Dynamic Collusion Bounded Functional Encryption from Identity-Based Encryption

Rachit Garg
UT Austin*

Rishab Goyal
MIT†

George Lu
UT Austin‡

Brent Waters
UT Austin and
NTT Research§

Abstract

Functional Encryption is a powerful notion of encryption in which each decryption key is associated with a function f such that decryption recovers the function evaluation $f(m)$. Informally, security states that a user with access to function keys $\mathsf{sk}_{f_1}, \mathsf{sk}_{f_2}, \ldots$ (and so on) can only learn $f_1(m), f_2(m), \ldots$ (and so on) but nothing more about the message. The system is said to be q-bounded collusion resistant if the security holds as long as an adversary gets access to at most $q = q(\lambda)$ function keys. A major drawback of such *statically* bounded collusion systems is that the collusion bound q must be declared at setup time and is fixed for the entire lifetime of the system.

We initiate the study of *dynamically* bounded collusion resistant functional encryption systems which provide more flexibility in terms of selecting the collusion bound, while reaping the benefits of statically bounded collusion FE systems (such as quantum resistance, simulation security, and general assumptions). Briefly, the virtues of a dynamically bounded scheme can be summarized as:

Fine-grained individualized selection. It lets each encryptor select the collusion bound by weighing the trade-off between performance overhead and the amount of collusion resilience.

Evolving encryption strategies. Since the system is no longer tied to a single collusion bound, thus it allows to dynamically adjust the desired collusion resilience based on any number of evolving factors such as the age of the system, or a number of active users, etc.

Ease and simplicity of updatability. None of the system parameters have to be updated when adjusting the collusion bound. That is, the same key sk_f can be used to decrypt ciphertexts for collusion bound $q = 2$ as well as $q = 2^\lambda$.

We construct such a dynamically bounded functional encryption scheme for the class of all polynomial-size circuits under the general assumption of Identity-Based Encryption.

*Email: rachg96@cs.utexas.edu.

†Email: goyal@utexas.edu. Research supported in part by NSF CNS Award #1718161, an IBM-MIT grant, and by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR00112020023. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

‡Email: gclu@cs.utexas.edu.

§Email: bwaters@cs.utexas.edu. Supported by NSF CNS-1908611, CNS-1414082, Packard Foundation Fellowship, and Simons Investigator Award.

1 Introduction

Public-key encryption [DH76] is one of the most fundamental concepts in cryptography. Traditionally, public-key encryption was defined to provide an "all-or-nothing" type functionality and security, where given a decryption key sk, a user can either recover the entire plaintext m from a ciphertext ct or nothing at all. In the recent years, an extremely powerful notion of encryption called Functional Encryption (FE) [SW05, BSW11] has emerged.

FE provides a fine-grained access control mechanism over encrypted data where a decryption key is now associated with a function f and the decryptor recovers the function evaluation $f(m)$ from the ciphertext. Moreover, a user with access to function keys $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_n}$ can only learn $f_1(m), \ldots, f_n(m)$ but nothing more about the message. This security requirement is commonly captured in a game based indistinguishability definition, where the adversary submits two messages, m_0 and m_1, as a challenge and must be unable to distinguish between encryptions of m_0 and m_1 with non-negligible probability given that $f_i(m_0) = f_i(m_1)$ hold for all keys in adversary's possession.

Over the last several years, FE has been studied extensively. Significant progress has been made towards building various expressive forms of FE under such indistinguishability-based definitions. Starting with initial works [BW07, KSW08] that built specific forms of predicate encryption over bilinear maps, the search for FE for general circuits under standard cryptographic assumptions culminated in the recent breakthrough work of Jain, Lin, and Sahai [JLS21]. They proposed an FE scheme for general circuits from a combination of PRGs in $\mathbf{NC}^0$, Symmetric eXternal Diffie-Hellman (SXDH), Learning with Errors (LWE), and Learning Parity with Noise (LPN) over large fields assumptions. While this is tremendous progress, an unfortunate limitation of this FE scheme is that it is susceptible to quantum attacks due to the post-quantum insecurity of the SXDH assumption. But even more broadly, pursuing the direction of indistinguishability-based security for FE suffers from the drawback that it is unclear how it captures the intuition that an attacker learns *at most the function evaluation but nothing more*.

For these reasons, FE has also been investigated in the bounded collusion model under simulation-based definitions. In the bounded collusion model, the FE system declares a bound q at the setup time, such that all the system parameters are allowed to grow polynomially with q (in addition to the security parameter λ). Additionally, the security requirement is captured via a simulation-based game, which says that as long as the attacker does not make more than q key queries, the adversary's view - which includes the ciphertext ct_m and function keys $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_q}$ - can be "simulated" given only the function evaluations $f_1(m), \ldots, f_q(m)$ and nothing more about m. Although this more closely captures the intuition behind FE, if the attacker corrupts more than q keys, then no security is provided. Despite its limitations, the bounded collusion model for FE has been very useful in various contexts such as proving negative results in differential privacy [KMUW18], applications to tracing [GKW18, CVW+18], etc. In some cases, it is the only currently known pathway to certain applications in the post-quantum regime. A notable feature of the bounded collusion model is that under them, FE can be built from the minimal assumption of public-key encryption (and OWFs in case of private-key FE) as studied in a long line of works [SS10, GVW12, AR17, Agr17, GKW18, CVW+18, AV19].

The question. A major drawback of such bounded collusion FE systems is that the setup authority needs to declare the collusion bound q at the very beginning, and the bound q is fixed, once and for all, for the entire lifetime of the system. This puts the authority in a difficult situation, as it

requires an incredible amount of foresight at the setup time. In particular, if the authority sets the bound q lower than the eventual number of compromised keys, then the system will be insecure; whereas overestimating the bound q would result in significant performance overhead. Now when the collusion bound is breached, the only option to would to do a fresh setup and redistribute the keys which is at best inefficient, and possibly infeasible in certain scenarios. Switching to the state-of-the-art fully collusion resistant FE schemes would suffer from drawbacks discussed above.

With the aforementioned limitations of existing FE systems, we ask the following –

> *Can we build an FE system for general circuits that reaps the benefits of bounded collusion FE systems – post-quantum security, simulation security, and general assumptions – while at the same time provide more flexibility to the authority in terms of selecting the collusion bound? And, would such an FE system lead to results in the domain of full collusion resistance?*

In this work, we study the above question. We answer the first part in affirmative by introducing a new flexible corruption model that we call the "dynamic collusion" model, and building a simulation secure FE system in the dynamic collusion model from the general assumption of Identity-Based Encryption (IBE) [Sha85, Coc01, BF01] (for which we have quantum-safe instantiations [GPV08, CHKP10, ABB10]). Since it is widely believed that the FE for general circuits is significantly more expressive than plain IBE, this seems to answer the latter part negatively. Next, we define our dynamic collusion model and provide a high level overview of our techniques.

Defining Dynamically Bounded Collusion Resistance

In this work, we refer to the traditional notion of bounded collusion resistance for FE as *statically bounded collusion resistance*. Recall that, syntactically, a statically bounded FE is defined exactly as fully collusion resistant FE, that is using four polynomial time algorithms – Setup, KeyGen, Enc, and Dec – except the Setup algorithm now additionally takes the target collusion bound q as an input. As mentioned previously, declaring the collusion bound q upfront lets the setup authority set up the system parameters with enough redundancy, and this typically leads to the running time and sizes of all system parameters (i.e., the keys and ciphertexts) to grow polynomially with q.

In the dynamic collusion model, the Setup algorithm no longer takes the collusion bound as input, but instead the Enc algorithm selects the collusion bound per ciphertext. That is, the setup and key generation algorithms no longer depend on the collusion bound q, but only the encryptor needs to specify the collusion bound.[1] *Basically, this lets the encryptor dynamically decide the size of set of colluding users against which it wants to hide its message.* As a consequence, in the dynamic collusion model, only the size of the ciphertexts potentially grows with the collusion bound q, but the running times of the Setup and KeyGen algorithms (therefore the public and secret keys) are independent of q. The security requirement is again captured via a simulation-based game but where the admissibility constraints on the attacker are lifted such that the number of key queries the attacker is permitted can be adaptively specified at the time of committing the challenge m instead of beginning of the game as in the static model.

Our dynamic collusion model and its comparison with the static model is discussed in detail in Section 3. Below we briefly highlight the virtues of the dynamic collusion model.

[1] However, note that it is essential that the master public-secret keys and every function key is resuable for all values of the collusion bound.

Fine-grained individualized selection. A dynamically bounded collusion FE scheme allows each user to select the collusion bound by weighing the trade-off between the performance overhead and amount of collusion resilience at encryption time. For example, depending upon the factors such as available computing resources, or the bandwidth on the communication channel, or the sensitivity of data etc, an encryptor might want to increase/decrease the amount of collusion resilience to better fit the computing/communication/privacy constraints.

Evolving encryption strategies. Since the system is no longer statically tied to a single collusion bound at setup time, thus it allows to dynamically adjust the desired collusion resilience based on any number of evolving factors such as the age of the system, or number of active users etc. Thus, the authority does not need to have any foresight about the attackers at setup time in contrast to statically bounded collusion FE systems.

(Of course the ciphertexts in which the collusion bound was exceeded will not secure, but future attacks can be prevented by adapting to a larger collusion bound.)

Ease and simplicity of updatability. While the above features are already highly desirable, a noteworthy property of these systems is that none of the parameters have to be updated when adjusting the collusion bound. That is, the same function key sk_f can be used to decrypt ciphertexts for collusion bound $q = 2$ as well as $q = 2^\lambda$ without requiring any updates. Also, the storage space for the parameters is bounded by a fixed polynomial in λ.

Next, we provide an overview of our approach and describe the technical ideas. Later on, we discuss some related works and open questions.

