. Update $\mathbf{K}_j[j^*_{\mathsf{GID}_q}, u]$ with new values.

Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to $\mathbf{Q}$. Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u \in \mathbf{S}_{\mathsf{GID}_q}}, \left\{\widehat{y}^u_{\mathsf{GID}_q}\right\}_{u \in \mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.

6. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q \neq q'} \mathbf{S}_q \cap \mathbf{S}_{q^*}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'}\right)$.

7. $\mathcal{A}$ outputs b'. Output b'.

$\underline{\mathsf{Hyb}_3(1^\lambda)}$: This is the $\mathsf{Expt}_1^{\mathsf{dCSF},\mathcal{A},\mathsf{sim}_{\mathsf{dCSF}}}$ from Definition 6.1. The changes are highlighted in red.

1-3. Perform these steps same as Hyb_2.

4. $\mathcal{A}$ sends circuit C. Using C:
   - Sample $\mathsf{MSK} \leftarrow \mathsf{sim}_{\mathsf{CorrGarb}}(1^\lambda, 1^Q, 1^s)$.
   - Sample $\{\mathsf{E}^u_0\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^n, t), \{\mathsf{E}^u_1\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 1^n, t)$.
   - Sample $\{\mathsf{E}^u_{\mathsf{CorrGarb.msk}}\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^{|\mathsf{CorrGarb.msk}|}, t)$.
   - Sample $\{\mathsf{E}^u_C\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^{|C|}, t)$.
   - Perform the rest of the steps same as Hyb_2.

5-7. Perform these steps same as Hyb_2.

C.2 Proofs of Claims

The proof of claim from Section 6 is as follows.

Proof of Claim 6.3. In both the hybrids, the values of the polynomial $\eta_h(\cdot)$ for any $h \in [s'']$ are random elements of a field. In both the hybrids, the relationship between $\eta_h(\cdot)$ and $\zeta_{h,j}(\cdot)$ is maintained. The only difference is that in $\mathsf{Hyb}_0(1^\lambda)$, we sample $\zeta_{h,j}(\cdot)$ and then $\eta_h(\cdot)$ is defined implicitly by the relationship. On the other hand, in $\mathsf{Hyb}_1(1^\lambda)$, we sample $\eta_h(\cdot)$ and $\zeta_{h,j}(\cdot)$ is explicitly calculated by the relationship in Equation 1 and the constraints in $\mathbf{K}_h$. As both of these are still random elements and because of Lemma 4.3, we have that Hyb_0 and Hyb_1 are identical. □

Proof of 6.4. Note that the only difference between Hyb_1 and Hyb_2 is that we are simulating the $\mathsf{CorrGarb}$ instantiation. We are also substituting $\mathsf{CorrGarb.msk}$ with an all 0's string. But this change remains statistically indistinguishable as the size of the set $[N] \setminus \widetilde{\mathbf{S}}$ is at most t from Lemma 4.2. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids Hyb_1 and Hyb_2, we will construct an adversary $\mathcal{B}$ that can break the security of $\mathsf{CorrGarb}$. More formally, if $\mathcal{A}$'s advantage is such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_1}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_2}\right]\right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction $\mathcal{B}$ which distinguishes between oracle access between Expt_0 and Expt_1 for the $\mathsf{CorrGarb}$ scheme. More formally,

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathsf{CorrGarb},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathsf{CorrGarb},\mathcal{B},\mathsf{sim}_{\mathsf{CorrGarb}}}\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda)}$:

1. $\mathcal{A}$ sends $(Q, n, s, \widetilde{\mathbf{S}})$ to $\mathcal{C}$. Initial $\mathbf{Q}$ to be empty initially.
2. $\mathcal{A}$ sends $q \in [Q_1]$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$. Do the following:
   - Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to $\mathbf{Q}$.
   - For each $u \in \mathbf{S}_{\mathsf{GID}_q}$, set $\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q} = (\mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.

   Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.
3. Create constraint dictionaries $\mathbf{K}_1, \ldots, \mathbf{K}_{s''}$ which are initially empty. Use the table $\mathbf{K}_h[h', u]$ to store the randomly sampled $\zeta_{h,h'}(u)$, where $h \in [s''], h' \in [T], u \in [N]$ for the $D-1$ degree polynomial $\zeta_{h,h'}$.
4. $\mathcal{A}$ sends circuit C and $\{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})}$. Do the following: Using C:
   - Send (Q, s, C) to $\mathcal{O}$.
   - Sample $\{\mathsf{E}^u_0\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^n, t), \{\mathsf{E}^u_1\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 1^n, t)$.
   - Sample $\{\mathsf{E}^u_{\mathsf{CorrGarb.msk}}\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, 0^{|\mathsf{CorrGarb.msk}|}, t)$.
   - Sample $\{\mathsf{E}^u_C\}_{u\in[N]} \leftarrow \mathsf{MsgEnc}(1^\lambda, 1^Q, 1^n, 1^s, C, t)$.

   For each $(\mathsf{GID}, \mathsf{id}, \mathbf{S}_{\mathsf{GID}}) \in \Psi_n(\mathbf{Q}), u \in \mathbf{S}_{\mathsf{GID}}$,
   - Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID},\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
   - Let $y_{\mathsf{GID}} = \mathcal{O}(\Delta_{\mathsf{GID}}, j^*_{\mathsf{GID}}, x = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n}))$.
   - For each $h \in [s'']$, sample a random degree $D-1$ polynomial η_h such that $\eta_h(0) = (y_{\mathsf{GID}})_h$.
   - Set $\widehat{y}^u_{\mathsf{GID}} = \eta_1(u) \,\|\, \ldots \,\|\, \eta_{s''}(u)$.
   - For each $h \in [s'']$, for the unique index $j^*_{\mathsf{GID}} \in \Delta_{\mathsf{GID}}$ (provided by the adversary), set the values of $\zeta_{h,j^*_{\mathsf{GID}}}(u)$ to follow the relation:
     $$\eta_h(u) = \left(\mathsf{CorrGarb.Garb}\left(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta_{\mathsf{GID}}, j^*_{\mathsf{GID}}\mathsf{E}^u_C, \mathsf{E}^u_x\right)\right)_h + \zeta_{h,j^*_{\mathsf{GID}}}(u) + \sum_{j\in\Delta_{\mathsf{GID}}\setminus\{j^*_{\mathsf{GID}}\}} \zeta_{h,j}(u)$$
     in accordance with the constraint values in $\mathbf{K}_h$. Update $\mathbf{K}_h[j^*_{\mathsf{GID}}, u]$ with the new values.

   For the remaining required $\zeta_{h,h'}(u)$, select random values and update $\mathbf{K}_h$ accordingly. For each $h \in [s'']$ set $\mathsf{E}^u_{z,h} = \{\zeta_{h,h'}(u)\}_{h'\in[T]}$. Set $\widehat{C}^u = \left(\mathsf{E}^u_0, \mathsf{E}^u_1, \mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \mathsf{E}^u_C, \left\{\mathsf{E}^u_{z,h}\right\}_{h\in[s'']}\right)$. Send $\left\{\widehat{C}^u\right\}_{u\in[N]\setminus\widetilde{\mathbf{S}}}$, $\{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q}),u\in\mathbf{S}_{\mathsf{GID}}}$ to $\mathcal{A}$.
5. $\mathcal{A}$ sends $q \in [Q_1]$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q), \mathbf{S}_{\mathsf{GID}_q}$. $\mathcal{A}$ also sends $j^*_{\mathsf{GID}_q}$ if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$. Do the following:
   - For each $u \in \mathbf{S}_{\mathsf{GID}_q}$, set $\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q} = (\mathsf{GID}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_q)$.

   If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$,
   - Parse E^u_0 as $\{\mu^u_{0,h'}\}_{h'\in[n]}$ and E^u_1 as $\{\mu^u_{1,h'}\}_{h'\in[n]}$. Set $\mathsf{E}^u_x = \{\mu^u_{x_{\mathsf{GID}_q,\mathsf{id}},\mathsf{id}}\}_{\mathsf{id}\in[n]}$.
   - Let $y_{\mathsf{GID}_q} = \mathcal{O}(\Delta_q, j^*_{\mathsf{GID}_q}, x = (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
   - For each $h \in [s'']$, sample a random degree $D-1$ polynomial η_h such that $\eta_h(0) = (y_{\mathsf{GID}_q})_h$.
   - Set $\widehat{y}^u_{\mathsf{GID}_q} = \eta_1(u) \,\|\, \ldots \,\|\, \eta_{s''}(u)$.
   - For each $h \in [s'']$, for the unique index $j^*_{\mathsf{GID}_q} \in \Delta_q$ (provided by the adversary), set the values of $\zeta_{h,j^*_{\mathsf{GID}_q}}(u)$ to follow the relation:
     $$\eta_h(u) = \left(\mathsf{CorrGarb.Garb}\left(\mathsf{E}^u_{\mathsf{CorrGarb.msk}}, \Delta_q, j^*_{\mathsf{GID}_q}, \mathsf{E}^u_C, \mathsf{E}^u_x\right)\right)_h + \zeta_{h,j^*_q}(u) + \sum_{j\in\Delta_q\setminus\{j^*_{\mathsf{GID}_q}\}} \zeta_{h,j}(u)$$

in accordance with the constraint values in $\mathbf{K}_h$. Update $\mathbf{K}_j[j^*_{\mathsf{GID}_q}, u]$ with new values.

Add $(\mathsf{GID}_q, \mathsf{id}_q, \mathbf{S}_{\mathsf{GID}_q})$ to $\mathbf{Q}$. Send $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}, \left\{\widehat{y}^u_{\mathsf{GID}_q}\right\}_{u\in\mathbf{S}_{\mathsf{GID}_q}}$ to $\mathcal{A}$.

6. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q^*}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$.

7. $\mathcal{A}$ outputs b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ, Q and as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is Expt_0, $\mathcal{B}$ behaves like Hyb_1 and if $\mathcal{O}$ is Expt_1, $\mathcal{B}$ behaves like Hyb_2. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between an honest challenger and simulator for $\mathsf{CorrGarb}$. This contradicts our assumption for a secure $\mathsf{CorrGarb}$ scheme. Hence, Hyb_1 and Hyb_2 are computationally indistinguishable. □

Proof of Claim 6.5. As the number of users whose client encodings are revealed to the adversary, that is, size of the set $[N] \setminus \widetilde{\mathbf{S}} = \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ is at most t (from Lemma 4.2), for any $h \in [s'']$, at most $D-1$ values of the polynomial $\eta_h(\cdot)$ are revealed. Hence, $\eta_h(0)$ remains perfectly hidden. This can also be seen from the security of Shamir secret sharing scheme. □

D Proofs from Section 7

In this section, we provide the missing proofs from Section 7.

D.1 Hybrid Descriptions

The description of hybrids is as follows.

$\underline{\mathsf{Hyb}_0(1^\lambda)}$: This is $\mathsf{Expt}_0^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathcal{C}}$ from Definition 7.1.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum challenge circuit size s, and for each $q \in [Q]$, secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following:

   - Compute for each $u \in [N], \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - Sample $K \xleftarrow{\$} \{0,1\}^\lambda$, and set $\mathsf{crs} = (Q, n, s, (\mathsf{1MAFE.crs}_u)_{u\in[N]}, K)$.
   - For each $\mathsf{id} \in [n], u \in [N], (\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{id})$. Set $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{1MAFE.mpk}_{\mathsf{id},u})_{u\in[N]}$ and $\mathsf{MSK}_{\mathsf{id}} = (\mathsf{1MAFE.msk}_{\mathsf{id},u})_{u\in[N]}$.
   - For each $q \in [Q]$,
     - $(\mathbf{S}_q, \Delta_q) = \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$.
     - Compute $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta)$.
     - For each $u \in \mathbf{S}_q$, compute $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
     - Set $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_q, \{\mathsf{1MAFE.sk}_u\}_{u\in\mathbf{S}_q})$.

   Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \{\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}\}_{q\in[Q]}$ to $\mathcal{A}$.

2. $\mathcal{A}$ sends C to $\mathcal{C}$. $\mathcal{C}$ does the following.

   - Compute $\{\widehat{C}^u\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{dCSF.UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.

Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.

3. $\mathcal{A}$ output b'. Output b'.

$\underline{\mathsf{Hyb}_1(1^\lambda)}$: In this hybrid, we will check if Lemma 4.2 holds for pseudorandom sets. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum challenge circuit size s, and for each $q \in [Q]$, secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following:

   - Sample $K \xleftarrow{\$} \{0,1\}^\lambda$, and deterministically compute $(\mathbf{S}_q, \Delta_q) \leftarrow \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$. For every $q, q' \in [Q^*]$, calculate $\mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| \geq t$, abort and output $\perp$.
   - Compute for each $u \in [N], \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - Set $\mathsf{crs} = (Q, n, s, (\mathsf{1MAFE.crs}_u)_{u\in[N]}, K)$.
   - Perform the rest of the steps same as Hyb_0.

2-3. Perform these steps same as Hyb_0.

$\underline{\mathsf{Hyb}_2(1^\lambda)}$: In this hybrid, we will check if Lemma 4.3 holds for pseudorandom sets. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum challenge circuit size s, and for each $q \in [Q]$, secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following:

   - Sample $K \xleftarrow{\$} \{0,1\}^\lambda$, and deterministically compute $(\mathbf{S}_q, \Delta_q) \leftarrow \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$. For every $q, q' \in [Q^*]$, calculate $\mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| \geq t$, abort and output $\perp$.
   - If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$.
   - Compute for each $u \in [N], \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - Set $\mathsf{crs} = (Q, n, s, (\mathsf{1MAFE.crs}_u)_{u\in[N]}, K)$.
   - Perform the rest of the steps same as Hyb_1.

2-3. Perform these steps same as Hyb_1.

$\underline{\mathsf{Hyb}_{3,j}(1^\lambda) \text{ for } j \in [N+1]}$: In this hybrid, we will simulate the first $j-1$ instantiation of 1MAFE using $\underline{\mathsf{sim}^u_{\mathsf{1MAFE}}}$. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum challenge circuit size s, and for each $q \in [Q]$, secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following:

   - Sample $K \xleftarrow{\$} \{0,1\}^\lambda$, and deterministically compute $(\mathbf{S}_q, \Delta_q) \leftarrow \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$. For every $q, q' \in [Q^*]$, calculate $\mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| \geq t$, abort and output $\perp$.
   - If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$.
   - Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$. For each $q \in [Q^*]$, set a $j^*_q \in [T]$ such that $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Correspondingly create the string $(j^*_1, \ldots, j^*_Q)$.
   - Compute for each $u \in [N]$,
     - If $u \geq j$ or $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.

- Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.

- Set $\mathsf{crs} = (Q, n, s, (\mathsf{1MAFE.crs}_u)_{u\in[N]}, K)$.
- For each $\mathsf{id} \in [n], u \in [N]$,
  - If $u \geq j$ or $u \in \mathbf{S}_{\mathsf{corr}}$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
  - Otherwise, $\mathsf{1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.

  Set $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{1MAFE.mpk}_{\mathsf{id},u})_{u\in[N]}$ and $\mathsf{MSK}_{\mathsf{id}} = (\mathsf{1MAFE.msk}_{\mathsf{id},u})_{u\in\mathbf{S}_{\mathsf{corr}}\cup[j,N]}$.
- For each $q \in [Q]$,
  - Compute $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta)$.
  - For each $u \in \mathbf{S}_q$,
    - If $u \in \mathbf{S}_q \cap \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
    - Otherwise, $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Set $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_q, \{\mathsf{1MAFE.sk}_u\}_{u\in\mathbf{S}_q})$.

Create $\mathbf{Q} = \{(\mathsf{GID}_q, \mathsf{id}_q) : q \in [Q]\}$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \{\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}\}_{q\in[Q]}$ to $\mathcal{A}$.

2. $\mathcal{A}$ sends C to $\mathcal{C}$. $\mathcal{C}$ does the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\{\widehat{C}^u\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot, \ldots, \cdot) = \mathsf{dCSF.UserComp}(\cdot, \ldots, \cdot, \widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
     - Otherwise, if $\exists\ q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q}\setminus\{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, \mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, F^u(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.
3. $\mathcal{A}$ output b'. Output b'.

$\underline{\mathsf{Hyb}_4(1^\lambda)}$: This is the $\mathsf{Expt}_1^{\mathsf{stMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{stMA\text{-}FE}}}$ from Definition 7.1. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum challenge circuit size s, and for each $q \in [Q]$, secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. Do the following:
   - Sample $K \xleftarrow{\$} \{0,1\}^\lambda$, and deterministically compute $(\mathbf{S}_q, \Delta_q) \leftarrow \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$. For every $q, q' \in [Q^*]$, calculate $\mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| \geq t$, abort and output $\perp$.
   - If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$.
   - Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$. For each $q \in [Q^*]$, set a $j^*_q \in [T]$ such that $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Correspondingly create the string $(j^*_1, \ldots, j^*_Q)$.
   - Initiate $\mathsf{sim}_{\mathsf{dCSF}}(1^\lambda, 1^Q, 1^n, 1^s, \widetilde{\mathbf{S}})$.
   - Compute for each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
     - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.

- Set $\mathsf{crs} = (Q, n, s, (\mathsf{1MAFE.crs}_u)_{u\in[N]}K)$.
- For each $\mathsf{id} \in [n], u \in [N]$,
  - If $u \in \mathbf{S}_{\mathsf{corr}}$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
  - Otherwise, $\mathsf{1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.

  Set $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{1MAFE.mpk}_{\mathsf{id},u})_{u\in[N]}$ and $\mathsf{MSK}_{\mathsf{id}} = (\mathsf{1MAFE.msk}_{\mathsf{id},u})_{u\in\mathbf{S}_{\mathsf{corr}}\cup[j,N]}$.
- For each $q \in [Q]$,
  - Compute $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in\mathbf{S}_q} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}((\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta), \mathbf{S}_q)$.
  - For each $u \in \mathbf{S}_q$,
    - If $u \in \mathbf{S}_q \cap \mathbf{S}_{\mathsf{corr}}$, $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
    - Otherwise, $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Set $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_q, \{\mathsf{1MAFE.sk}_u\}_{u\in\mathbf{S}_q})$.

Create $\mathbf{Q} = \{(\mathsf{GID}_q, \mathsf{id}_q) : q \in [Q]\}$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \{\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}\}_{q\in[Q]}$ to $\mathcal{A}$.

2. $\mathcal{A}$ sends C. Do the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\{\widehat{C}^u\}_{u\in\mathbf{S}_{\mathsf{corr}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q}),u\in\mathbf{S}_{\mathsf{GID}}} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}(1^{|C|}, \mathbf{V}, \{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})})$.
   - For each $u \in \mathbf{S}_{\mathsf{corr}}$, let $F^u(\cdot, \ldots, \cdot) = \mathsf{dCSF.UserComp}(\cdot, \ldots, \cdot, \widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$, $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
     - Otherwise, if $\exists\, q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q}\setminus\{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \widehat{y}^u_{\mathsf{GID}_q})$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.

3. $\mathcal{A}$ output b'. Output b'.

D.2 Proofs of Claims

The proofs of claims are as follows.

Proof of Claim 7.3. Note that Lemma 4.2 talks about random strings. However, we want to argue the same about pseudorandom strings. Assume that the abort probability of Hyb_1 is ϵ and consider the following adversary against the PRF scheme.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda)}$:

1. Query $\mathcal{O}$ with randomly generated $\mathsf{GID}_1, \ldots, \mathsf{GID}_Q$ to receive $(\mathbf{S}_1, \Delta_1), \ldots, (\mathbf{S}_Q, \Delta_Q)$.
2. Calculate $\mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| \geq t$, output 0. Otherwise output 1.

Now, we calculate the advantage of $\mathcal{B}$ in the PRF security game defined as follows:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B}} &= \left|\Pr[0 \leftarrow \mathcal{B}^{\mathcal{O}_0(\cdot)}(1^\lambda)] - \Pr[0 \leftarrow \mathcal{B}^{\mathcal{O}_1(\cdot)}(1^\lambda)]\right| \\ &= \epsilon - \mathsf{negl}(\lambda)\end{aligned}$$

Hence, ϵ has to be a negligible function in λ. □

Proof of Claim 7.4. Note that Lemma 4.3 talks about random strings. However, we want to argue the same about pseudorandom strings. Assume that the abort probability of Hyb_2 is ϵ and consider the following adversary against the PRF scheme.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda)}$:

1. Query $\mathcal{O}$ with randomly generated $\mathsf{GID}_1, \ldots, \mathsf{GID}_Q$ to receive $(\mathbf{S}_1, \Delta_1), \ldots, (\mathbf{S}_Q, \Delta_Q)$.
2. If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q' \neq q} \Delta_{q'} \right) = \emptyset$, output 0. Otherwise output 1.

Now, we calculate the advantage of $\mathcal{B}$ in the PRF security game defined as follows:

$$\begin{aligned} \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B}} &= \left| \Pr[0 \leftarrow \mathcal{B}^{\mathcal{O}_0(\cdot)}(1^\lambda)] - \Pr[0 \leftarrow \mathcal{B}^{\mathcal{O}_1(\cdot)}(1^\lambda)] \right| \\ &= \epsilon - \mathsf{negl}(\lambda) \end{aligned}$$

Hence, ϵ has to be a negligible function in λ. □

Proof of Claim 7.6. Note that in $\mathsf{Hyb}_{3,j}$ and $\mathsf{Hyb}_{3,j+1}$, the only difference is in the j-th instantiation of 1MAFE. Moreover, if $j \in \mathbf{S}_{\mathsf{corr}}$, we do not simulate this instantiation. WLOG, assume that $j \notin \mathbf{S}_{\mathsf{corr}}$. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{3,j}$ and $\mathsf{Hyb}_{3,j+1}$, we will construct an adversary $\mathcal{B}$ that can break the security of 1MAFE. More formally, if $\mathcal{A}$'s advantage is such that

$$\left| \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_{3,j}}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_{3,j+1}}\right] \right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction which distinguishes between oracle access between Expt_0 and Expt_1 for the 1MAFE scheme. More formally,

$$\left| \Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathsf{1MAFE},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathsf{1MAFE},\mathcal{B},\mathsf{sim}_{\mathsf{1MAFE}}}\right] \right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda)}$:

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum challenge circuit size s, and for each $q \in [Q]$, secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q, \mathsf{id}_q})$. $\mathcal{C}$ does the following:
   - Sample $K \xleftarrow{\$} \{0,1\}^\lambda$, and deterministically compute $(\mathbf{S}_q, \Delta_q) \leftarrow \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$. For every $q, q' \in [Q^*]$, calculate $\mathbf{S}_{\mathsf{corr}} = \bigcup_{q \neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| \geq t$, abort and output $\perp$.
   - If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'} \right) = \emptyset$, abort and output $\perp$.
   - Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$. For each $q \in [Q^*]$, set a $j^*_q \in [T]$ such that $j^*_q \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'} \right)$. Correspondingly create the string $(j^*_1, \ldots, j^*_Q)$.
   - Compute for each $u \in [N]$,
     - If $u > j$ or $u \in \mathbf{S}_{\mathsf{corr}}$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
     - Otherwise, if $u = j$, $\mathsf{1MAFE.crs}_u \leftarrow \mathcal{O}(1^\lambda, 1^n)$.
     - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.
   - Set $\mathsf{crs} = (Q, n, s, (\mathsf{1MAFE.crs}_u)_{u \in [N]}, K)$.
   - For each $\mathsf{id} \in [n], u \in [N]$,

- If $u > j$ or $u \in \mathbf{S}_{\mathsf{corr}}$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
- Otherwise, if $u = j$, $\mathsf{1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathcal{O}(\mathsf{id})$.
- Otherwise, $\mathsf{1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.