1.1 Technical Overview

In this section, we provide a high level overview of our new collusion framework and the corresponding FE construction. The overview is split into five parts which roughly correspond to the proceeding sections of the paper. First, we informally introduce the notion of dynamically bounded collusion resistant FE. Second, we define an efficiency property that we refer to as *weak optimality* for statically bounded collusion FE systems, and show that any weakly optimal FE construction could be generically lifted to a dynamically bounded collusion FE scheme. Next, we build such a weakly optimal FE scheme via the framework of tagged functional encryption scheme, where a tagged FE scheme is same as a regular FE scheme except each ciphertext and secret key is additionally embedded with a tag such that only ciphertexts and keys with the sam tag can be combined together. Finally, we build a tagged FE scheme for statically bounded collusions in two steps – first, reduce the problem of constructing tagged FE with static collusions to the simpler setting of at most one key corruption (also referred to as 1-bounded collusion); and second, design a tagged FE system in the simpler setting directly from IBE.

Dynamic vs. Static Bounded Collusion Model

Let us start by recalling the syntax of functional encryption in the static collusion model. An FE scheme in the static collusion model consists of four algorithms with the following semantics:

– Setup takes as input the collusion bound q and samples the master public-secret key pair $(\mathsf{mpk}, \mathsf{msk})$.

– KeyGen generates a function key sk_f given function f and master key msk.

– Enc encrypts a message m to a ciphertext ct.

– Dec recovers $f(m)$ from the ciphertext and decryption key.

In the dynamic collusion model, the collusion bound q is not fixed at the system setup, but instead the encryptor chooses the amount of collusion resilience it wants every time a fresh ciphertext is created. This is reflected with the following changes:

– Setup no longer takes the collusion bound q as an input.

– Enc takes the desired collusion bound q as an additional input for sampling the ciphertext.

Note that since the collusion bound q is not specified during setup or key generation at all, thus the efficiency condition for a dynamically bounded collusion FE scheme requires the running time of Setup and KeyGen to be fixed polynomials in λ, whereas in static setting they are allowed to grow polynomially with q.

Static to Dynamic via Weak Optimality

As we mentioned before, our first observation is that a dynamically bounded collusion FE scheme can be constructed from any statically bounded scheme if it satisfies a 'weak optimality' property. Intuitively, the weak optimality property says that the running time of the setup and key generation algorithms grows only poly-logarithmically in the collusion bound q.

Now looking closely at the notion of weakly-optimal statically bounded collusion FE, we observe that the major difference between this and a dynamic system is that the Setup algorithm requires q as an explicit input in the static setting, but not in the dynamic setting. Our idea to get around this is to exploit the efficiency property of the static scheme, where the dynamic collusion FE scheme essentially runs λ independent instances of the static collusion FE scheme in parallel with geometrically increasing collusion bounds. That is, i-th subsystem (running a single instance of the static scheme) is set up with collusion bound $q_i = 2^i$. And, now the master public-secret key pair as well as each function key in the dynamic system contains λ independently sampled keys where the i-th sub-key is sampled using the i-th static FE system. Since the encryption algorithm receives the target collusion bound q as input, thus the encryptor uniquely selects a static FE sub-system under which it encrypts the message. The target collusion bound to subsystem index mapping can simply be defined $i := \lceil \log q \rceil$ (i.e., nearest power of two). Note that setting up the system this way ensures the dynamic system achieves the desired efficiency. This is because the setup and key generation will be efficient (by weak optimality of the static FE scheme), and since $2^i = 2^{\lceil \log q \rceil} < 2q$, thus the running time of encryption and decryption is a polynomial in q.

Since the above transformation is very natural, one would expect the simulation security of the resulting dynamic FE system to also follow directly from the simulation security of the underlying static FE schemes. However, this is not the case. To better understand the technical barrier, let us first consider the most natural simulation strategy described next. The simulator for the dynamic system simply runs the simulator for each of the underlying static systems in parallel, where the ciphertext simulator is only run for the static system corresponding to the adversarially selected challenge target collusion bound q^*. While this seems to compile, there are two subtle issues that need to be carefully handled.

First, the running time of each static FE simulator grows with the underlying collusion bound which grows as large as exponential in λ. For avoiding the problem of inefficient simulation, we

additionally require the underlying static FE scheme to have weakly-optimal simulators as well which means that all but the ciphertext simulation phase of the static FE could be performed optimally (i.e., the simulator running time grows only poly-logarithmically in q). However, this is still not enough for proving simulation security. The reason is that typically the simulation security states that the distribution of secret keys and ciphertext are simulatable as long as the adversary does not make more key queries than what is specified by the static collusion bound. That is, if the adversary makes more key queries then no guarantee is provided. Now our dynamic FE simulator must invoke the underlying static FE simulator even for collusion bounds smaller than q^*, thus the standard simulation guarantee is insufficient. To get around this issue, we define a notion called *strong* simulation security for static-bounded-collusion FE schemes under which we require that the real and ideal worlds are also indistinguishable even when the adversary makes more key queries than that specified by the collusion bound as long as the adversary does not make any challenge message queries. More details are provided in Section 3.2.

From Tagged FE to Weak Optimality

Next, our next idea is to embed auxiliary tagging information inside each individual ciphertext and decryption key such that the auxiliary information is useful for achieving weak optimality generically by embedding information about the collusion bound inside the auxiliary information. Formally, in a tagged FE system, the semantics of encryption and key generation are changed as:

– $\mathsf{KeyGen}, \mathsf{Enc}$, both also take in a tag string tg as an input.

And, now the decryption algorithm recovers $f(m)$ from the ciphertext and decryption key corresponding to tags $\mathsf{tg}_1, \mathsf{tg}_2$ (respectively) iff $\mathsf{tg}_1 = \mathsf{tg}_2$. Basically, the intuition behind a tagged FE scheme is to efficiently implement many parallel instances of a statically bounded collusion FE scheme such that the master public-secret keys do not grow with number of underlying (untagged) FE instances.

In other words, the idea behind tagged FE is to serve as an extension to regular (untagged) FE in the same way as IBE is to PKE, that is to capture the same master public-secret key compression properties. That is, a tagged FE enables compressing exponentially many parallel instances of untagged FE into a succinct system where all the system parameters are efficient, and the ciphertexts and decryption keys corresponding to each underlying untagged FE system can be efficiently computed given those parameters. In terms of simulation security for tagged FE, the property is a natural extension of statically bounded-collusion security model for FE to the tagged setting, where now the adversary is allowed to query keys and ciphertexts for an unbounded number of tags, and the simulation security must hold for all challenge ciphertexts (queried under separate tags) as long as the number of key queries does not exceed the collusion bound on any tag for which a challenge ciphertext is also requested.

Looking ahead, the benefit of a tagged FE scheme will be that we can distribute the final desired collusion bound over to the auxiliary tag space as well, and not just the collusion bound ingrained in the tagged FE system. And, since tagged FE can encode the tag space more efficiently than the collusion bound, thus this is useful for obtaining the desired weak optimality.

At a high level, to transform any tagged FE scheme into an FE scheme that satisfies the desired weak optimality property, we rely on the linearization trick by Ananth and Vaikuntanathan [AV19] where they suggested a generic compiler to improve efficiency of a statically bounded-collusion FE

scheme from an arbitrary polynomial dependence on the collusion bound, q, to only a linear dependence. Our observation is that if we substitute all the underlying FE scheme in the linearization transformation from [AV19] with a single tagged FE scheme, then that would result in a statically bounded-collusion FE scheme with weak optimality.

Briefly, collusion bound linearization transformation simply consists of running q many parallel instances of the inefficient (untagged) FE scheme each, but with collusion bound set to be the security parameter λ. While for encrypting the message m, the ciphertext is computed as an encryption of m under each of the underlying FE schemes; the key generator only generated a decryption key for a random instance out of the q inefficient FE systems. By a standard balls and bins concentration argument, it was shown that, with all but negligible probability, the collusion bound of λ was never crossed for any of the underlying FE system as long as only q many total key queries were made. We rely on the same idea for our weak optimality transformation wherein we simply replace the q many parallled untagged FE systems with a single tagged FE system, where now the i-th FE sub-scheme in the [AV19] transformation is set to be the sub-scheme corresponding to tag value i. The full transformation is provided later in Section 5.

Intuitively, we use the linearization trick to absorb the blow-up due to the collusion bound in the tag space of the FE scheme instead. This decouples the desired collusion bound, q, from the resulting FE scheme with the collusion bound fixed inside the underlying tagged FE system thereby allowing us to set the collusion bound for the tagged system to be simply λ. Thus, we can reason from the efficiency of the tagged FE scheme that the resulting scheme only has polylogarithmic dependence in the λ for Setup and KeyGen, making it weakly optimal.

Amplifying Collusion Bound in Tagged FE

The next component in our sequence of transformations is a generic collusion bound amplification procedure for tagged FE, in turn reducing the problem to constructing tagged FE for 1-bounded collusion instead. Our approach follows the general bootstrapping blueprint developed for upgrading collusion bound in untagged FE literature [GVW12, AV19] which runs a specific multiparty computation protocol in the head.

In such MPC protocols, there are N servers/parties and a single client with an input x with the computation proceeding in two phases – offline and online. In the offline phase, the client encodes its input x into N individual encodings – $\{\hat{x}^1, \ldots, \hat{x}^{\mathsf{N}}\}$ – one for each server. While in the online phase, a function f is encoded into N individual encodings – $\{\hat{f}^1, \ldots, \hat{f}^{\mathsf{N}}\}$ – such that i-th server learns the i-th function encoding, and any subset S of servers of size p can locally decode their individual encodings to obtain partial evaluations, $\hat{y}^i$ for $i \in S$, such that all these p partial evaluations can be publicly combined to compute the function evaluation $f(x)$. And importantly, the security of the MPC protocol assures that, even if at most t servers get corrupted, no information other than actual value $f(x)$ can be adversarially learned given the public partial evaluations. Now for applications to collusion bound amplification in FE, it is important to have MPC protocols in which the client can delegate the computation for multiple functions w.r.t. a single offline phase.