Set $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{1MAFE.mpk}_{\mathsf{id},u})_{u\in[N]}$ and $\mathsf{MSK}_{\mathsf{id}} = (\mathsf{1MAFE.msk}_{\mathsf{id},u})_{u\in\mathbf{S}_{\mathsf{corr}}\cup[j,N]}$.

- For each $q \in [Q]$,
  - Compute $\left\{\widehat{x}^u_{\mathsf{GID},\mathsf{id}}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta)$.
  - For each $u \in \mathbf{S}_q$,
    - If $u \in \mathbf{S}_q \cap \mathbf{S}_{\mathsf{corr}}$ or $u > j$, $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
    - Otherwise, if $u = j$, $\mathsf{1MAFE.sk}_u \leftarrow \mathcal{O}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
    - Otherwise, $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Set $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_q, \{\mathsf{1MAFE.sk}_u\}_{u\in\mathbf{S}_q})$.

Create $\mathbf{Q} = \{(\mathsf{GID}_q, \mathsf{id}_q) : q \in [Q]\}$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}, \{\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}\}_{q\in[Q]}$ to $\mathcal{A}$.

2. $\mathcal{A}$ sends C to $\mathcal{C}$. $\mathcal{C}$ does the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\{\widehat{C}^u\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{dCSF.UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u > j$, $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
     - Otherwise, if $u = j$ and $\mathsf{ct}_u \leftarrow \mathcal{O}(F^u)$.
     - Otherwise, if $\exists\ q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q}\setminus\{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, \mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, F^u(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.
3. $\mathcal{A}$ output b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ, Q, n, s, and as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is Expt_0, $\mathcal{B}$ behaves like $\mathsf{Hyb}_{3,j}$ and if $\mathcal{O}$ is Expt_1, $\mathcal{B}$ behaves like $\mathsf{Hyb}_{3,j+1}$. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between an honest challenger and simulator for 1MAFE. This contradicts our assumption for a secure 1MAFE scheme. Hence, $\mathsf{Hyb}_{3,j}$ and $\mathsf{Hyb}_{3,j+1}$ are computationally indistinguishable. □

Proof of Claim 7.8. Note that the only difference between $\mathsf{Hyb}_{3,N+1}$ and Hyb_4 is that we simulate the dCSF instantiation. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{3,N+1}$ and Hyb_4, we will construct an adversary $\mathcal{B}$ that can break the security of dCSF. More formally, if $\mathcal{A}$'s advantage is such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_{3,N+1}}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_4}\right]\right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction which distinguishes between oracle access between Expt_0 and Expt_1 for the dCSF scheme. More formally,

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathsf{dCSF},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathsf{dCSF},\mathcal{B},\mathsf{sim}_{\mathsf{dCSF}}}\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda):}$

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum challenge circuit size s, and for each $q \in [Q]$, secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. Do the following:

   - Sample $K \xleftarrow{\$} \{0,1\}^\lambda$, and deterministically compute $(\mathbf{S}_q, \Delta_q) \leftarrow \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$. For every $q, q' \in [Q^*]$, calculate $\mathbf{S}_{\mathsf{corr}} = \bigcup_{q \neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| \geq t$, abort and output $\perp$.
   - If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'} \right) = \emptyset$, abort and output $\perp$.
   - Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$. For each $q \in [Q^*]$, set a $j_q^* \in [T]$ such that $j_q^* \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'} \right)$. Correspondingly create the string $(j_1^*, \ldots, j_Q^*)$.
   - Initiate $\mathcal{O}(1^\lambda, 1^Q, 1^n, 1^s, \widetilde{\mathbf{S}})$.
   - Compute for each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
     - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.
   - Set $\mathsf{crs} = (Q, n, s, (\mathsf{1MAFE.crs}_u)_{u \in [N]} K)$.
   - For each $\mathsf{id} \in [n], u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, (\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
     - Otherwise, $\mathsf{1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.

     Set $\mathsf{MPK}_{\mathsf{id}} = (\mathsf{1MAFE.mpk}_{\mathsf{id},u})_{u \in [N]}$ and $\mathsf{MSK}_{\mathsf{id}} = (\mathsf{1MAFE.msk}_{\mathsf{id},u})_{u \in \mathbf{S}_{\mathsf{corr}} \cup [j,N]}$.
   - For each $q \in [Q]$,
     - Compute $\left\{ \widehat{x}^u_{\mathsf{GID},\mathsf{id}} \right\}_{u \in \mathbf{S}_q} \leftarrow \mathcal{O}((\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta), \mathbf{S}_q)$.
     - For each $u \in \mathbf{S}_q$,
       - If $u \in \mathbf{S}_q \cap \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.sk}_u \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx} \in [n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
       - Otherwise, $\mathsf{1MAFE.sk}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
     - Set $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_q, \{\mathsf{1MAFE.sk}_u\}_{u \in \mathbf{S}_q})$.

   Create $\mathbf{Q} = \{(\mathsf{GID}_q, \mathsf{id}_q) : q \in [Q]\}$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id} \in [n]}, \{\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}}\}_{q \in [Q]}$ to $\mathcal{A}$.

2. $\mathcal{A}$ sends C. Do the following.

   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\{\widehat{C}^u\}_{u \in \mathbf{S}_{\mathsf{corr}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID} \in \Psi_n(\mathbf{Q}), u \in \mathbf{S}_{\mathsf{GID}}} \leftarrow \mathcal{O}(C, \{j^*_{\mathsf{GID}}\}_{\mathsf{GID} \in \Psi_n(\mathbf{Q})})$.
   - For each $u \in \mathbf{S}_{\mathsf{corr}}$, let $F^u(\cdot, \ldots, \cdot) = \mathsf{dCSF.UserComp}(\cdot, \ldots, \cdot, \widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id} \in [n]}, F^u)$.
     - Otherwise, if $\exists\, q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q} \setminus \{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, \mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \widehat{y}^u_{\mathsf{GID}_q})$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u \in [N]}$ to $\mathcal{A}$.

3. $\mathcal{A}$ output b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ, Q, n, s, and as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is an honest challenger for dCSF, $\mathcal{B}$ behaves like $\mathsf{Hyb}_{3,N+1}$ and if $\mathcal{O}$ is a simulator for dCSF, $\mathcal{B}$ behaves like Hyb_4. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between an honest challenger and simulator for dCSF. This contradicts our assumption for a secure BFFE scheme. Hence, $\mathsf{Hyb}_{3,N+1}$ and Hyb_4 are computationally indistinguishable. $\square$

E Proofs from Section 8

In this section, we provide the missing proofs from Section 8.

E.1 Hybrid Descriptions

The description of hybrids is as follows.

$\underline{\mathsf{Hyb}_0(1^\lambda)}$: This is $\mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)$ from Definition 3.5.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.
   - For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N]$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, $(\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.sv}_{\mathsf{id}})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \le Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - Compute $K \leftarrow \mathsf{niKE.KeyGen}(\{\mathsf{niKE.pv}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{id}, \mathsf{niKE.sv}_{\mathsf{id}})$.
   - Deterministically sample $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \leftarrow \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.
3. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.
4. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - Compute $K \leftarrow \mathsf{niKE.KeyGen}(\{\mathsf{niKE.pv}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{id}, \mathsf{niKE.sv}_{\mathsf{id}})$.
   - Deterministically sample $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \leftarrow \mathsf{PRF.Eval}(K, \mathsf{GID}_q)$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.
5. $\mathcal{A}$ outputs bit b'. Output b'.

$\underline{\mathsf{Hyb}_1(1^\lambda)}$: In this hybrid, we utilize the correctness of niKE and sample the key used for the PRF instantiation early. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.

   - For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N]$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, $(\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.sv}_{\mathsf{id}})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.

2. Choose $\mathsf{id} \xleftarrow{\$} [n]$, and sample $K \leftarrow \mathsf{niKE.KeyGen}(\{\mathsf{niKE.pv}_{\mathsf{idx}}\}_{\mathsf{idx}\in[n]}, \mathsf{id}, \mathsf{niKE.sv}_{\mathsf{id}})$. Use this key K in secret key generation.

3-5. Perform the rest of the steps same as Hyb_0.

$\underline{\mathsf{Hyb}_2(1^\lambda)}$: In this hybrid, we sample the key K uniformly randomly. The output distribution is computationally indistinguishable from niKE security. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.

   - For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N]$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, $(\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.

2. Choose $K \xleftarrow{\$} \{0,1\}^\lambda$. Use this key K in PRF evaluation.

3-5. Perform these steps same as Hyb_0.

$\underline{\mathsf{Hyb}_3(1^\lambda)}$: In this hybrid, we sample $\mathbf{S}_q$'s and Δ_q's uniformly randomly. The output distribution is computationally indistinguishable from PRF security. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.

   - For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N]$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, $(\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.

2. Initiate a dictionary $\mathbf{G}$ to be empty.

3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.

   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.

- For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

4. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.

5. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

6. $\mathcal{A}$ outputs bit b'. Output b'.

$\underline{\mathsf{Hyb}_4(1^\lambda)}$: In this hybrid, we sample $\mathbf{S}_q$'s and Δ_q's in advance and check if Lemmas 4.2 and 4.3 hold. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.
   - Initiate a dictionary $\mathbf{G}$ to be empty.
   - For each $q \in [Q]$, sample $\mathbf{S}_q \xleftarrow{\$} [N]$ of size D and $\Delta_q \xleftarrow{\$} [T]$ of size v. Initiate a counter g $= 1$. Compute for each $q, q' \in [Q], \mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| > t$, abort and output $\perp$. If for any $q \in [Q], \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$. Compute the string $(j^*_1, \ldots, j^*_Q)$ such that $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$.
   - For each $u \in [N], \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N], (\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - For each $\mathsf{id} \in [n], (\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.

2. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

3. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.

4. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

5. $\mathcal{A}$ outputs bit b'. Output b'.

$\mathsf{Hyb}_{5,j}(1^\lambda)$ for $j \in [N+1]$: In this hybrid, we simulate the first $j-1$ instantiations of 1MAFE using $\mathsf{sim}^u_{\mathsf{1MAFE}}$. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.
   - Initiate a dictionary $\mathbf{G}$ to be empty.
   - For each $q \in [Q]$, sample $\mathbf{S}_q \xleftarrow{\$} [N]$ of size D and $\Delta_q \xleftarrow{\$} [T]$ of size v. Initiate a counter g = 1. Compute for each $q, q' \in [Q], \mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| > t$, abort and output $\perp$. If for any $q \in [Q], \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$. Compute the string $(j^*_1, \ldots, j^*_Q)$ such that $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
     - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
     - Otherwise, $\mathsf{1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, $(\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{corr}}\cup[j,N]})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.

2. Initiate $\mathbf{Q}$ initially to empty.

3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.

- If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
- Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$.
- Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
- For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate
  - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

4. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot, \ldots, \cdot) = \mathsf{UserComp}(\cdot, \ldots, \cdot, \widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
     - Otherwise, if $\exists\ q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q}\setminus\{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, F^u(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.
5. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$,
     - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
     - Otherwise, if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, compute $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}, \widehat{F}^u(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
     - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}))$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.
6. $\mathcal{A}$ outputs bit b'. Output b'.

$\underline{\mathsf{Hyb}_6(1^\lambda)}$: This is $\mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{MA\text{-}FE}}}$ from Definition 3.5. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.
   - Initiate a dictionary $\mathbf{G}$ to be empty.

- For each $q \in [Q]$, sample $\mathbf{S}_q \xleftarrow{\$} [N]$ of size D and $\Delta_q \xleftarrow{\$} [T]$ of size v. Initiate a counter g = 1. Compute for each $q, q' \in [Q], \mathbf{S}_{\mathsf{corr}} = \bigcup_{q \neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| > t$, abort and output $\perp$. If for any $q \in [Q], \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'} \right) = \emptyset$, abort and output $\perp$. Compute the string $(j_1^*, \ldots, j_Q^*)$ such that $j_q^* \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'} \right)$. Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$.
- For each $u \in [N]$,
  - If $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
  - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.
- $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
- For each $\mathsf{id} \in [n], u \in [N]$,
  - If $u \in \mathbf{S}_{\mathsf{corr}}, (\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
  - Otherwise, $\mathsf{n_1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.
- For each $\mathsf{id} \in [n], (\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
- Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u \in [N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u \in \mathbf{S}_{\mathsf{corr}}})$.

Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u \in [N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id} \in [n]}$ to $\mathcal{A}$.

2. Initiate $\mathbf{Q}$ initially to empty.
3. Initiate $\mathsf{sim}_{\mathsf{dCSF}}(1^\lambda, 1^Q, 1^n, 1^s, \widetilde{\mathbf{S}})$.
4. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q, \mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q, \mathsf{id}_q}\right\}_{u \in \widetilde{\mathbf{S}}} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q, \mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate
     - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \widehat{x}^u_{\mathsf{GID}_q, \mathsf{id}_q})$.
     - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q, \mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q, \mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u \in \mathbf{S}_{\mathsf{id}_q}})$.
5. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\left\{\widehat{C}^u\right\}_{u \in \mathbf{S}_{\mathsf{corr}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID} \in \Psi_n(\mathbf{Q})} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}(C, \{j^*_{\mathsf{GID}}\}_{\mathsf{GID} \in \Psi_n(\mathbf{Q})})$.
   - For each $u \in \widetilde{\mathbf{S}}$, let $F^u(\cdot, \ldots, \cdot) = \mathsf{UserComp}(\cdot, \ldots, \cdot, \widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id} \in [n]}, F^u)$.
     - Otherwise, if $\exists\ q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q} \setminus \{(\mathsf{GID}_q, \mathsf{id}_q)\}, n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, \mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \widehat{y}^u_{\mathsf{GID}_q})$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u \in [N]}$ to $\mathcal{A}$.
6. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q, \mathsf{id}_q})$. $\mathcal{C}$ does the following.

- If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
- If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, V_q = (\mathsf{GID}_q, j^*_{\mathsf{GID}_q}, X = (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}), F^u(X))$.
- Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{id}_q}}, \{\widehat{y}^u_{\mathsf{GID}_q}\}_{u\in\mathbf{S}_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}.(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q}, V_q)$.
- For each $u \in \mathbf{S}_{\mathsf{id}_q}$,
  - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Otherwise, if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, \mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}, \widehat{y}^u_{\mathsf{GID}_q})$.
  - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}))$.

Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

7. $\mathcal{A}$ outputs bit b'. Output b'.

E.2 Proofs of Claims

The proof of claim from Section 8 is as follows.

Proof of Claim 8.3. Note that the only difference between Hyb_1 and Hyb_2 is that we sample the PRF key uniformly randomly. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids Hyb_1 and Hyb_2, we will construct an adversary $\mathcal{B}$ that can break the security of niKE. More formally, if $\mathcal{A}$'s advantage is such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_1}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_2}\right]\right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction which distinguishes between oracle access between Expt_0 and Expt_1 for the niKE scheme. More formally,

$$\left|\Pr\left[1 \leftarrow \mathcal{B}^{\mathsf{niKE},0}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathcal{B}^{\mathsf{niKE},1}\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda):}$

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.
   - For each $u \in [N], \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - $K, \mathsf{niKE.crs}, (\mathsf{niKE.pv}_{\mathsf{id}})_{\mathsf{id}\in[n]} \leftarrow \mathcal{O}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n], u \in [N], (\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. Use the key K obtained above in PRF evaluation.
3. Perform the remaining steps exactly as in Hyb_0.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ, Q, n, s, and as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is Expt_0 for niKE, $\mathcal{B}$ behaves like Hyb_1 and if $\mathcal{O}$ is Expt_1 for niKE, $\mathcal{B}$ behaves like Hyb_2. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between an Expt_0 and Expt_1 for niKE. This contradicts our assumption for a secure niKE scheme. Hence, Hyb_1 and Hyb_2 are computationally indistinguishable. □

Proof of Claim 8.4. Note that in Hyb_2 and Hyb_3, the only difference is that we sample the PRF values uniformly randomly. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids Hyb_2 and Hyb_3, we will construct an adversary $\mathcal{B}$ that can break the security of PRF. More formally, if $\mathcal{A}$'s advantage is such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_2}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_3}\right]\right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction which distinguishes between oracle access between Expt_0 and Expt_1 for the PRF scheme. More formally,

$$\left|\Pr\left[1 \leftarrow \mathcal{B}^{\mathcal{O}_0(\cdot)}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathcal{B}^{\mathcal{O}_1(\cdot)}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda):}$

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.
   - For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N]$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - For each $\mathsf{id} \in [n]$, $(\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - Sample $\mathbf{S}_q, \Delta_q \leftarrow \mathcal{O}(\mathsf{GID}_q)$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.
3. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.
4. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - Sample $\mathbf{S}_q, \Delta_q \leftarrow \mathcal{O}(\mathsf{GID}_q)$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.
5. $\mathcal{A}$ outputs bit b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is $\mathcal{O}_0$ for PRF, $\mathcal{B}$ behaves like Hyb_2 and if $\mathcal{O}$ is $\mathcal{O}_1$ for PRF, $\mathcal{B}$ behaves like Hyb_3. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between an $\mathcal{O}_0$ and $\mathcal{O}_1$ for PRF. This contradicts our assumption for a secure PRF scheme. Hence, Hyb_2 and Hyb_3 are computationally indistinguishable. □

Proof of Claim 8.7. Note that in $\mathsf{Hyb}_{5,j}$ and $\mathsf{Hyb}_{5,j+1}$, the only difference is in the j-th instantiation of 1MAFE. Moreover, if $j \in \mathbf{S}_{\mathsf{corr}}$, we do not simulate this instantiation. WLOG, assume that $j \notin \mathbf{S}_{\mathsf{corr}}$. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{5,j}$ and $\mathsf{Hyb}_{5,j+1}$, we will construct an adversary $\mathcal{B}$ that can break the security of 1MAFE. More formally, if $\mathcal{A}$'s advantage is such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_{5,j}}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_{5,j+1}}\right]\right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction which distinguishes between oracle access between Expt_0 and Expt_1 for the 1MAFE scheme. More formally,

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathsf{1MAFE},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathsf{1MAFE},\mathcal{B},\mathsf{sim}_{\mathsf{1MAFE}}}\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda):}$

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.
   - Initiate a dictionary $\mathbf{G}$ to be empty.
   - For each $q \in [Q]$, sample $\mathbf{S}_q \xleftarrow{\$} [N]$ of size D and $\Delta_q \xleftarrow{\$} [T]$ of size v. Initiate a counter g = 1. Compute for each $q, q' \in [Q], \mathbf{S}_{\mathsf{corr}} = \bigcup_{q \neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| > t$, abort and output $\perp$. If for any $q \in [Q], \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$. Compute the string $(j_1^*, \ldots, j_Q^*)$ such that $j_q^* \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'}\right)$. Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u > j$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
     - Otherwise, if $u = j$, $\mathsf{1MAFE.crs}_u, (\mathsf{1MAFE.mpk}_{u,\mathsf{id}})_{\mathsf{id} \in [n]} \leftarrow \mathcal{O}(1^\lambda, 1^n)$.
     - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u > j$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
     - Otherwise, $\mathsf{1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n], (\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u \in [N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u \in \mathbf{S}_{\mathsf{corr}} \cup [j,N]})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u \in [N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id} \in [n]}$ to $\mathcal{A}$.
2. Initiate $\mathbf{Q}$ initially to empty.
3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.

- Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$.
- Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
- For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate
  - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}$ or $u > j$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Otherwise, if $u = j$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathcal{O}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

4. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u > j$, $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
     - Otherwise, if $u = j$, $\mathsf{ct}_u \leftarrow \mathcal{O}(\widehat{C}^u)$.
     - Otherwise, if $\exists\ q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q}\setminus\{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, \mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, F^u(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.

5. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$,
     - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}$ or $u > j$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
     - Otherwise, if $u = j$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathcal{O}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
     - Otherwise, if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, compute $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}, \widehat{F}^u(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
     - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}))$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

6. $\mathcal{A}$ outputs bit b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ, Q, n, s, and as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is Expt_0, $\mathcal{B}$ behaves like $\mathsf{Hyb}_{5,j}$ and if $\mathcal{O}$ is Expt_1, $\mathcal{B}$ behaves like $\mathsf{Hyb}_{5,j+1}$. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between an honest challenger and simulator for 1MAFE. This contradicts our assumption for a secure NCE scheme. Hence, $\mathsf{Hyb}_{5,j}$ and $\mathsf{Hyb}_{5,j+1}$ are computationally indistinguishable. □

Proof of Claim 8.8. Note that the only difference between $\mathsf{Hyb}_{5,N+1}$ and Hyb_6 is that we simulate the dCSF instantiation. Assuming that there exists an adversary $\mathcal{A}$ that can distinguish between the hybrids $\mathsf{Hyb}_{5,N+1}$ and Hyb_6, we will construct an adversary $\mathcal{B}$ that can break the security of dCSF. More formally, if $\mathcal{A}$'s advantage is such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_{5,N+1}}\right] - \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{Hyb}_6}\right]\right| > \mathsf{negl}(\lambda)$$

We will construct a polynomial time reduction which distinguishes between oracle access between Expt_0 and Expt_1 for the dCSF scheme. More formally,

$$\left|\Pr\left[1 \leftarrow \mathsf{Expt}_0^{\mathsf{dCSF},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[1 \leftarrow \mathsf{Expt}_1^{\mathsf{dCSF},\mathcal{B},\mathsf{sim}_{\mathsf{dCSF}}}\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda):}$

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, and the maximum size of the challenge circuit s. $\mathcal{C}$ does the following.
   - Initiate a dictionary $\mathbf{G}$ to be empty.
   - For each $q \in [Q]$, sample $\mathbf{S}_q \xleftarrow{\$} [N]$ of size D and $\Delta_q \xleftarrow{\$} [T]$ of size v. Initiate a counter g = 1. Compute for each $q, q' \in [Q], \mathbf{S}_{\mathsf{corr}} = \bigcup_{q \neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| > t$, abort and output $\perp$. If for any $q \in [Q], \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$. Compute the string $(j_1^*, \ldots, j_Q^*)$ such that $j_q^* \in \Delta_q \setminus \left(\bigcup_{q \neq q'} \Delta_{q'}\right)$. Set $\widetilde{\mathbf{S}} = [N] \setminus \mathbf{S}_{\mathsf{corr}}$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
     - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.
   - $\mathsf{niKE.crs} \leftarrow \mathsf{niKE.Setup}(1^\lambda)$.
   - For each $\mathsf{id} \in [n], u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, (\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
     - Otherwise, $\mathsf{n_1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.
   - For each $\mathsf{id} \in [n], (\mathsf{niKE.pv}_{\mathsf{id}}, \mathsf{niKE.sv}_{\mathsf{id}}) \leftarrow \mathsf{niKE.Publish}(\mathsf{niKE.crs}, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u \in [N]}, \mathsf{niKE.pv}_{\mathsf{id}})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u \in \mathbf{S}_{\mathsf{corr}}})$.

   Send $\mathsf{crs} = ((\mathsf{1MAFE.crs}_u)_{u \in [N]}, \mathsf{niKE.crs}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id} \in [n]}$ to $\mathcal{A}$.
2. Initiate $\mathbf{Q}$ initially to empty.
3. Initiate $\mathcal{O}(1^\lambda, 1^Q, 1^n, 1^s, \widetilde{\mathbf{S}})$.
4. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u \in \widetilde{\mathbf{S}}} \leftarrow \mathcal{O}(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate
     - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx} \in [n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

– Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

5. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in\mathbf{S}_{\mathsf{corr}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})} \leftarrow \mathcal{O}(C, \{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})})$.
   - For each $u \in \widetilde{\mathbf{S}}$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - For each $u \in [N]$,
     – If $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
     – Otherwise, if $\exists\ q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q}\setminus\{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q,\mathsf{id}_q) = 1, \mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \widehat{y}^u_{\mathsf{GID}_q})$.
     – Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.

6. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, V_q = (\mathsf{GID}_q, j^*_{\mathsf{GID}_q}, X = (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}), F^u(X))$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{id}_q}}, \{\widehat{y}^u_{\mathsf{GID}_q}\}_{u\in\mathbf{S}_{\mathsf{id}_q}} \leftarrow \mathcal{O}.(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q}, V_q)$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$,
     – If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
     – Otherwise, if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, \mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}, \widehat{y}^u_{\mathsf{GID}_q})$.
     – Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}))$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

7. $\mathcal{A}$ outputs bit b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the parameters for λ, Q, n, s, and as $\mathcal{A}$ runs in polynomial time too. If the oracle $\mathcal{O}$ is an honest challenger for dCSF, $\mathcal{B}$ behaves like $\mathsf{Hyb}_{5,N+1}$ and if $\mathcal{O}$ is a simulator for dCSF, $\mathcal{B}$ behaves like Hyb_6. As $\mathcal{A}$ can distinguish between them with non-negligible advantage we can see that $\mathcal{B}$, with non-negligible probability distinguishes between an honest challenger and simulator for dCSF. This contradicts our assumption for a secure BFFE scheme. Hence, $\mathsf{Hyb}_{5,N+1}$ and Hyb_6 are computationally indistinguishable.

□

F Proofs from Section 9

In this section, we provide the missing proofs from Section 9.

F.1 Hybrid Descriptions

The description of hybrids is as follows.

$\mathsf{Hyb}_0(1^\lambda)$: This is $\mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{A},\mathcal{C}}(1^\lambda)$ from Definition 3.5.

1. $\mathcal{A}$ sends Q, n_1, n_2, s to $\mathcal{C}$.
2. $\mathcal{C}$ runs $\mathsf{n}_1\mathsf{MAFE.crs} \leftarrow \mathsf{n}_1\mathsf{MAFE.GlobalSetup}(1^\lambda, 1^Q, 1^{n_1}, 1^s)$ and $\mathsf{n}_2\mathsf{MAFE.crs} \leftarrow \mathsf{n}_2\mathsf{MAFE.GlobalSetup}(1^\lambda, 1^Q, 1^{n_2}, 1^s)$. Then,
   - $\forall\ \mathsf{id} \in [n_1], (\mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}}, \mathsf{n}_1\mathsf{MAFE.msk}_{\mathsf{id}}) \leftarrow \mathsf{n}_1\mathsf{MAFE.AuthSetup}(\mathsf{id})$.
   - $\forall\ \mathsf{id} \in [n_1+1, n_1+n_2], (\mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}}, \mathsf{n}_2\mathsf{MAFE.msk}_{\mathsf{id}}) \leftarrow \mathsf{n}_2\mathsf{MAFE.AuthSetup}(\mathsf{id} - n_1)$.

   Set $\mathsf{crs} = (\mathsf{n}_1\mathsf{MAFE.crs}, \mathsf{n}_2\mathsf{MAFE.crs})$ and $(\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) = (\mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}}, \mathsf{n}_1\mathsf{MAFE.msk}_{\mathsf{id}})$ for each $\mathsf{id} \in [n_1]$ and $(\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) = (\mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}}, \mathsf{n}_2\mathsf{MAFE.msk}_{\mathsf{id}})$ for each $\mathsf{id} \in [n_1+1, n_1+n_2]$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1+n_2]}$ to $\mathcal{A}$.
3. Let $\mathcal{A}$ make $Q_1 \le Q$ queries in this phase. $\forall\ q \in [Q_1]$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_1\mathsf{MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1,\ n_1+n_2], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_2\mathsf{MAFE.KeyGen}(\mathsf{id}_q - n_1, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.
4. $\mathcal{A}$ submits challenge circuit C.
   - Sample a PRF key $K \xleftarrow{\$} \{0,1\}^\lambda$.
   - Send $\mathsf{CT} \leftarrow \mathsf{n}_2\mathsf{MAFE.Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1+1,\ n_1+n_2]}, F)$ to $\mathcal{A}$, where F is defined in Figure 6.
5. $\forall\ q \in \{Q_1+1, \ldots, Q\}$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_1\mathsf{MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1,\ n_1+n_2], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_2\mathsf{MAFE.KeyGen}(\mathsf{id}_q - n_1, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs its guess b'. Output b'.

$\mathsf{Hyb}_1(1^\lambda)$: In this hybrid, we will simulate $\mathsf{n}_2\mathsf{MAFE}$ using the simulator sim_2. The changes are highlighted in red.

1. $\mathcal{A}$ sends Q, n_1, n_2, s to $\mathcal{C}$.
2. $\mathcal{C}$ runs $\mathsf{n}_1\mathsf{MAFE.crs} \leftarrow \mathsf{n}_1\mathsf{MAFE.GlobalSetup}(1^\lambda, 1^Q, 1^{n_1}, 1^s)$ and $\mathsf{n}_2\mathsf{MAFE.crs} \leftarrow \mathsf{sim}_2(1^\lambda, 1^Q, 1^{n_2}, 1^s)$. Then,
   - $\forall\ \mathsf{id} \in [n_1], (\mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}}, \mathsf{n}_1\mathsf{MAFE.msk}_{\mathsf{id}}) \leftarrow \mathsf{n}_1\mathsf{MAFE.AuthSetup}(\mathsf{id})$.
   - $\forall\ \mathsf{id} \in [n_1+1, n_1+n_2], \mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}} \leftarrow \mathsf{sim}_2(\mathsf{id} - n_1)$.

   Set $\mathsf{crs} = (\mathsf{n}_1\mathsf{MAFE.crs}, \mathsf{n}_2\mathsf{MAFE.crs})$ and $(\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) = (\mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}}, \mathsf{n}_1\mathsf{MAFE.msk}_{\mathsf{id}})$ for each $\mathsf{id} \in [n_1]$ and $\mathsf{MPK}_{\mathsf{id}} = \mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}}$ for each $\mathsf{id} \in [n_1+1, n_1+n_2]$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1+n_2]}$ to $\mathcal{A}$.
3. Let $\mathcal{A}$ make $Q_1 \le Q$ queries in this phase. $\forall\ q \in [Q_1]$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_1\mathsf{MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1,\ n_1+n_2], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.

4. $\mathcal{A}$ submits challenge circuit C.
   - Sample a PRF key $K \xleftarrow{\$} \{0,1\}^\lambda$.
   - For each $\mathsf{id} \in [n_1 + n_2]$, create $\chi^{(\mathsf{id})} = \{x_{\mathsf{id}_q} : q \in [Q_1], \mathsf{id}_q = \mathsf{id}\}$.
   - Compute $\mathbf{V} = \left\{(X, F(X, K, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, C)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}\right\}$.
   - Send $\mathsf{CT} \leftarrow \mathsf{sim}_2(1^{|F|}, \mathbf{V})$ to $\mathcal{A}$.
5. $\forall\ q \in \{Q_1 + 1, \ldots, Q\}$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1]$, $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_1\mathsf{MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1,\ n_1+n_2]$, compute $\chi_q^{\mathsf{id}_q} = \left\{x_{\mathsf{id}_q}\right\}$ and $V_q = \big\{(X, F(X, K, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, C)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi_q^{\mathsf{id}_q} \times \ldots \times \chi^{(n_1+n_2)}\big\}$. $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q}, V_q)$. Update $\chi^{(\mathsf{id}_q)} = \chi^{(\mathsf{id}_q)} \cup \chi_q^{\mathsf{id}_q}$ and $\mathbf{V} = \mathbf{V} \cup V_q$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs its guess b'. Output b'.

$\underline{\mathsf{Hyb}_2(1^\lambda)}$: In this hybrid, we will change the circuit F that is simulated so that it uses a hardwired string instead of evaluating the PRF as part of it's evaluation. The changes are highlighted in red.

1-3. Perform these steps same as Hyb_1.

4. $\mathcal{A}$ submits challenge circuit C. Sample a PRF key $K \xleftarrow{\$} \mathsf{PRF.Gen}(1^\lambda)$.
   - Perform these steps same as Hyb_1.
   - Create $\mathbf{V} = \{(X, \widetilde{F}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, \mathsf{PRF.Eval}(K, X), C)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}\}$, where $\widetilde{F}$ is as described in Figure 9.
   - Send $\mathsf{CT} \leftarrow \mathsf{sim}_2(1^{|F|}, \mathbf{V})$ to $\mathcal{A}$.
5. $\forall\ q \in \{Q_1 + 1, \ldots, Q\}$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1]$, $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_1\mathsf{MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1 + 1, n_1 + n_2]$, compute $\chi_q^{\mathsf{id}_q} = \{x_{\mathsf{id}_q}\}$ and $V_q = \{(X, \widetilde{F}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, \mathsf{PRF.Eval}(K, X), C)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi_q^{\mathsf{id}_q} \times \ldots \times \chi^{(n_1+n_2)}\}$. Sample $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q}, V_q)$. Update $\chi^{\mathsf{id}_q} = \chi^{\mathsf{id}_q} \cup \chi_q^{\mathsf{id}_q}$ and $\mathbf{V} = \mathbf{V} \cup V_q$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs its guess b'. Output b'.

$\underline{\widetilde{F}(x_1, \ldots, x_{n_2}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, \underline{R}, C)}$:

- Output $\mathsf{n}_1\mathsf{MAFE.Enc}(\{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, C(\cdot, \ldots, \cdot, x_1, \ldots, x_{n_2}); \underline{R})$.

Figure 9: Description of the circuit $\widetilde{F}$.

$\underline{\mathsf{Hyb}_3(1^\lambda)}$: In this hybrid, we will replace all PRF evaluations with uniformly random strings. The changes are highlighted in red.

1. Perform this step same as Hyb_2.
2. Create a dictionary $\mathbf{R}$ which is initially empty.

3-4. Perform these steps same as Hyb_2.

5. $\mathcal{A}$ submits challenge circuit C.

   - ...
   - For each $X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}$ sample a uniformly random string, $R_X \leftarrow \{0,1\}^L$ and add (X, R_X) to $\mathbf{R}$.
   - Create $\mathbf{V} = \{(X, \widetilde{F}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, R_X, C)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}\}$, where $\widetilde{F}$ is as described in Figure 9.
   - Send $\mathsf{CT} \leftarrow \mathsf{sim}_2(1^{|F|}, \mathbf{V})$ to $\mathcal{A}$.

6. $\forall\ q \in \{Q_1+1, \ldots, Q\}$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.

   - If $\mathsf{id}_q \in [n_1]$, $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_1\mathsf{MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q} x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1, n_1+n_2]$, compute $\chi_q^{\mathsf{id}_q} = \{x_{\mathsf{id}_q}\}$ and $V_q = \{(X, \widetilde{F}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, \mathsf{PRF.Eval}(K, X), C)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi_q^{\mathsf{id}_q} \times \ldots \times \chi^{(n_1+n_2)}\}$, where R_X is sampled uniformly or reused from $\mathbf{R}$. Sample $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q}, V_q)$. Update $\chi^{\mathsf{id}_q} = \chi^{\mathsf{id}_q} \cup \chi_q^{\mathsf{id}_q}$ and $\mathbf{V} = \mathbf{V} \cup V_q$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.

7. $\mathcal{A}$ outputs its guess b'. Output b'.

$\underline{\mathsf{Hyb}_4(1^\lambda)}$: This is $\mathsf{Expt}_1^{\mathsf{MA\text{-}FE}, \mathcal{A}, \mathsf{sim}_{\mathsf{MA\text{-}FE}}}$ from Definition 3.5. The changes are highlighted in red.