The high level idea is that each ciphertext encodes the message m into various different pieces where each piece corresponds to an individual offline encoding for a particular server, and now the key generator selects a random subset of servers for which it gives the appropriate function encodings for each selected server. In more detail, the bootstrapping procedure works as follows, where $\mathsf{1KeyFE}$ is any 1-bounded collusion untagged FE scheme:

– Setup samples N independent master public-secret key pair $(\mathsf{mpk}_i, \mathsf{msk}_i)$ for the $\mathsf{1KeyFE}$ scheme. These N key pairs are set as the master public-secret key pairs for this scheme respectively.

– Enc encodes the message m using the offline phase to compute encodings $\{\hat{x}^1, \ldots, \hat{x}^{\mathsf{N}}\}$, and encrypts the i-th encoding under the i-th master public key, that is $\mathsf{ct}_i \leftarrow \mathsf{1KeyFE.Enc}(\mathsf{mpk}_i, \hat{x}^i)$ for $i \in [\mathsf{N}]$, and outputs $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{\mathsf{N}})$ as the full ciphertext.

– KeyGen selects a random subset $S \subseteq [\mathsf{N}]$ of size p, and performs the online phase to compute $\{\hat{f}^1, \ldots, \hat{f}^{\mathsf{N}}\}$. Now enable decryption, it creates a FE decryption key for each server $i \in S$ enabling the local circuit computation, that is $\mathsf{sk}_{f,i} \leftarrow \mathsf{1KeyFE.KeyGen}(\mathsf{msk}_i, \mathsf{Local}(\hat{f}^i, \cdot))$, and sets the final decryption key as these individual decryption keys $\mathsf{sk}_{f,i}$ for $i \in S$.

– Dec first recovers the partial evaluations $\hat{y}^i = \mathsf{Local}(\hat{f}^i, \hat{x}^i)$ for $i \in S$ by running the $\mathsf{1KeyFE}$ decryption, and then combines them to compute $f(m)$.

It turns out that the above compiler amplifies the collusion bound from 1 to q if a simple combinatorial property, regarding the random sets $(S_1, \ldots, S_q)$ sampled for each key, is satisfied. Here $S_j \subseteq [\mathsf{N}]$ be the set sampled while anwering the j-th key query. Observe that whenever two sets $S_j, S_{j'}$ intersect at an index i, we learn two keys for the underlying $\mathsf{1KeyFE}$ scheme thereby breaking its security, and an adversary can completely learn the underlying encoding $\hat{x}^i$. And, if the security is broken for enough $\mathsf{1KeyFE}$ systems (i.e., $> \mathsf{t}$), then our MPC guarantee fails. Thus, to prove security it is sufficient to show that the total number of pairwise intersections is not larger than t. With this combinatorial guarantee, we can rely on the security of $\mathsf{1KeyFE}$ and the MPC protocol to ensure no information other than $f(m)$ is revealed.

Our observation here is that the same blueprint can also be used for tagged FE schemes where for amplifying 1-bounded collusion to q-bounded collusion, we start with a slightly larger tag space for the underlying 1-bounded tagged FE scheme. Basically, to build a q-bounded collusion tagged FE scheme with tag space $\mathcal{T}$, we start with a 1-bounded scheme with tag space $[\mathsf{N}] \times \mathcal{T}$, and replace i-th instantiation of the $\mathsf{1KeyFE}$ scheme with 1-bounded tagged FE scheme and the tag is set as (i, tg) where tg is the tag to be embedded (during encryption and key generation, respectively). Now the correctness and security of the resulting compiler closely follows the analysis for untagged FE schemes from [AV19], with some subtleties in the analysis that arise due to the fact that in tagged FE simulation security we need to be able to jointly simulate multiple ciphertexts (though for distinct tags) at that the same time. More details follow later in Section 6.

Adding Tags to 1-Bounded-Collusion FE via IBE

Lastly, to instantiate our above transformations to build a dynamically bounded collusion FE scheme, we need a tagged FE scheme that achieves 1-bounded collusion simulation security. To that end, we look back at the 1-bounded collusion untagged FE construction by Sahai and Seyalioglu [SS10] which works by combining garbled circuits with plain public-key encryption. In a few words, our idea is to imitate the same ideology for instantiating our tagged FE scheme, but replace the plain public-key encryption scheme with an identity-based encryption scheme to introduce additional space for efficiently embedding tags in the identity space of the IBE scheme.

Recall that in the well-known 1-bounded collusion untagged FE construction, an encryptor garbles the universal circuit U with message m hardwired such that, on an input a description of a circuit C, the hardwired circuit computes $C(m)$. Now the encryptor hides the wire keys for

the garbled circuit under the corresponding PKE public keys chosen during setup time, where two PKE key pairs are sampled per bit of the description length of the circuit C. And, the decryption key for a circuit C simply corresponds to half of the PKE secret keys selected on the basis of bit description of C, that $C[i]$-th PKE secret key for each i. Basically, a decryptor first uncovers the wire labels corresponding to circuit C using PKE decryption, and then simply evaluates the garbled circuit to learn the circuit evaluation $C(m)$.

We observe that the same construction can be upgraded to a tagged FE scheme if we simply replace each PKE system in the above transformation with an IBE system[2], where the identity space of the IBE system will be used to encode the "designated tag". Thus, the encryptor simply sets the IBE identity corresponding to which encryption is performed to be the input tag $\mathtt{tg}$, and the decryption key consists of appropriate IBE keys where the identity for each underlying IBE system is the tag $\mathtt{tg}$ to be embedded. Clearly, this gives the desired efficiency if the underlying IBE scheme is efficient, and the security follows by a similar argument as to before where a more careful analysis is needed to argue simulation security in presence of multiple tags. While the above transformation is sufficient to prove security in the non-adaptive setting as the original [SS10] construction, we rely on the delayed/non-committing encryption strategies [CFGN96] as in [GVW12, GSW21] to upgrade to adaptive security. Our tagged FE scheme with 1-bounded collusion security is described in Section 7.

1.2 Related Work and Future Directions

Prior work on bounded collusion resistance. The intial works on bounded collusion resistance for FE were for the specific class of IBE systems. Dodis et al. [DKXY02] and Goldwasser, Lewko, and Wilson [GLW12] constructed bounded collusion secure IBE with varying parametere size from regular public-key encryption and special types of linearly key homomorphic public-key encryption, respectively. For more expressive classes of FE, Sahai and Seyalioglu [SS10] proposed general functional encryption schemes resilient against a single function-key query using garbled circuits [Yao86]. Following [SS10], GVW [GVW12] build a statically bounded collusion resistant FE scheme for $\mathbf{NC}^1$ circuits from any public-key encryption scheme, and also provided a generic compiler to improve to the class of all polynomial time computable functions by additionally relying on PRFs computable in $\mathbf{NC}^1$. Afterwards, a number of follow-up works [AR17, Agr17, GKW18, CVW+18] improved the concrete efficiency of the statically bounded collusion resistant FE scheme wherein they improved the dependence of the FE scheme parameters on the collusion bound q by relying on more structured algebraic assumptions. Most recently, Ananth and Vaikuntanathan [AV19] achieved optimally efficient statically secure FE scheme from the minimal assumption of public-key encryption. The optimal efficiency states that the system parameters grow only linearly with the collusion bound q, since any further improvement would lead to a fully collusion resistant FE scheme via the bootstrapping theorems from [GGH+13, SW14, AJ15, BV15, AJS15].

Comparison with bundling functionalities and encrypt ahead FE. Goyal, Koppula, and Waters (GKW) [GKW16] proposed the concept of bundling functionalities in FE systems, where bundling functionalities in an FE scheme meant having the property that a single set of public

[2]Technically, we compress the keys even further as we replace all the PKE key pairs with a single IBE key pair instead of a sequence of IBE key pairs. However, for the purpose of this overview, we present this simpler version.

parameters can support the union of all message/function spaces supported by the underlying FE system. They provided a generic transformation that started with IBE (and other implied primitives) and was able to upgrade any FE scheme to its bundled counterpart. One might ask that whether applying the [GKW16] transformation to the family of bounded collusion FE, where the the collusion bound q is treated as part of the functionality index that is bundled, already leads to a dynamically bounded collusion FE system. It turns out this is not the case because such a generic transformation suffers from the limitation that a function key for a given collusion bound is not reusable for other collusion bounds. In particular, this necessitates each user to make additional queries to the authority for obtaining function keys for desired collusion bound, and this only solves the problem of removing the problem of removing the collusion bound dependence for the setup algorithm. Additionally, GKW proposed a novel variant of FE called encrypt ahead FE. One could ask the same question about relationship between encrypt ahead FE and dynamically bounded collusion resistant FE, and the answer is the same as for the case of bundling functionalities which is they are insufficient.

Open questions. Our work introduces a new interesting avenue for exploring dynamic collusion resilience in FE systems. It is a interesting question whether public-key encryption is sufficient for building *dynamically* bounded collusion resistant FE systems, or whether IBE is necessary.[3] Another interesting research direction is studying similar concepts of dynamic "query" resilience in other cryptographic contexts. For example, one could ask the same question for the concept of CCA-secure encryption where we know that CPA-secure public-key encryption implies (statically-)bounded-query-CCA security for public-key encryption [CHH+07]. We believe answering the question of dynamically bounded-query-CCA security might provide more insight in resolving the longstanding open problem of constructing a (general) CCA-secure encryption scheme from a CPA-secure one.

1.3 Concurrent Work

In a concurrent and independent work, Agrawal et al. [AMVY21] also achieved similar results as this work with some differences in the presentation and abstractions. They introduced the concept of dynamic collusion bounded functional encryption as we do to provide more flexibility for selecting the collusion bound. And, similar to us, they construct such functional encryption schemes for all polynomial-sized circuits from identity-based encryption. In addition, they also extend their results to uniform computation models while relying on specific algebraic assumptions.

2 Preliminaries

Notations. Let PPT denote probabilistic polynomial-time. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q. We denote the set of all positive integers upto n as $[n] := \{1, \ldots, n\}$. For any finite set S, $x \leftarrow S$ denotes a uniformly random element x from the set S. Similarly, for any distribution $\mathcal{D}$, $x \leftarrow \mathcal{D}$ denotes an element x drawn from distribution $\mathcal{D}$. The distribution $\mathcal{D}^n$ is used to represent a distribution over vectors of n components, where each

[3] Agrawal et al. [AMVY21] showed that IBE is necessary for dynamically bounded collusion resistant FE, thereby answering our question.

component is drawn independently from the distribution $\mathcal{D}$. Two distributions $\mathcal{D}_1$ and $\mathcal{D}_2$, parameterized by security parameter λ, are said to be computationally indistinguishable, represented by $\mathcal{D}_1 \approx_c \mathcal{D}_2$, if for all PPT adversaries $\mathcal{A}$, $|\Pr[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{D}_1] - \Pr[\mathcal{A}(x) = 1 : x \leftarrow \mathcal{D}_2]| \leq \mathsf{negl}(\lambda)$.

2.1 Garbled Circuits

Our definition of garbled circuits [Yao86] is based upon the work of Bellare et al. [BHR12]. Let $\{\mathcal{C}_n\}_n$ be a family of circuits where each circuit in $\mathcal{C}_n$ takes n bit inputs. A garbling scheme GC for circuit family $\{\mathcal{C}_n\}_n$ consists of polynomial-time algorithms Garble and Eval with the following syntax.

- $\mathsf{Garble}(1^\lambda, C \in \mathcal{C}_n)$: The garbling algorithm takes as input the security parameter λ and a circuit $C \in \mathcal{C}_n$. It outputs a garbled circuit $\widetilde{C}$, together with $2n$ wire keys $\{w_{i,b}\}_{i \leq n, b \in \{0,1\}}$.
- $\mathsf{Eval}(\widetilde{C}, \{w_i\}_{i \leq n})$: The evaluation algorithm takes as input a garbled circuit $\widetilde{C}$ and n wire keys $\{w_i\}_{i \leq n}$ and outputs $y \in \{0,1\}$.

Correctness. A garbling scheme GC for circuit family $\{\mathcal{C}_n\}_n$ is said to be correct if for all λ, n, $x \in \{0,1\}^n$ and $C \in \mathcal{C}_n$, $\mathsf{Eval}(\widetilde{C}, \{w_{i,x_i}\}_{i \leq n}) = C(x)$, where $(\widetilde{C}, \{w_{i,b}\}_{i \leq n, b \in \{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, C)$.

Security. Informally, a garbling scheme is said to be secure if for every circuit C and input x, the garbled circuit $\widetilde{C}$ together with input wires $\{w_{i,x_i}\}_{i \leq n}$ corresponding to some input x reveals only the output of the circuit $C(x)$, and nothing else about the circuit C or input x.

Definition 2.1. A garbling scheme $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$ for a class of circuits $\mathcal{C} = \{\mathcal{C}_n\}_n$ is said to be a secure garbling scheme if there exists a polynomial-time simulator Sim such that for all n, $C \in \mathcal{C}_n$ and $x \in \{0,1\}^n$, the following distributions are computationally indistinguishable:

$$\left\{\mathsf{Sim}\left(1^\lambda, 1^n, 1^{|C|}, C(x)\right)\right\}_\lambda \approx_c \left\{\left(\widetilde{C}, \{w_{i,x_i}\}_{i \leq n}\right) \ : \ \left(\widetilde{C}, \{w_{i,b}\}_{i \leq n, b \in \{0,1\}}\right) \leftarrow \mathsf{Garble}(1^\lambda, C)\right\}_\lambda.$$

While this definition is not as general as the definition in [BHR12], it suffices for our construction.

2.2 Identity-Based Encryption

An Identity-Based Encryption (IBE) scheme IBE for set of identity spaces $\mathcal{I} = \{\mathcal{I}_n\}_{n \in \mathbb{N}}$ and message spaces $\mathcal{M}$ consists of four polynomial time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with the following syntax:

$\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and identity space index n. It outputs the public parameters mpk and the master secret key msk.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}) \to \mathsf{sk}_{\mathsf{id}}$. The key generation algorithm takes as input the master secret key msk and an identity $\mathsf{id} \in \mathcal{I}_n$. It outputs a secret key $\mathsf{sk}_{\mathsf{id}}$.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m) \to \mathsf{ct}$. The encryption algorithm takes as input the public parameters mpk, a message $m \in \mathcal{M}$, and an identity $\mathsf{id} \in \mathcal{I}_n$. It outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) \to m/\perp$. The decryption algorithm takes as input a secret key $\mathsf{sk}_{\mathsf{id}}$ and a ciphertext ct. It outputs either a message $m \in \mathcal{M}$ or a special symbol $\perp$.

Correctness. We say an IBE scheme $\mathsf{IBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ satisfies correctness if for all $\lambda, n \in \mathbb{N}$, $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, $\mathsf{id} \in \mathcal{I}_n$, $m \in \mathcal{M}$, $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$, and $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$, we have that $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}) = m$.

Definition 2.2. We say an IBE scheme $\mathsf{IBE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is secure if for any stateful PPT adversary $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda, n \in \mathbb{N}$, the following holds

$$\Pr\left[\mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{st}, \mathsf{ct}) = b : \begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \quad b \leftarrow \{0,1\} \\ (m_0, m_1, \mathsf{id}^*) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(1^\lambda, 1^n, \mathsf{mpk}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}^*, m_b)\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where all identities id queried by $\mathcal{A}$ satisfy $\mathsf{id} \neq \mathsf{id}^*$.

3 Functional Encryption: Dynamic Bounded Collusion

In this section, we define the notion of functional encryption (FE) where we start by recalling the regime of (statically) bounded collusion secure FE systems as studied in prior works [SS10, GVW12]. We follow that by extending the notion to *dynamic* collusion bounded secure FE systems. And, along the way we also introduce a special compactness property for statically bounded collusion secure FE schemes. This will serve as an appropriate intermediate abstraction to build a fully dynamic collusion bounded FE schemes.

Syntax. Let $\mathcal{M} = \{\mathcal{M}_n\}_{n\in\mathbb{N}}$, $\mathcal{R} = \{\mathcal{R}_n\}_{n\in\mathbb{N}}$ be families of sets, and $\mathbb{F} = \{\mathcal{F}_n\}$ a family of functions, where for all $n \in \mathbb{N}$ and $f \in \mathcal{F}_n$, $f : \mathcal{M}_n \to \mathcal{R}_n$. We will also assume that for all $n \in \mathbb{N}$, the set $\mathcal{F}_n$ contains an *empty function* $\epsilon_n : \mathcal{M}_n \to \mathcal{R}_n$. As in [BSW11], the empty function is used to capture information that intentionally leaks from the ciphertext.

A functional encryption scheme FE for a family of function classes $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$ and message spaces $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ consists of four polynomial-time algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ with the following semantics.

$\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and the functionality index n[4] (in unary), and outputs the master public-secret key pair $(\mathsf{mpk}, \mathsf{msk})$.

$\mathsf{Enc}(\mathsf{mpk}, m \in \mathcal{M}_n) \to \mathsf{ct}$. The encryption algorithm takes as input the master public key mpk and a message $m \in \mathcal{M}_n$ and outputs a ciphertext ct.

$\mathsf{KeyGen}(\mathsf{msk}, f \in \mathcal{F}_n) \to \mathsf{sk}_f$. The key generation algorithm takes as input the master secret key msk and a function $f \in \mathcal{F}_n$ and outputs a function key sk_f.

$\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) \to \mathcal{R}_n$. The decryption algorithm takes as input a ciphertext ct and a secret key sk_f and outputs a value $y \in \mathcal{R}_n$.

[4] One coud additionally consider the setup algorithm to take as input a sequence of functionality indices where the function class and message space are characterized by all such indices (e.g., having input length and circuit depth as functionality indices). For ease of notation, we keep a single functionality index in the above definition.

Correctness and Efficiency. A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be correct if for all $\lambda, n \in \mathbb{N}$, functions $f \in \mathcal{F}_n$, messages $m \in \mathcal{M}_n$ and $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, we have that

$$\Pr\left[\mathsf{Dec}(\mathsf{KeyGen}(\mathsf{msk}, f), \mathsf{Enc}(\mathsf{mpk}, m)) = f(m)\right] = 1,$$

where the probability is taken over the coins of key generation and encryption algorithms. And, it is said to be efficient if the running time of the algorithms is a fixed polynomial in the parameters λ and n.

3.1 Bounded Collusion FE: Static and Dynamic

Informally, a functional encryption scheme is said to be secure if an adversary having secret keys for functions $\{f_i\}_{i \le q}$ and a ciphertext ct for message m learns only $\{f_i(m)\}_{i \le q}$, $\epsilon(m)$ and nothing else about the underlying message m. Here ϵ is the empty function associated with the message space.

The Static Setting. Now in the "static" bounded collusion setting, the scheme is said to guarantee security so long as q is a polynomial in the security parameter λ and *fixed a-priori at the setup time*. Thus, the syntax of the setup algorithm changes as follows:

$\mathsf{Setup}(1^\lambda, 1^n, q) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and the functionality index n (in unary), and also takes as input the 'collusion bound' q (in binary).[5] It outputs the master public-secret key pair $(\mathsf{mpk}, \mathsf{msk})$.

Efficiency. Although the collusion bound q is given in binary to the setup algorithm, the efficiency condition for a statically bounded collusion FE scheme only requires that the running time of the all the algorithms is a fixed polynomial in λ, n and q. That is, the running time of Setup, KeyGen, Enc, and Dec is allowed to polynomially grow with the collusion bound q.

Static bounded collusion security. This is formally captured via the following 'simulation based' security definition as follows. We first provide the adaptive definition, and later provide the non-adaptive definition.