1. $\mathcal{A}$ sends Q, n_1, n_2, s to $\mathcal{C}$.
2. Create a dictionary $\mathbf{R}$ which is initially empty.
3. $\mathcal{C}$ runs $\mathsf{n}_1\mathsf{MAFE.crs} \leftarrow \mathsf{sim}_1(1^\lambda, 1^Q, 1^{n_1}, 1^s)$ and $\mathsf{n}_2\mathsf{MAFE.crs} \leftarrow \mathsf{sim}_2(1^\lambda, 1^Q, 1^{n_2}, 1^s)$. Then,

   - $\forall\ \mathsf{id} \in [n_1]$, $\mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}} \leftarrow \mathsf{sim}_1(\mathsf{id})$.
   - $\forall\ \mathsf{id} \in [n_1+1, n_1+n_2]$, $\mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}} \leftarrow \mathsf{sim}_2(\mathsf{id} - n_1)$.

   Set $\mathsf{crs} = (\mathsf{n}_1\mathsf{MAFE.crs}, \mathsf{n}_2\mathsf{MAFE.crs})$ and $\mathsf{MPK}_{\mathsf{id}} = \mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}}$ for each $\mathsf{id} \in [n_1]$ and $\mathsf{MPK}_{\mathsf{id}} = \mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}}$ for each $\mathsf{id} \in [n_1+1, n_1+n_2]$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1+n_2]}$ to $\mathcal{A}$.

4. Let $\mathcal{A}$ make $Q_1 \le Q$ queries in this phase. $\forall\ q \in [Q_1]$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.

   - If $\mathsf{id}_q \in [n_1]$, $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_1(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1,\ n_1+n_2]$, $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.

5. $\mathcal{A}$ submits challenge circuit C.

   - For each $\mathsf{id} \in [n_1+n_2]$, create $\chi^{(\mathsf{id})} = \{x_{\mathsf{id}_q} : q \in [Q_1], \mathsf{id}_q = \mathsf{id}\}$.
   - Create $\mathbf{V}_1 = \{(X, C(X)) : \forall\ X \in \chi^{(1)} \times \ldots \times \chi^{(n_1+n_2)}\}$.
   - For each $X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}$ sample a uniformly random string, $R_X \leftarrow \{0,1\}^L$ and add (X, R_X) to $\mathbf{R}$.
   - Create $\mathbf{V}_2 = \{(X, \widetilde{F}_{\mathbf{V}_1}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, R_X, 1^{|C|})) : X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}\}$, where $\widetilde{F}_{\mathbf{V}_1}$ is as described in Figure 10.
   - Send $\mathsf{CT} \leftarrow \mathsf{sim}_2(1^{|F|}, \mathbf{V}_2)$ to $\mathcal{A}$.

6. $\forall\ q \in \{Q_1+1, \ldots, Q\}$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.

- If $\mathsf{id}_q \in [n_1]$, compute $\chi_q^{\mathsf{id}_q} = \{x_{\mathsf{id}_q}\}$ and $V_q^{(1)} = \{(X, C(X) : X \in \chi^{(1)} \times \ldots \times \chi_q^{\mathsf{id}_q} \times \ldots \times \chi^{(n_1+n_2)}\}$. Sample $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_1(\mathsf{id}_q, x_{\mathsf{id}_q}, V_q^{(1)})$. Update $\chi^{\mathsf{id}_q} = \chi^{\mathsf{id}_q} \cup \chi_q^{\mathsf{id}_q}$ and $\mathbf{V}_1 = \mathbf{V}_1 \cup V_q^{(1)}$.
- Otherwise, if $\mathsf{id}_q \in [n_1 + 1, n_1 + n_2]$, compute $\chi_q^{\mathsf{id}_q} = \{x_{\mathsf{id}_q}\}$ and $V_q^{(2)} = \{(X, \widetilde{F}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, R_X, 1^{|C|})) : X \in \chi^{(n_1+1)} \times \ldots \times \chi_q^{\mathsf{id}_q} \times \ldots \times \chi^{(n_1+n_2)}\}$ where R_X is sampled uniformly or reused from $\mathbf{R}$. Sample $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q}, V_q^{(2)})$. Update $\chi^{\mathsf{id}_q} = \chi^{\mathsf{id}_q} \cup \chi_q^{\mathsf{id}_q}$ and $\mathbf{V}_2 = \mathbf{V}_2 \cup V_q^{(2)}$.

Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.

7. $\mathcal{A}$ outputs its guess b'. Output b'.

$\underline{\widetilde{F}_{\mathbf{V}}(x_1, \ldots, x_{n_2}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, R, 1^{|C|}) :}$

- Compute $\mathbf{V}' = \{(X, C(X)) : (X, C(X)) \in \mathbf{V} \text{ and } X = (\cdot, \ldots, \cdot, x_1, \ldots, x_{n_2})\}$.
- Output $\mathsf{sim}_1(1^{|C|}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, \mathbf{V}'; R)$.

Figure 10: Description of the circuit $\widetilde{F}_{\mathbf{V}}$.

F.2 Proofs of Claims

The proof of claim from Section 9 is as follows.

Proof of Claim 9.2. Note that the only difference between $\mathsf{Hyb}_0(1^\lambda)$ and $\mathsf{Hyb}_1(1^\lambda)$ is that we are simulating $\mathsf{n}_2\mathsf{MAFE}$. Assume for the sake of contradiction that there exists a PPT adversary $\mathcal{A}$ that can distinguish $\mathsf{Hyb}_0(1^\lambda)$ and $\mathsf{Hyb}_1(1^\lambda)$. That is,

$$\left|\Pr\left[0 \leftarrow \mathcal{A}^{\mathsf{Hyb}_0}(1^\lambda)\right] - \Pr\left[0 \leftarrow \mathcal{A}^{\mathsf{Hyb}_1}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

We can construct a PPT adversary $\mathcal{B}$ that can distinguish between oracle access to Expt_0 and Expt_1 for the $\mathsf{n}_1\mathsf{MAFE}$ scheme. More formally,

$$\left|\Pr\left[0 \leftarrow \mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[0 \leftarrow \mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{B},\mathsf{sim}_2}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda)}$:

1. $\mathcal{A}$ sends Q, n_1, n_2, s to $\mathcal{C}$.
2. $\mathcal{C}$ runs $\mathsf{n}_1\mathsf{MAFE.crs} \leftarrow \mathsf{n}_1\mathsf{MAFE.GlobalSetup}(Q, n_1, s)$ and $\mathsf{n}_2\mathsf{MAFE.crs} \leftarrow \mathcal{O}(1^\lambda, 1^Q, 1^{n_2}, 1^s)$. Then,
   - $\forall\ \mathsf{id} \in [n_1], (\mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}}, \mathsf{n}_1\mathsf{MAFE.msk}_{\mathsf{id}}) \leftarrow \mathsf{n}_1\mathsf{MAFE.AuthSetup}(\mathsf{id})$.
   - $\forall\ \mathsf{id} \in [n_1 + 1, n_1 + n_2], \mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}} \leftarrow \mathcal{O}(\mathsf{id} - n_1)$.

   Set $\mathsf{crs} = (\mathsf{n}_1\mathsf{MAFE.crs}, \mathsf{n}_2\mathsf{MAFE.crs})$ and $(\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) = (\mathsf{n}_1\mathsf{MAFE.mpk}_{\mathsf{id}}, \mathsf{n}_1\mathsf{MAFE.msk}_{\mathsf{id}})$ for each $\mathsf{id} \in [n_1]$ and $\mathsf{MPK}_{\mathsf{id}} = \mathsf{n}_2\mathsf{MAFE.mpk}_{\mathsf{id}}$ for each $\mathsf{id} \in [n_1 + 1, n_1 + n_2]$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1+n_2]}$ to $\mathcal{A}$.
3. Let $\mathcal{A}$ make $Q_1 \le Q$ queries in this phase. $\forall\ q \in [Q_1], \mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n}_1\mathsf{MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1 + 1,\ n_1 + n_2], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathcal{O}(\mathsf{id}_q - n_1, x_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.
4. $\mathcal{A}$ submits challenge circuit C.

- Sample a PRF key $K \xleftarrow{\$} \{0,1\}^\lambda$.
- Create the circuit F as defined in Figure 6.
- Send $\mathsf{CT} \leftarrow \mathcal{O}(F)$ to $\mathcal{A}$.

5. $\forall\ q \in \{Q_1+1, \ldots, Q\}$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n_1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1,\ n_1+n_2]$, compute $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathcal{O}(\mathsf{id}_q - n_1, x_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.
6. $\mathcal{A}$ outputs its guess b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the running time of λ, Q, n_1, n_2, s, and as $\mathcal{A}$ runs in polynomial time. If $\mathcal{O}$ is the honest challenger for $\mathsf{n_2MAFE}$, the output distribution of $\mathcal{B}$ is exactly the same as Hyb_0. On the other hand, if $\mathcal{O}$ is a simulator for $\mathsf{n_2MAFE}$, $\mathcal{B}$'s output distribution is same as Hyb_1. As $\mathcal{A}$ can distinguish between Hyb_0 and Hyb_1 with non-negligible probability, $\mathcal{B}$ can also distinguish between the real and ideal worlds for $\mathsf{n_2MAFE}$. This contradicts the adaptive simulation security of $\mathsf{n_2MAFE}$. Hence, Hyb_0 and Hyb_1 are indistinguishable. □

Proof of Claim 9.4. Note that the only difference between $\mathsf{Hyb}_2(1^\lambda)$ and $\mathsf{Hyb}_3(1^\lambda)$ is that we are substituting PRF evaluations with uniformly random strings. Assume for the sake of contradiction that there exists a PPT adversary $\mathcal{A}$ that can distinguish $\mathsf{Hyb}_2(1^\lambda)$ and $\mathsf{Hyb}_3(1^\lambda)$. That is,

$$\left|\Pr\left[0 \leftarrow \mathcal{A}^{\mathsf{Hyb}_2}(1^\lambda)\right] - \Pr\left[0 \leftarrow \mathcal{A}^{\mathsf{Hyb}_3}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

We can construct a PPT adversary $\mathcal{B}$ that can distinguish between Expt_0 and Expt_1 for the PRF scheme. More formally,

$$\left|\Pr\left[0 \leftarrow \mathsf{Expt}_0^{\mathsf{PRF},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[0 \leftarrow \mathsf{Expt}_1^{\mathsf{PRF},\mathcal{B},\mathcal{C}}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}^{\mathcal{O}}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda)}$:

1. $\mathcal{A}$ sends Q, n_1, n_2, s to $\mathcal{C}$.
2. $\mathcal{C}$ runs $\mathsf{n_1MAFE.crs} \leftarrow \mathsf{n_1MAFE.GlobalSetup}(1^\lambda, 1^Q, 1^{n_1}, 1^s)$ and $\mathsf{n_2MAFE.crs} \leftarrow \mathsf{sim}_2(1^\lambda, 1^Q, 1^{n_2}, 1^s)$. Then,
   - $\forall\ \mathsf{id} \in [n_1], (\mathsf{n_1MAFE.mpk}_{\mathsf{id}}, \mathsf{n_1MAFE.msk}_{\mathsf{id}}) \leftarrow \mathsf{n_1MAFE.AuthSetup}(\mathsf{id})$.
   - $\forall\ \mathsf{id} \in [n_1+1, n_1+n_2], \mathsf{n_2MAFE.mpk}_{\mathsf{id}} \leftarrow \mathsf{sim}_2(\mathsf{id} - n_1)$.