Definition 3.1 (static-bounded-collusion simulation-security)**.** A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be statically-bounded-collusion simulation-secure if there exists a stateful PPT simulator $\mathsf{Sim} = (\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3)$ such that for every stateful PPT adversary

[5] Although most prior works on bounded collusion security consider the collusion bound q to either be a global parameter, or given in unary to the setup algorithm. Here we instead pass it in binary for technical reasons as will become clear in the sequel. See Remark 3.1 for more details.

$\mathcal{A}$, the following distributions are computationally indistinguishable:

$$\left\{ \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) : \begin{array}{c} (1^n, 1^q) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, q) \\ m \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, m) \end{array} \right\}_{\lambda \in \mathbb{N}}$$

$$\approx_c$$

$$\left\{ \mathcal{A}^{\mathsf{S}_3^{U_m(\cdot)}(\mathsf{st}_2,\cdot)}(\mathsf{ct}) : \begin{array}{c} (1^n, 1^q) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{mpk}, \mathsf{st}_0) \leftarrow \mathsf{S}_0(1^\lambda, 1^n, q) \\ m \leftarrow \mathcal{A}^{\mathsf{S}_1(\mathsf{st}_0,\cdot)}(\mathsf{mpk}) \\ (\mathsf{ct}, \mathsf{st}_2) \leftarrow \mathsf{S}_2(\mathsf{st}_1, \Pi^m) \end{array} \right\}_{\lambda \in \mathbb{N}}$$

whenever the following admissibility constraints and properties are satisfied:

- S_1 and S_3 are stateful in that after each invocation, they updates their states st_1 and st_3 (respectively) which is carried over to its next invocation.
- Π^m contains a list of functions f_i queried by $\mathcal{A}$ in the pre-challenge phase along with the their output on the challenge message m. That is, if f_i is the i-th function queried by $\mathcal{A}$ to oracle S_1 and q_{pre} be the number of queries $\mathcal{A}$ makes before outputting m, then $\Pi^m = \big((f_1, f_1(m)), \ldots, (f_{q_{\mathsf{pre}}}, f_{q_{\mathsf{pre}}}(m))\big)$.[6]
- $\mathcal{A}$ makes at most q queries combined to the key generation oracles in the corresponding games.
- S_3 for each queried function f_i, in the post-challenge phase, makes a single query to its message oracle U_m on the same f_i itself.

Remark 3.1 (unary vs binary)**.** Note that in the above security games, we require the adversary to specify the collusion bound q in unary at the beginning. This is in contrast to the setup algorithm which gets q in binary as an input. The reason for this distinction is that in the security game for bounded collusion security we do not want to allow the attacker to specify super-polynomial collusion bounds, whereas (as we point out later) allowing the setup algorithm to be run on super-polynomial values of the collusion bound is important for our dynamic collusion bounded FE schemes.

Weak optimality. Additionally, we also introduce the notion of a "weakly optimal" statically-bounded-collusion secure FE scheme where this system provides better efficiency properties. That is, in a weakly optimal static bounded collusion system, the running time of the setup and key generation algorithms grows only poly-logarithmically in the collusion bound q. Concretely, we define it below.

Definition 3.2 (weakly optimal statically-bounded-collusion)**.** A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be 'weakly optimal' statically-bounded-collusion FE scheme if the running time of the Setup and KeyGen algorithm is additionally upper bounded by a fixed polynomial in λ, n and $\log q$.

[6] To be more precise, Π^m should also contain the empty function and the evaluation of empty function on challenge message $(\epsilon_n, \epsilon_n(m))$. However, for ease of notation, throughout the paper we assume that to be implicitly added to the list of function-value pairs.

Strengthening the simulation guarantee. In this work, we consider a strengthening of the above simulation-secure properties (for the class of weakly optimal static-bounded-collusion FE schemes) which will be be crucial towards building a dynamic-bounded-collusion functional encryption scheme. Note that typically the simulation security states that the distribution of secret keys and ciphertext are simulatable as long as the adversary does not make more key queries than what is specified by the static collusion bound. That is, if the adversary makes more key queries then no guarantee is provided. However, we consider a stronger simulation guarantee below wherein the real world is still simulatable even when the adversary makes more key queries than that specified by the collusion bound as long as the adversary does not make any challenge message queries. That is, either the collusion bound is not crossed, or no challenge ciphertext is queried. In addition to this, we require the running time of the simulator algorithms $\mathsf{S}_0, \mathsf{S}_1$ and S_3 (that is, all except the ciphertext simulator S_2) grow only poly-logarithmically in the static collusion bound q. Formally, we define it below.

Definition 3.3 (strong simulation-security)**.** A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be statically-bounded-collusion *strong* simulation-secure if, in the security game defined in Definition 3.1, the following additional conditions hold:

1. the number of key queries made by adversary is allowed to exceed the static collusion bound q as long as the adversary does not submit any challenge message, and
2. the running time of the simulator algorithms $\mathsf{S}_0, \mathsf{S}_1$ and S_3 is upper bounded by a fixed polynomial in λ, n and $\log q$.

Lastly, we also define the non-adaptive variant of the simulation security.

Definition 3.4 (non-adaptive simulation-security)**.** A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be statically-bounded-collusion *non-adaptive* (regular/strong) simulation-secure if the adversary is prohibited from making any key queries in the post-challenge phase (that is, after receiving the challenge ciphertext) in its respective security game.

The Dynamic Setting. Now in the "dynamic" bounded collusion setting, the scheme is no longer tied to a single collusion bound q fixed a-priori at the system setup, but instead the encryptor could choose the amount of collusion resilience it wants. Thus, this changes the syntax of the setup and encryption algorithm when compared to the static setting from above:

$\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ and the functionality index n (in unary). It outputs the master public-secret key pair $(\mathsf{mpk}, \mathsf{msk})$.

(Note that thus syntactically the setup of a dynamic bounded collusion scheme is same as that of a fully collusion resistant scheme.)

$\mathsf{Enc}(\mathsf{mpk}, m \in \mathcal{M}_n, 1^q) \to \mathsf{ct}$. The encryption algorithm takes as input the master public key mpk, a message $m \in \mathcal{M}_n$, and it takes the desired collusion bound q (in unary) as an input. It outputs a ciphertext ct.

Efficiency. Since the collusion bound q is not specified during setup or key generation at all, thus the efficiency condition for a dynamically bounded collusion FE scheme requires the running time of Setup and KeyGen to be fixed polynomials in λ and n. While since the encryptor takes q as input

in unary, thus the running time of the Enc algorithm could grow polynomially with collusion bound q. Similarly, the running time of Dec is also allowed to grow polynomially with collusion bound q.

Dynamic bounded collusion security. This is formally captured via a 'simulation based' security definition as in the static setting. The game is similar to that provided in Definition 3.1, except now the attacker specifies the collusion bound q while making the challenge ciphertext query and the simulator also only receives the collusion bound as input at that point. For completeness, we describe it formally below (both the adaptive and non-adaptive variants).

Definition 3.5 (dynamic-bounded-collusion simulation-security)**.** A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be dynamically-bounded-collusion simulation-secure if there exists a stateful PPT simulator $\mathsf{Sim} = (\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3)$ such that for every stateful PPT adversary $\mathcal{A}$, the following distributions are computationally indistinguishable:

$$\left\{ \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) : \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n) \\ (m, 1^q) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, m, 1^q) \end{array} \right\}_{\lambda \in \mathbb{N}}$$

$$\approx_c$$

$$\left\{ \mathcal{A}^{\mathsf{S}_3^{U_m(\cdot)}(\cdot)}(\mathsf{ct}) : \begin{array}{c} 1^n \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{mpk} \leftarrow \mathsf{S}_0(1^\lambda, 1^n) \\ (m, 1^q) \leftarrow \mathcal{A}^{\mathsf{S}_1(\cdot)}(\mathsf{mpk}) \\ \mathsf{ct} \leftarrow \mathsf{S}_2(\Pi^m, 1^q) \end{array} \right\}_{\lambda \in \mathbb{N}}$$

whenever the admissibility constraints and properties, as defined in Definition 3.1, are satisfied.

Definition 3.6 (non-adaptive simulation-security)**.** A functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be dynamically-bounded-collusion *non-adaptive* simulation-secure if, in the security game defined in Definition 3.5, the adversary is prohibited from making any key queries in the post-challenge phase (that is, after receiving the challenge ciphertext).

3.2 Upgrading Static to Dynamic Bounded Collusion FE via Weak Optimal Efficiency

In this section, we provide a generic construction of a dynamic-bounded-collusion FE scheme from any static-bounded-collusion FE scheme that satisfies the strong simulation property (Definition 3.3) and the weak optimality property (Definition 3.2). Below we provide our construction followed by correctness and security proofs.

3.2.1 Construction

Let $\mathsf{Static\text{-}FE} = (\mathsf{S\text{-}FE.Setup}, \mathsf{S\text{-}FE.Enc}, \mathsf{S\text{-}FE.KeyGen}, \mathsf{S\text{-}FE.Dec})$ be a weakly-optimal static-bounded-collusion FE scheme for a family of function classes $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$ and message spaces $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$. We use $\mathsf{Static\text{-}FE}$ to build a dynamic-bounded-collusion FE scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ as follows.

$\mathsf{Setup}(1^\lambda, 1^n) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm runs the Static-FE setup algorithm λ times with increasing values of the static collusion bound q as follows:

$$\forall i \in [\lambda], \quad (\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{S\text{-}FE.Setup}(1^\lambda, 1^n, q = 2^i).$$

It then sets the master secret and public keys as an λ-tuple of all these keys, i.e. $\mathsf{msk} = (\mathsf{msk}_i)_{i\in[\lambda]}$ and $\mathsf{mpk} = (\mathsf{mpk}_i)_{i\in[\lambda]}$.

$\mathsf{KeyGen}(\mathsf{msk}, f) \to \mathsf{sk}_f$. Let $\mathsf{msk} = (\mathsf{msk}_i)_{i\in[\lambda]}$. The key generation algorithm runs the Static-FE key generation algorithm with all λ keys independently as $\mathsf{sk}_{i,f} \leftarrow \mathsf{S\text{-}FE.KeyGen}(\mathsf{msk}_i, f)$ for $i \in [\lambda]$. It outputs the secret key sk as $\mathsf{sk} = (\mathsf{sk}_{i,f})_{i\in[\lambda]}$.

$\mathsf{Enc}(\mathsf{mpk}, m, 1^Q) \to \mathsf{ct}$. Let $\mathsf{mpk} = (\mathsf{mpk}_i)_{i\in[\lambda]}$. The encryption algorithm simply encrypts the message m under $\lceil \log Q \rceil$-th master public key as $\mathsf{ct} \leftarrow \mathsf{S\text{-}FE.Enc}(\mathsf{mpk}_{\lceil \log Q \rceil}, m)$. (It also includes Q as part of the ciphertext.)

$\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) \to z$. Let $\mathsf{sk}_f = (\mathsf{sk}_{i,f})_{i\in[\lambda]}$. The decryption algorithm runs the Static-FE decryption using the $\lceil \log Q \rceil$-th function key as $z \leftarrow \mathsf{S\text{-}FE.Dec}(\mathsf{sk}_{\lceil \log Q \rceil, f}, \mathsf{ct})$.

3.2.2 Correctness, Efficiency, and Security

The correctness of the above scheme follows directly from the correctness of the underlying static-bounded-collusion FE system, while for the desired efficiency consider the following arguments. First, note that by weak optimality of Static-FE we have that the running time of S-FE.Setup and S-FE.KeyGen grows as $\mathsf{poly}(\lambda, n, \log q)$. Since the Setup and S-FE.KeyGen algorithms run S-FE.Setup and S-FE.KeyGen (respectively) λ many times for $\log q \in \{1, \ldots, \lambda\}$, thus we get that running time of Setup and KeyGen is $\mathsf{poly}(\lambda, n)$ as desired. Lastly, the encryption and decryption algorithm run in time at most $\mathsf{poly}(\lambda, n, 2^{\lceil \log Q \rceil}) = \mathsf{poly}(\lambda, n, Q)$ since the $\lceil \log Q \rceil$-th static-bounded-collusion FE system uses $2^{\lceil \log Q \rceil} \leq 2 \cdot Q$ as the static collusion bound. Thus, the resulting FE scheme satisfies the required efficiency properties.
To conclude, we prove the following.

Theorem 3.1. If Static-FE = (S-FE.Setup, S-FE.Enc, S-FE.KeyGen, S-FE.Dec) is a weakly-optimal static-bounded-collusion simulation-secure FE scheme (as per Definitions 3.2 and 3.3), then the above scheme FE = (Setup, Enc, KeyGen, Dec) is a dynamic-bounded-collusion simulation-secure FE scheme (as per Definition 3.5).

The proof follows from a composition of the static-bounded-collusion simulation-security property of Static-FE. Recall that in the static setting, we require the scheme to provide a stronger form of real world vs. ideal world indistinguishability. Where typically the simulation security states that the distribution of secret keys and ciphertext are simulatable as long as the adversary does not make more key queries than what is specified by the static collusion bound. That is, if the adversary makes more key queries then no guarantee is provided. However, in our formalization of simulation security for static-bounded-collusion FE schemes, we require that the real and ideal worlds are also indistinguishable even when the adversary makes more key queries than that specified by the collusion bound as long as the adversary does not make any challenge message queries. That is, either the collusion bound is not crossed, or no challenge ciphertext is queried. Also, the

running time of the simulator algorithms $\mathsf{S}_0, \mathsf{S}_1$ and S_3 (all except the ciphertext simulator S_2) grow only poly-logarithmically in the collusion bound.

Thus, the simulator for the dynamic-bounded-collusion FE scheme simply runs the S_0 algorithms for all collusion bounds $q = 1, \ldots, 2^\lambda$ to simulate the individual master public keys. It then also runs the S_1 algorithms for simulating the individual function keys for each of these static-bounded-collusion FE systems for answering each adversarial key query. Note that since the running time of S_0 and S_1 is also $\mathsf{poly}(\lambda, n, \log q)$ where $q = 1, \ldots, 2^\lambda$, thus this is efficient.

Now when the adversary makes the challenge query for message m, it also specifies the target collusion bound Q^*. The dynamic-bounded-collusion simulator then runs only the ciphertext simulator algorithm S_2 for the static FE system corresponding to collusion bound $\log q = \lceil \log Q^* \rceil$. Note that the simulator does not run S_2 for the underlying FE schemes with lower (and even higher) collusion bounds. This is important for two reasons: (1) we want to invoke the simulation security of the i-th static FE scheme for $i < \lceil \log Q^* \rceil$ but we can only do this if the ciphertext simulator S_2 is not run for these static FE schemes, (2) the running time of S_2 could grow polynomially with the collusion bound q, thus we should not invoke simulator algorithm S_2 for $i > \lceil \log Q^* \rceil$ as well (since for say $i = \lambda$, the running time would be exponential in λ which would make the dynamic simulator inefficient). Thus, even the ciphertext simulation is efficient and the dynamic simulator is an admissible adversary with respect to static FE challenger, therefore our dynamic FE simulator is both efficient and can rely on simulation security of the underlying static FE schemes. The last phase of simulation (i.e., post-challenge key generation phase) works the same as the second phase simulator (i.e., pre-challenge key generation phase) which is by running S_3 for all collusion bounds. This completes a high level sketch. A complete proof is provided later in Appendix B.

Remark 3.2 (non-adaptive simulation-security)**.** If the underlying static-bounded-collusion FE scheme only provides security against non-adaptive attackers (as per Definition 3.4), then the resulting dynamic-bounded-collusion FE scheme is also secure only against non-adaptive attackers (as per Definition 3.6).

4 Tagged Functional Encryption

In this work, we introduce the concept of tagged functional encryption where the basic difference when compared to regular functional encryption systems is that ciphertexts and secret keys are embedded with a tag value such that only the ciphertexts and keys with the same tags can be combined during decryption.

Formally, a tagged FE scheme in the static collusion model for a set of tag spaces $\mathcal{I} = \{\mathcal{I}_z\}_{z \in \mathbb{N}}$ consists of the same four algorithms with following modification to the syntax:

$\mathsf{Setup}(1^\lambda, 1^n, 1^z, 1^q) \to (\mathsf{mpk}, \mathsf{msk})$. In addition to the normal inputs taken by a static-bounded FE scheme, the setup also takes in a tag space index z, which fixes a tag space $\mathcal{I}_z$.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{tg} \in \mathcal{I}_z, m \in \mathcal{M}_n) \to \mathsf{ct}$. The encryption also takes in a tag $\mathsf{tg} \in \mathcal{I}_z$ to bind to the ciphertext.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{tg} \in \mathcal{I}_z, f \in \mathcal{F}_n) \to \mathsf{sk}_{\mathsf{tg},f}$. The key generation also binds the secret keys to a fixed tag $\mathsf{tg} \in \mathcal{I}_z$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{tg},f}, \mathsf{ct}) \to \mathcal{R}_n$. The decryption algorithm has syntax identical to a non-tagged scheme.

Correctness and Efficiency. A tagged FE scheme tgfe is said to be correct if for all $\lambda, n, z, q \in \mathbb{N}$, every function $f \in \mathcal{F}_n$, message $m \in \mathcal{M}_n$, tag $\mathsf{tg} \in \mathcal{I}_z$, and $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^z, 1^q)$, we have that

$$\Pr\left[\mathsf{Dec}(\mathsf{KeyGen}(\mathsf{msk}, \mathsf{tg}, f), \mathsf{Enc}(\mathsf{mpk}, \mathsf{tg}, m)) = f(m)\right] = 1,$$

where the probability is taken over the coins of key generation and encryption algorithms. And, it is said to be efficient if the running time of the algorithms is a fixed polynomial in the parameters λ, n, q and z.

Security. The security definition is modelled in a similar fashion to the ordinary static bounded collusion FE game with the difference that the adversary plays it on multiple tg simultaneously and the simulator must simulate the ciphertexts for every tag. In addition, the adversary is also allowed to make arbitrary many secret key queries for all other tags. The formal definition follows.

Definition 4.1 (tagged-static-bounded-collusion simulation-security)**.** For any choice of parameters $\lambda, n, q, z \in \mathbb{N}$, consider the following list of stateful oracles $\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2$ where these oracles simulate the FE setup, key generation, and encryption algorithms respectively, and all three algorithms share and update the same global state of the simulator. Here the attacker interacts with the execution environment $\mathcal{E}$, and the environment makes queries to the simulator oracles. Formally, the simulator oracles and the environment are defined below:

$\mathsf{S}_0(1^\lambda, 1^n, 1^z, 1^q)$ generates the simulated master public key mpk of the system, and initializes the global state st of the simulator which is used by the next two oracles.

$\mathsf{S}_1(\cdot, \cdot, \cdot)$, upon a call to generate secret key on a function-tag-value tuple $(f_i, \mathsf{tg}_i, \mu_i)$, where the function value is either $\mu_i = \perp$ (signalling that the adversary has not yet made any encryption query on tag tg_i), or $(m^{\mathsf{tg}_i}, \mathsf{tg}_i)$ has already been queried for encryption (for some message m^{tg_i}), and $\mu_i = f_i(m^{\mathsf{tg}_i})$, the oracle outputs a simulated key $\mathsf{sk}_{f_i, \mathsf{tg}_i}$.

$\mathsf{S}_2(\cdot, \cdot)$, upon a call to generate ciphertext on a tag-list tuple $(\mathsf{tg}_i, \Pi^{m^{\mathsf{tg}_i}})$, where the list $\Pi^{m^{\mathsf{tg}_i}}$ is a possibly empty list of the form $\Pi^{m^{\mathsf{tg}_i}} = \left((f_1^{\mathsf{tg}_i}, f_1^{\mathsf{tg}_i}(m^{\mathsf{tg}_i})), \ldots, (f_{q_{\mathsf{pre}}}^{\mathsf{tg}_i}, f_{q_{\mathsf{pre}}}^{\mathsf{tg}_i}(m^{\mathsf{tg}_i}))\right)$ (that is, contains the list of function-value pairs for which the adversary has already received a secret key for), the oracle outputs a simulated ciphertext $\mathsf{ct}_{\mathsf{tg}_i}$.