   Set $\mathsf{crs} = (\mathsf{n_1MAFE.crs}, \mathsf{n_2MAFE.crs})$ and $(\mathsf{MPK}_{\mathsf{id}}, \mathsf{MSK}_{\mathsf{id}}) = (\mathsf{n_1MAFE.mpk}_{\mathsf{id}}, \mathsf{n_1MAFE.msk}_{\mathsf{id}})$ for each $\mathsf{id} \in [n_1]$ and $\mathsf{MPK}_{\mathsf{id}} = \mathsf{n_2MAFE.mpk}_{\mathsf{id}}$ for each $\mathsf{id} \in [n_1+1, n_1+n_2]$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id} \in [n_1+n_2]}$ to $\mathcal{A}$.
3. Let $\mathcal{A}$ make $Q_1 \leq Q$ queries in this phase. $\forall\ q \in [Q_1], \mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n_1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q}, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1,\ n_1+n_2], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.
4. $\mathcal{A}$ submits challenge circuit C.
   - For each $\mathsf{id} \in [n_1+n_2]$, create $\chi^{(\mathsf{id})} = \{x_{\mathsf{id}_q} : q \in [Q_1], \mathsf{id}_q = \mathsf{id}\}$.
   - Create $\mathbf{V} = \{(X, \widetilde{F}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id} \in [n_1]}, \mathcal{O}(X), C)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}\}$, where $\widetilde{F}$ is as described in Figure 9.

- Send $\mathsf{CT} \leftarrow \mathsf{sim}_2(1^{|F|}, \mathbf{V})$ to $\mathcal{A}$.

5. $\forall\ q \in \{Q_1+1, \ldots, Q\}, \mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{n_1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{MSK}_{\mathsf{id}_q} x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1, n_1+n_2]$, compute $\chi_q^{\mathsf{id}_q} = \{x_{\mathsf{id}_q}\}$ and $V_q = \{(X, \widetilde{F}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1]}, \mathcal{O}(X), C)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi_q^{\mathsf{id}_q} \times \ldots \times \chi^{(n_1+n_2)}\}$. Sample $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q}, V_q)$. Update $\chi^{\mathsf{id}_q} = \chi^{\mathsf{id}_q} \cup \chi_q^{\mathsf{id}_q}$ and $\mathbf{V} = \mathbf{V} \cup V_q$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.

6. $\mathcal{A}$ outputs its guess b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the running time of $\mathcal{A}$, $\mathcal{O}$, and λ. If $\mathcal{O}$ is Expt_0 for the PRF scheme, the output distribution of $\mathcal{B}$ is exactly the same as Hyb_2. On the other hand, if $\mathcal{O}$ is Expt_1 for the PRF scheme, $\mathcal{B}$'s output distribution is same as Hyb_3. As $\mathcal{A}$ can distinguish between Hyb_2 and Hyb_3 with non-negligible probability, $\mathcal{B}$ can also distinguish between the oracles with non-negligible probability. This contradicts the security of PRF. Hence, Hyb_2 and Hyb_3 are indistinguishable. □

Proof of Claim 9.5. Note that the only difference between $\mathsf{Hyb}_3(1^\lambda)$ and $\mathsf{Hyb}_4(1^\lambda)$ is that we are simulating the $\mathsf{n_1MAFE}$ instantiation. Assume for the sake of contradiction that there exists a PPT adversary $\mathcal{A}$ that can distinguish $\mathsf{Hyb}_3(1^\lambda)$ and $\mathsf{Hyb}_4(1^\lambda)$. That is,

$$\left|\Pr\left[0 \leftarrow \mathcal{A}^{\mathsf{Hyb}_3}(1^\lambda)\right] - \Pr\left[0 \leftarrow \mathcal{A}^{\mathsf{Hyb}_4}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

We can construct a PPT adversary $\mathcal{B}$ that can distinguish between Expt_0 and Expt_1 for the $\mathsf{n_1MAFE}$ scheme. More formally,

$$\left|\Pr\left[0 \leftarrow \mathsf{Expt}_0^{\mathsf{MA\text{-}FE},\mathcal{B},\mathcal{C}}(1^\lambda)\right] - \Pr\left[0 \leftarrow \mathsf{Expt}_1^{\mathsf{MA\text{-}FE},\mathcal{B},\mathsf{sim}_1}(1^\lambda)\right]\right| > \mathsf{negl}(\lambda)$$

The description of $\mathcal{B}^{\mathcal{O}}$ is as follows.

$\underline{\mathcal{B}^{\mathcal{O}}(1^\lambda)}$:

1. $\mathcal{A}$ sends Q, n_1, n_2, s to $\mathcal{C}$.
2. Create a dictionary $\mathbf{R}$ which is initially empty.
3. $\mathcal{C}$ runs $\mathsf{n_1MAFE.crs} \leftarrow \mathcal{O}(Q, n_1, s)$ and $\mathsf{n_2MAFE.crs} \leftarrow \mathsf{sim}_2(1^\lambda, 1^Q, 1^{n_2}, 1^s)$. Then,
   - $\forall\ \mathsf{id} \in [n_1], \mathsf{n_1MAFE.mpk}_{\mathsf{id}} \leftarrow \mathcal{O}(\mathsf{id})$.
   - $\forall\ \mathsf{id} \in [n_1+1, n_1+n_2], \mathsf{n_2MAFE.mpk}_{\mathsf{id}} \leftarrow \mathsf{sim}_2(\mathsf{id} - n_1)$.

   Set $\mathsf{crs} = (\mathsf{n_1MAFE.crs}, \mathsf{n_2MAFE.crs})$ and $\mathsf{MPK}_{\mathsf{id}} = \mathsf{n_1MAFE.mpk}_{\mathsf{id}}$ for each $\mathsf{id} \in [n_1]$ and $\mathsf{MPK}_{\mathsf{id}} = \mathsf{n_2MAFE.mpk}_{\mathsf{id}}$ for each $\mathsf{id} \in [n_1+1, n_1+n_2]$. Send $\mathsf{crs}, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n_1+n_2]}$ to $\mathcal{A}$.
4. Let $\mathcal{A}$ make $Q_1 \le Q$ queries in this phase. $\forall\ q \in [Q_1], \mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathcal{O}(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1+1,\ n_1+n_2], \mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.
5. $\mathcal{A}$ submits challenge circuit C.
   - For each $\mathsf{id} \in [n_1+n_2]$, create $\chi^{(\mathsf{id})} = \{x_{\mathsf{id}_q} : q \in [Q_1], \mathsf{id}_q = \mathsf{id}\}$.
   - For each $X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}$ sample a uniformly random string, $R_X \leftarrow \{0,1\}^L$ and add (X, R_X) to $\mathbf{R}$.

- Create $\mathbf{V}_2 = \{(X, \mathcal{O}(C(\cdot, \ldots, \cdot, X)) : X \in \chi^{(n_1+1)} \times \ldots \times \chi^{(n_1+n_2)}\}$.
- Send $\mathsf{CT} \leftarrow \mathsf{sim}_2(1^{|F|}, \mathbf{V}_2)$ to $\mathcal{A}$.

6. $\forall\ q \in \{Q_1 + 1, \ldots, Q\}$, $\mathcal{A}$ sends $(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - If $\mathsf{id}_q \in [n_1]$, compute $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathcal{O}(\mathsf{id}_q, x_{\mathsf{id}_q})$.
   - Otherwise, if $\mathsf{id}_q \in [n_1 + 1, n_1 + n_2]$, compute $\chi_q^{\mathsf{id}_q} = \{x_{\mathsf{id}_q}\}$ and $V_q^{(2)} = \{(X, \widetilde{F}(X, \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id} \in [n_1]}, R_X, 1^{|C|})) : X \in \chi^{(n_1+1)} \times \ldots \times \chi_q^{\mathsf{id}_q} \times \ldots \times \chi^{(n_1+n_2)}\}$ where R_X is sampled uniformly or reused from $\mathbf{R}$. Sample $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_2(\mathsf{id}_q - n_1, x_{\mathsf{id}_q}, V_q^{(2)})$. Update $\chi^{\mathsf{id}_q} = \chi^{\mathsf{id}_q} \cup \chi_q^{\mathsf{id}_q}$ and $\mathbf{V}_2 = \mathbf{V}_2 \cup V_q^{(2)}$.

   Send $\mathsf{SK}_{\mathsf{id}_q, x_{\mathsf{id}_q}}$ to $\mathcal{A}$.

7. $\mathcal{A}$ outputs its guess b'. Output b'.

As we can see, $\mathcal{B}$ runs in polynomial time in the running time of $\mathcal{A}$, $\mathcal{O}$, and λ. If $\mathcal{O}$ is the honest challenger for $\mathsf{n}_1\mathsf{MAFE}$, the output distribution of $\mathcal{B}$ is exactly the same as Hyb_3. On the other hand, if $\mathcal{O}$ is a simulator for $\mathsf{n}_1\mathsf{MAFE}$, $\mathcal{B}$'s output distribution is same as Hyb_4. As $\mathcal{A}$ can distinguish between Hyb_3 and Hyb_4 with non-negligible probability, $\mathcal{B}$ can also distinguish between the real and ideal worlds for $\mathsf{n}_1\mathsf{MAFE}$. This contradicts the adaptive simulation security of $\mathsf{n}_2\mathsf{MAFE}$. Hence, Hyb_3 and Hyb_4 are indistinguishable. □

G Proofs from Section 10

In this section, we provide the missing proofs from Section 10.

G.1 Hybrid Descriptions

The description of hybrids is as follows.

$\underline{\mathsf{Hyb}_0(1^\lambda):}$ This is $\mathsf{Expt}_0^{\mathsf{parMA\text{-}FE}, \mathcal{A}, \mathcal{C}}$ from Definition 10.1.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum size of the challenge circuit s, the maximum size of the challenge circuit s, the set of non-corrupted users, $\widetilde{\mathbf{S}}$, and the set of unique indices $(j_1^*, \ldots, j_Q^*)$. $\mathcal{C}$ does the following.
   - Sample a hash function $\mathcal{H}$ from $\mathcal{GID} \to 2^{[N]}\big|_D \times 2^{[T]}\big|_v$.
   - For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n]$, $u \in [N]$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u \in [N]})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u \in [N]})$.

   Send $\mathsf{crs} = (\lambda, Q, n, s, \widetilde{\mathbf{S}}, (j_1^*, \ldots, j_Q^*), (\mathsf{1MAFE.crs}_u)_{u \in [N]}, \mathcal{H}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id} \in [n]}$ to $\mathcal{A}$.
2. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q, \mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - Deterministically sample $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \leftarrow \mathcal{H}(\mathsf{GID}_q)$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q, \mathsf{id}_q}\right\}_{u \in [N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q, \mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx} \in [n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID}_q, \mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q, \mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u \in \mathbf{S}_{\mathsf{id}_q}})$.
3. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Compute $\left\{\widehat{C}^u\right\}_{u \in [N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.

- For each $u \in [N]$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
- Sample $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.

Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.

4. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - Deterministically sample $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \leftarrow \mathcal{H}(\mathsf{GID}_q)$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.
5. Let $Q^* \le Q$ be the number of unique $\mathbf{S}_q$'s and Δ_q's. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Abort and output $\perp$ if either of these checks fail.
6. $\mathcal{A}$ outputs bit b'. Output b'.

$\underline{\mathsf{Hyb}_1(1^\lambda)}$: In this hybrid, we sample the output of $\mathcal{H}$ uniformly randomly. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum size of the challenge circuit s, the maximum size of the challenge circuit s, the set of non-corrupted users, $\widetilde{\mathbf{S}}$, and the set of unique indices $(j^*_1,\ldots,j^*_Q)$. $\mathcal{C}$ does the following.
   - Initialize an empty dictionary $\mathbf{H}$ and a counter j' to 0.
   - As the $\mathcal{A}$'s running time is polynomially bounded, there exists $P = P(\lambda)$ such that $\mathcal{A}$ makes at most P queries to $\mathcal{H}$. Sample $\mathbf{S}_1,\ldots,\mathbf{S}_P$ and $\Delta_1,\ldots,\Delta_P$ uniformly randomly. Whenever $\mathcal{A}$ or $\mathcal{C}$ queries $\mathcal{H}$ for GID, if $\mathbf{H}[\mathsf{GID}] = \perp$, set $\mathbf{H}[\mathsf{GID}] = j'$ and return $(\mathbf{S}_{j'}, \Delta_{j'})$.
   - For each $u \in [N]$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n], u \in [N]$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]})$.