$\mathcal{E}^{\mathsf{S}_1, \mathsf{S}_2}(\cdot, \cdot)$, receives two types of queries – secret key query and encryption query. Upon a secret key query on a function-tag pair (f_i, tg_i), if $(m^{\mathsf{tg}_i}, \mathsf{tg}_i)$ has already been queried for encryption (for some message m^{tg_i}) then $\mathcal{E}$ queries key oracle S_1 on tuple $(f_i, \mathsf{tg}_i, \mu_i = f_i(m^{\mathsf{tg}_i}))$, otherwise it adds (f_i, tg_i) to the its local state, and queries S_1 on tuple $(f_i, \mathsf{tg}_i, \mu_i = \perp)$. And, it simply forwards oracle's simulated key $\mathsf{sk}_{f_i, \mathsf{tg}_i}$ to the adversary.

Upon a ciphertext query on a message-tag pair (m_i, tg_i), if the adversary made an encryption query on the same tag tg_i previously, then the query is disallowed (that is, at most one message query per every unique tag is permitted). Otherwise, it computes a (possibly empty) list of function-value pairs of the form $\Pi^{m_i} = \left((f_1^{\mathsf{tg}_i}, f_1^{\mathsf{tg}_i}(m^{\mathsf{tg}_i})), \ldots, (f_{q_{\mathsf{pre}}}^{\mathsf{tg}_i}, f_{q_{\mathsf{pre}}}^{\mathsf{tg}_i}(m^{\mathsf{tg}_i}))\right)$ where $(f_j^{\mathsf{tg}_i}, \mathsf{tg}_i)$ are stored in $\mathcal{E}$'s local state, and removes all such pairs $(f_j^{\mathsf{tg}_i}, \mathsf{tg}_i)$ from its local state. $\mathcal{E}$ then queries ciphertext oracle S_2 on tuple $(\mathsf{tg}_i, \Pi^{m_i})$, and simply forwards oracle's simulated ciphertext $\mathsf{ct}_{\mathsf{tg}_i}$ to the adversary.

A tagged functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be tagged-statically-bounded-collusion simulation-secure if there exists a stateful PPT simulator $\mathsf{Sim} = (\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2)$ such that for every stateful *admissible* PPT adversary $\mathcal{A}$, the following distributions are computationally indistinguishable:

$$\left\{ \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot),\mathsf{Enc}(\mathsf{mpk},\cdot,\cdot)}(\mathsf{mpk}) : \begin{array}{c} (1^n, 1^q, 1^z) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^z, 1^q) \end{array} \right\}_{\lambda \in \mathbb{N}}$$

$$\approx_c$$

$$\left\{ \mathcal{A}^{\mathcal{E}^{\mathsf{S}_1,\mathsf{S}_2}(\cdot,\cdot)}(\mathsf{mpk}) : \begin{array}{c} (1^n, 1^q, 1^z) \leftarrow \mathcal{A}(1^\lambda) \\ \mathsf{mpk} \leftarrow \mathsf{S}_0(1^\lambda, 1^n, 1^z, 1^q) \end{array} \right\}_{\lambda \in \mathbb{N}}$$

where $\mathcal{A}$ is an admissible adversary if:

- $\mathcal{A}$ makes at most one encryption query per unique tag (that is, if the adversary made an encryption query on some tag tg_i previously, then making another encryption query for the same tag is disallowed)
- $\mathcal{A}$ makes at most q queries combined to the key generation oracles in the above experiments for all tags tg_i such that it also submitted an encryption query for tag tg_i.

Definition 4.2 (tagged-static-bounded-collusion non-adaptive simulation-security)**.** A tagged functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is said to be tagged-statically-bounded-collusion *non-adaptive* simulation-secure if, in the security game defined in Definition 4.1, the adversary is prohibited from making any key queries on any particular tag in the post-challenge phase (that is, if the adversary makes an encryption query w.r.t. tag tg, then it must not make any more key queries on the same tag tg but can make key queries for other tags).

5 Tagged to Weakly Optimal Static Collusion FE

In this section we show how to convert the our construction of Q-bounded tagged FE to a weakly optimal statically secure functional encryption scheme with collusion bound Q. The transformation is very similar to the transformation in [AV19] that achieves linear complexity for any bounded-key FE scheme.

Let Q be the desired collusion bound for the static scheme. The transformation in [AV19] starts with Q instances of q-bounded statically secure FE, where q is set to some polynomial in the security parameter. The setup parameters are thus linearly bounded in Q. Encryption simply calls the base encrypt algorithm (for the q-bounded collusion scheme) on each instance. Since the base encryption scheme is for collusion bound $q = \mathsf{poly}(\lambda)$, one instance of encrypt takes time polynomial in the security parameter and thus encrypt is linearly bounded in Q. Key generation for a circuit C simply selects one of the instances at random and outputs the key generated by the base scheme on this instance. Correctness holds as the encrytor has encrypted for all possible instances. Security fails only if, after giving out Q secret keys, the load on a particular instance exceeds $q = \mathsf{poly}(\lambda)$ (which happens with only negligible probability via a simple Chernoff argument).

The transformation has the drawback that the setup outputs Q instances and thus all the algorithms depend linearly on Q. Our observation is thar if we start with a tagged FE scheme instead, then we can compress the public and secret parameters using the tag space by setting it

proportional to the desired collusion bound Q. Similarly the key generation algorithm takes in a single master public key and master secret key and outputs one instance of the secret key. This helps us satisfy the weakly optimal property. More formal details follow below.

5.1 Construction

Let $\mathsf{tgfe} = (\mathsf{TgFE.Setup}, \mathsf{TgFE.Enc}, \mathsf{TgFE.KeyGen}, \mathsf{TgFE.Dec})$ be a bounded-collusion tagged FE scheme for a family of circuit classes $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$, message spaces $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$, and tag space $\{\mathcal{I}_z = \{0,1\}^z\}_{z\in\mathbb{N}}$. We use TgFE to build $\mathsf{Static\text{-}FE}$ a *weakly-optimal* static-bounded-collusion FE scheme for the same function classes and message spaces.

$\mathsf{Setup}(1^\lambda, 1^n, Q) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm runs the TgFE setup algorithm with the tag space of $\mathcal{I}_z = [2^{\lceil \log Q \rceil}]$ and collusion bound $q = \lambda$ and sets the master public-secret keys as

$$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{TgFE.Setup}(1^\lambda, 1^n, 1^z = 1^{\lceil \log Q \rceil}, 1^q = 1^\lambda).$$

Notation. Here and throughout the paper, we represent $\lceil \log Q \rceil$-bit tags as elements over a larger alphabet $[2^{\lceil \log Q \rceil}]$, and when we write $u \leftarrow [Q]$ then that denotes sampling u as a random integer between 1 and Q which can be uniquely encoded as an $\lceil \log Q \rceil$-bit tag.

$\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{sk}_C$. It samples a tag $u \leftarrow [Q]$, key $\mathsf{sk}_{C,u} \leftarrow \mathsf{TgFE.KeyGen}(\mathsf{msk}, u, C)$, and outputs $\mathsf{sk}_C = (\mathsf{sk}_{C,u}, u)$.

$\mathsf{Enc}(\mathsf{mpk}, m, 1^Q) \to \mathsf{ct}$. It encrypts the message m for all possible tags, and outputs the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$ where each sub-ciphertext is computed as:

$$\forall u \in [Q], \quad \mathsf{ct}_u \leftarrow \mathsf{TgFE.Enc}(\mathsf{mpk}, u, m).$$

$\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) \to y$. Let $\mathsf{sk}_C = (\mathsf{sk}_{C,u}, u)$ and $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$. The algorithm outputs $y \leftarrow \mathsf{TgFE.Dec}(\mathsf{sk}_{C,u}, \mathsf{ct}_u)$.

5.2 Correctness, Efficiency, and Security

The correctness of the above scheme follows directly from the correctness of the underlying TgFE scheme. For the efficiency, recall the requirements (Definition 3.2) which state that the Setup and KeyGen algorithms should be bound by a polynomial in λ, n and $\log Q$. Both Setup and KeyGen run $\mathsf{TgFE.Setup}$ and $\mathsf{TgFE.KeyGen}$ once respectively. From the efficiency of these algorithms, we know that the running time is $\mathsf{poly}(\lambda, n, \lceil \log Q \rceil)$.

Formally, we prove the following.

Theorem 5.1. If $\mathsf{tgfe} = (\mathsf{TgFE.Setup}, \mathsf{TgFE.Enc}, \mathsf{TgFE.KeyGen}, \mathsf{TgFE.Dec})$ is a tagged-statically-bounded-collusion simulation-secure FE scheme (as per Definition 4.1), then the above scheme is a weakly-optimal static-bounded-collusion *strong* simulation-secure FE scheme (as per Definition 3.3).

Proof. Let $\mathsf{TgFE.Sim} = (\mathsf{TgFE.S}_0, \mathsf{TgFE.S}_1, \mathsf{TgFE.S}_2)$ be the simulators satisfying Definition 4.1. Let these simulators share a global state $\mathsf{TgFE.st}$. We will construct simulators $\mathsf{Sim} = (\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3)$ that satisfy Definition 3.3 as follows.

$\mathsf{S}_0(1^\lambda, 1^n, Q)$

Sample $\mathsf{mpk} \leftarrow \mathsf{TgFE.S}_0(1^\lambda, 1^n, 1^{\lceil \log Q \rceil}, 1^\lambda)$. Let $\mathsf{st}_0 \leftarrow \mathsf{TgFE.st}$.