   Send $\mathsf{crs} = (\lambda, Q, n, s, \widetilde{\mathbf{S}}, (j^*_1,\ldots,j^*_Q), (\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathcal{H}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.

2-6. Perform these steps same as Hyb_0.

$\underline{\mathsf{Hyb}_2(1^\lambda)}$: In this hybrid, we check if all $\mathbf{S}_q$'s and Δ_q's obey Lemmas 4.2 and 4.3 respectively. The changes are highlighted in red.

1-4. Perform these steps same as Hyb_1.

5. Compute for each $q, q' \in [Q^*], \mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| > t$, abort and output $\perp$. If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$.
6. Let $Q^* \le Q$ be the number of unique $\mathbf{S}_q$'s and Δ_q's. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Abort and output $\perp$ if either of these checks fail.
7. $\mathcal{A}$ output b'. Output b'.

$\mathsf{Hyb}_{3,j}(1^\lambda)$ for $j \in [N+1]$: In this hybrid, we simulate the first $j-1$ instantiations of 1MAFE using $\mathsf{sim}^u_{\mathsf{1MAFE}}$. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum size of the challenge circuit s, the maximum size of the challenge circuit s, the set of non-corrupted users, $\widetilde{\mathbf{S}}$, and the set of unique indices $(j_1^*, \ldots, j_Q^*)$. $\mathcal{C}$ does the following.
   - Initialize an empty dictionary $\mathbf{H}$ and a counter j' to 0.
   - As the $\mathcal{A}$'s running time is polynomially bounded, there exists $P = P(\lambda)$ such that $\mathcal{A}$ makes at most P queries to $\mathcal{H}$. Sample $\mathbf{S}_1, \ldots, \mathbf{S}_P$ and $\Delta_1, \ldots, \Delta_P$ uniformly randomly. Whenever $\mathcal{A}$ or $\mathcal{C}$ queries $\mathcal{H}$ for GID, if $\mathbf{H}[\mathsf{GID}] = \bot$, set $\mathbf{H}[\mathsf{GID}] = j'$ and return $(\mathbf{S}_{j'}, \Delta_{j'})$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
     - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n], u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $(\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
     - Otherwise, $\mathsf{n_1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]})$.

   Send $\mathsf{crs} = (\lambda, Q, n, s, \widetilde{\mathbf{S}}, (j_1^*, \ldots, j_Q^*), (\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathcal{H}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. Initiate $\mathbf{Q}$ initially to empty.
3. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate
     - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
     - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.
4. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ClientEnc}(1^\lambda, 1^Q, 1^n, 1^s, C)$.
   - For each $u \in [N]$, let $F^u(\cdot, \ldots, \cdot) = \mathsf{UserComp}(\cdot, \ldots, \cdot, \widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
     - Otherwise, if $\exists\ q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q}\setminus\{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, \mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, F^u(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.
5. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.

- If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
- Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in[N]} \leftarrow \mathsf{dCSF.ServEnc}(1^\lambda, 1^Q, 1^n, 1^s, \mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
- For each $u \in \mathbf{S}_{\mathsf{id}_q}$,
  - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}$ or $u \geq j$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Otherwise, if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, compute $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}, \widehat{F}^u(x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}))$.
  - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

  Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

6. Compute for each $q, q' \in [Q^*], \mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| > t$, abort and output $\perp$. If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$.
7. Let $Q^* \leq Q$ be the number of unique $\mathbf{S}_q$'s and Δ_q's. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Abort and output $\perp$ if either of these checks fail.
8. $\mathcal{A}$ output b'. Output b'.

$\underline{\mathsf{Hyb}_4(1^\lambda)}$: This is $\mathsf{Expt}_1^{\mathsf{parMA\text{-}FE},\mathcal{A},\mathsf{sim}_{\mathsf{parMA\text{-}FE}}}$ from Definition 10.1. The changes are highlighted in red.

1. $\mathcal{A}$ sends the query bound Q, the number of authorities n, the maximum size of the challenge circuit s, the maximum size of the challenge circuit s, the set of non-corrupted users, $\widetilde{\mathbf{S}}$, and the set of unique indices $(j^*_1, \ldots, j^*_Q)$. $\mathcal{C}$ does the following.
   - Initialize an empty dictionary $\mathbf{H}$ and a counter j' to 0.
   - As the $\mathcal{A}$'s running time is polynomially bounded, there exists $P = P(\lambda)$ such that $\mathcal{A}$ makes at most P queries to $\mathcal{H}$. Sample $\mathbf{S}_1, \ldots, \mathbf{S}_P$ and $\Delta_1, \ldots, \Delta_P$ uniformly randomly. Whenever $\mathcal{A}$ or $\mathcal{C}$ queries $\mathcal{H}$ for GID, if $\mathbf{H}[\mathsf{GID}] = \perp$, set $\mathbf{H}[\mathsf{GID}] = j'$ and return $(\mathbf{S}_{j'}, \Delta_{j'})$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, \mathsf{1MAFE.crs}_u \leftarrow \mathsf{1MAFE.GlobalSetup}(1^\lambda, 1^n)$.
     - Otherwise, $\mathsf{1MAFE.crs}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^\lambda, 1^n)$.
   - For each $\mathsf{id} \in [n], u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}, (\mathsf{1MAFE.mpk}_{\mathsf{id},u}, \mathsf{1MAFE.msk}_{\mathsf{id},u}) \leftarrow \mathsf{1MAFE.AuthSetup}(\mathsf{1MAFE.crs}_u, \mathsf{id})$.
     - Otherwise, $\mathsf{n_1MAFE.mpk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id})$.
   - Set $\mathsf{MPK}_{\mathsf{id}} = (\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{u\in[N]})$ and $\mathsf{MSK}_{\mathsf{id}} = (\{\mathsf{1MAFE.msk}_{\mathsf{id},u}\}_{u\in[N]})$.

   Send $\mathsf{crs} = (\lambda, Q, n, s, \widetilde{\mathbf{S}}, (j^*_1, \ldots, j^*_Q), (\mathsf{1MAFE.crs}_u)_{u\in[N]}, \mathcal{H}), \{\mathsf{MPK}_{\mathsf{id}}\}_{\mathsf{id}\in[n]}$ to $\mathcal{A}$.
2. Initiate $\mathbf{Q}$ initially to empty.
3. Initiate $\mathsf{sim}_{\mathsf{dCSF}}(1^\lambda, 1^Q, 1^n, 1^s, \widetilde{\mathbf{S}})$.
4. $\mathcal{A}$ makes $q \in [Q_1], Q_1 \leq Q$ secret key queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - Add $(\mathsf{GID}_q, \mathsf{id}_q)$ to $\mathbf{Q}$.

- Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\widetilde{\mathbf{S}}} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$.
- For each $u \in \mathbf{S}_{\mathsf{id}_q}$, calculate
  - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
  - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.

Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

5. $\mathcal{A}$ sends the challenge circuit C. $\mathcal{C}$ does the following.
   - Create $\mathbf{V} = \{(\mathsf{GID}, X, C(X)) : \mathsf{GID} \in \Psi_n(\mathbf{Q}), X = (x_{\mathsf{GID},1}, \ldots, x_{\mathsf{GID},n})\}$.
   - Compute $\left\{\widehat{C}^u\right\}_{u\in\mathbf{S}_{\mathsf{corr}}}, \{\widehat{y}^u_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}(C, \{j^*_{\mathsf{GID}}\}_{\mathsf{GID}\in\Psi_n(\mathbf{Q})})$.
   - For each $u \in \widetilde{\mathbf{S}}$, let $F^u(\cdot,\ldots,\cdot) = \mathsf{UserComp}(\cdot,\ldots,\cdot,\widehat{C}^u)$.
   - For each $u \in [N]$,
     - If $u \in \mathbf{S}_{\mathsf{corr}}$, $\mathsf{ct}_u \leftarrow \mathsf{1MAFE.Enc}(\{\mathsf{1MAFE.mpk}_{\mathsf{id},u}\}_{\mathsf{id}\in[n]}, F^u)$.
     - Otherwise, if $\exists\ q \in [Q], u \in \mathbf{S}_q$, and $\psi_{\mathbf{Q}\setminus\{(\mathsf{GID}_q,\mathsf{id}_q)\},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \widehat{y}^u_{\mathsf{GID}_q})$.
     - Otherwise, $\mathsf{ct}_u \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(1^{|F^u|}, \emptyset)$.

   Send $\mathsf{CT} = (\mathsf{ct}_u)_{u\in[N]}$ to $\mathcal{A}$.

6. $\mathcal{A}$ makes at most $Q - Q_1$ queries of the form $(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q})$. $\mathcal{C}$ does the following.
   - If $(\mathsf{GID}_q, \cdot) \in \mathbf{G}$, set $(\mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q})$ such that $(\mathsf{GID}_q, \mathbf{S}_{\mathsf{id}_q}, \Delta_{\mathsf{id}_q}) \in \mathbf{G}$. Otherwise, add $(\mathsf{GID}_q, \mathbf{S}_g, \Delta_g)$ to $\mathbf{G}$ and increment the counter g.
   - If $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1, V_q = (\mathsf{GID}_q, j^*_{\mathsf{GID}_q}, X = (x_{\mathsf{GID}_q,1}, \ldots, x_{\mathsf{GID}_q,n}), F^u(X))$.
   - Compute $\left\{\widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}\right\}_{u\in\mathbf{S}_{\mathsf{id}_q}}, \{\widehat{y}^u_{\mathsf{GID}_q}\}_{u\in\mathbf{S}_{\mathsf{id}_q}} \leftarrow \mathsf{sim}_{\mathsf{dCSF}}.(\mathsf{GID}_q, \mathsf{id}_q, x_{\mathsf{GID}_q,\mathsf{id}_q}, \Delta_{\mathsf{id}_q}, V_q)$.
   - For each $u \in \mathbf{S}_{\mathsf{id}_q}$,
     - If $u \in \mathbf{S}_{\mathsf{id}_q} \cap \mathbf{S}_{\mathsf{corr}}$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{1MAFE.KeyGen}(\mathsf{id}_q, \mathsf{1MAFE.msk}_{\mathsf{id}_q,u}, \{\mathsf{1MAFE.mpk}_{\mathsf{idx},u}\}_{\mathsf{idx}\in[n]}, \mathsf{GID}, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q})$.
     - Otherwise, if $\psi_{\mathbf{Q},n}(\mathsf{GID}_q, \mathsf{id}_q) = 1$, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}, \widehat{y}^u_{\mathsf{GID}_q})$.
     - Otherwise, $\mathsf{1MAFE.sk}_{\mathsf{id},u} \leftarrow \mathsf{sim}^u_{\mathsf{1MAFE}}(\mathsf{id}_q, \widehat{x}^u_{\mathsf{GID}_q,\mathsf{id}_q}))$.

   Send $\mathsf{SK}_{\mathsf{GID}_q,\mathsf{id}_q,x_{\mathsf{GID}_q,\mathsf{id}_q}} = (\mathbf{S}_{\mathsf{id}_q}, \{\mathsf{1MAFE.sk}_{\mathsf{id},u}\}_{u\in\mathbf{S}_{\mathsf{id}_q}})$.

7. Compute for each $q, q' \in [Q^*], \mathbf{S}_{\mathsf{corr}} = \bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}$. If $|\mathbf{S}_{\mathsf{corr}}| > t$, abort and output $\perp$. If for any $q \in [Q^*], \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right) = \emptyset$, abort and output $\perp$.

8. Let $Q^* \leq Q$ be the number of unique $\mathbf{S}_q$'s and Δ_q's. Check if for any $q, q' \in [Q^*], \widetilde{\mathbf{S}} = [N] \setminus \left(\bigcup_{q\neq q'} \mathbf{S}_q \cap \mathbf{S}_{q'}\right)$ and $j^*_q \in \Delta_q \setminus \left(\bigcup_{q\neq q'} \Delta_{q'}\right)$. Abort and output $\perp$ if either of these checks fail.

9. $\mathcal{A}$ output b'. Output b'.