$\mathsf{S}_1(\mathsf{st}_0, C)$

Sample $u \leftarrow [Q]$. It simulates secret key as $\mathsf{sk}_{C,u} \leftarrow \mathsf{TgFE.S}_1((f, \mathsf{tg}, \mu) = (C, u, \perp))$ and keeps $\mathsf{TgFE.st}$ updated. It also adds (C, u) as the key query to its state st_1 and ensures $\mathsf{TgFE.st}$ is included in st_1. It outputs $(\mathsf{sk}_{C,u}, u)$.

$\mathsf{S}_2(\mathsf{st}_1, \Pi^m)$

1. Let $|\Pi^m| = Q_{\mathsf{pre}}$ and $\Pi^m = ((C_1, C_1(m)), \ldots, (C_{Q_{\mathsf{pre}}}, C_{Q_{\mathsf{pre}}}(m)))$. For all $u \in [Q]$, let $\mathsf{qset}_u = 0$ and $\mathsf{uset}_u = 0$. For every $u \in [Q]$, let Π^m_u be the sequence of those $(C, C(m))$ that get (sk_C, u) as query response by S_1.
2. For all $j \in [Q_{\mathsf{pre}}]$, let $(\mathsf{sk}_{C_j,u}, u)$ be the reply to query C_j to S_1. Let $\mathsf{qset}_j = u$ and $\mathsf{uset}_u = \mathsf{uset}_u + 1$, i.e. note down the statistics about each u.
3. For every $u \in [Q]$, it simulates the ciphertext as $\mathsf{ct}_u \leftarrow \mathsf{TgFE.S}_2(u, \Pi^m_u)$ and keeps $\mathsf{TgFE.st}$ updated.
4. For $j \in [Q_{\mathsf{pre}} + 1, Q]$, sample $u \leftarrow [Q]$ and set $\mathsf{qset}_j = u$, $\mathsf{uset}_u = \mathsf{uset}_u + 1$. For every $u \in [Q]$, if $\mathsf{uset}_u > \lambda$, then output $\perp$. Add all the local state (such as qset, Q_{pre} etc) and $\mathsf{TgFE.st}$ to the simulator state's st_2 for post-challenge key generation simulation.

$\mathsf{S}_3^{U_m(\cdot)}(\mathsf{st}_2, C)$

1. The state st_2 contains qset, Q_{pre}, and state stored by tagged FE simulator $\mathsf{TgFE.st}$. On the j^{th} key query made by $\mathcal{A}$ to S_3, it sets $u = \mathsf{qset}_{Q_{\mathsf{pre}}+j}$ (as j-th post-challenge query is $(Q_{\mathsf{pre}} + j)$-th overall key query).
2. It queries $U_m(\cdot)$ to learn $C(m)$. It simulates secret key as $\mathsf{sk}_{C,u} \leftarrow \mathsf{TgFE.S}_1((f, \mathsf{tg}, \mu) = (C, u, C(m)))$ and keeps $\mathsf{TgFE.st}$ updated. It outputs $(\mathsf{sk}_{C,u}, u)$.

Note that we can verify that this satisfies the efficiency requirement of strong simulation security. S_0 only runs TgFE with polynomial dependence on λ, n and Q, where the only dependence on Q is in the size of the tag space. By efficiency of TgFE, this has logarithmic in Q efficiency. Similarly, in S_1 and S_3, the dependence on Q is only in the tag space, for which we only have to randomly sample elements (from $[Q]$ in one case and $\mathsf{qset}_{Q_{\mathsf{pre}}+j}$ in the other) or invoke the TgFE simulator once.

Next, we will show through a sequence of experiments that the real and simulated games are computationally indistinguishable.

Experiment 0. This is the experiment with adversary $\mathcal{A}$ and weakly-optimal static-bounded-collusion FE scheme. The experiment is parameterized by $\lambda \in \mathbb{N}$.

$$\left\{ \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) : \begin{array}{c} (1^n, 1^Q) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, Q) \\ m \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk}) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, m) \end{array} \right\}$$

- **Setup:** $(1^n, 1^Q) \leftarrow \mathcal{A}(1^\lambda)$. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{TgFE.Setup}(1^\lambda, 1^n, 1^{\lceil \log Q \rceil}, 1^\lambda)$. Send mpk to $\mathcal{A}$.
- **Pre Challenge Key Queries:** $\mathcal{A}$ makes a key query for a circuit C.
  1. Sample $u \leftarrow [Q]$. Generate $\mathsf{sk}_{C,u} \leftarrow \mathsf{TgFE.KeyGen}(\mathsf{msk}, u, C)$. It outputs $(\mathsf{sk}_{C,u}, u)$ to $\mathcal{A}$.
- **Ciphertext Queries:** $\mathcal{A}$ outputs a message m.
  1. $\forall u \in [Q], \mathsf{ct}_u \leftarrow \mathsf{TgFE.Enc}(\mathsf{mpk}, u, m)$. Output $\mathsf{ct} \leftarrow (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$ to $\mathcal{A}$.
- **Post Challenge Key Queries:** $\mathcal{A}$ makes a key query for a circuit C.
  1. Sample $u \leftarrow [Q]$. Generate $\mathsf{sk}_{C,u} \leftarrow \mathsf{TgFE.KeyGen}(\mathsf{msk}, u, C)$. It outputs $(\mathsf{sk}_{C,u}, u)$ to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b.

Experiment 1. In this experiment we just make syntactical changes by pre-sampling some of adversaries responses and keeping count of the amount of key generation queries we output on a particular tag.

- **Ciphertext Queries:** $\mathcal{A}$ outputs a message m.
  1. For all $u \in [Q]$, let $\mathsf{qset}_u = 0$ and $\mathsf{uset}_u = 0$. Let Q_{pre} be the number of queries made by $\mathcal{A}$ in the previous phase.
  2. For all $j \in [Q_{\mathsf{pre}}]$, let $(\mathsf{sk}_{C_j,u}, u)$ be the reply to query C_j made by $\mathcal{A}$ to KeyGen in the previous phase. Let $\mathsf{qset}_j \leftarrow u$ and $\mathsf{uset}_u \leftarrow \mathsf{uset}_u + 1$.
  3. $\forall u \in [Q], \mathsf{ct}_u \leftarrow \mathsf{TgFE.Enc}(\mathsf{mpk}, u, m)$. Output $\mathsf{ct} \leftarrow (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$ to $\mathcal{A}$.
  4. For $j \in [Q_{\mathsf{pre}} + 1, Q]$, sample $u \leftarrow [Q]$ and set $\mathsf{qset}_j \leftarrow u$, $\mathsf{uset}_u \leftarrow \mathsf{uset}_u + 1$.
- **Post Challenge Key Queries:** $\mathcal{A}$ makes a key query for a circuit C.
  1. Let this be the j^{th} query made by $\mathcal{A}$ to KeyGen in this phase. Let $u \leftarrow \mathsf{qset}_{Q_{\mathsf{pre}}+j}$.
  2. Generate $\mathsf{sk}_{C,u} \leftarrow \mathsf{TgFE.KeyGen}(\mathsf{msk}, u, C)$. It outputs $(\mathsf{sk}_{C,u}, u)$ to $\mathcal{A}$.

Experiment 2. In this experiment we output $\perp$ if the random tags sampled for each query are disproportionately sampled. That is, if more than λ keys are sampled for a tag, then abort the experiment and output $\perp$.

- **Ciphertext Queries:** These are answered as in previous experiment, except for every $u \in [Q]$, if $\mathsf{uset}_u > q$, then output $\perp$.

Experiment 3. In this experiment, we use TgFE.Sim to answer KeyGen queries and simulate the ciphertext. The experiment keeps track of states TgFE.st when calling TgFE.Sim. Let states $\mathsf{st}_0, \mathsf{st}_1, \mathsf{st}_2$ be empty states initialized by the challenger to keep track of different phases.

- **Setup:** $(1^n, 1^Q) \leftarrow \mathcal{A}(1^\lambda)$. Let $\mathsf{mpk} \leftarrow \mathsf{TgFE.S}_0(1^\lambda, 1^n, 1^{\lceil \log Q \rceil}, 1^\lambda)$, and $\mathsf{st}_0 \leftarrow \mathsf{TgFE.st}$. Send mpk to $\mathcal{A}$.

- **Pre Challenge Key Queries:** $\mathcal{A}$ makes a key query for a circuit C.

  1. Sample $u \leftarrow [Q]$. It simulates secret key as $\mathsf{sk}_{C,u} \leftarrow \mathsf{TgFE.S}_1((f, \mathsf{tg}, \mu) = (C, u, \perp))$ and keeps $\mathsf{TgFE.st}$ updated. It also adds (C, u) as the key query to its state st_1 and ensures $\mathsf{TgFE.st}$ is included in st_1. It outputs $(\mathsf{sk}_{C,u}, u)$ to $\mathcal{A}$.

- **Ciphertext Queries:** $\mathcal{A}$ outputs a message m.

  1-2. Perform steps 1,2 as in previous experiment.

  3. Let $|\Pi^m| = Q_{\mathsf{pre}}$ and $\Pi^m = ((C_1, C_1(m)), \ldots, (C_{Q_{\mathsf{pre}}}, C_{Q_{\mathsf{pre}}}(m)))$. For every $u \in [Q]$, let Π^m_u be the sequence of those $(C, C(m))$ that get (sk_C, u) as query response by S_1. For every $u \in [Q]$, $(\mathsf{st}_1, \mathsf{ct}_u) \leftarrow \mathsf{TgFE.S}_2(\mathsf{st}_1, u, \Pi^m_u)$ and keep $\mathsf{TgFE.st}$ updated. Add all the local state (such as qset, Q_{pre} etc) and $\mathsf{TgFE.st}$ to the simulator state's st_2 for post-challenge key generation simulation. Output $\mathsf{ct} \leftarrow (\mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$ to $\mathcal{A}$.

  4. Perform step 4 as previously.

- **Post Challenge Key Queries:** $\mathcal{A}$ makes a key query for a circuit C.

  1. The state st_2 contains qset, Q_{pre}, and state stored by tagged FE simulator $\mathsf{TgFE.st}$. Let this be the j^{th} query made by $\mathcal{A}$ to KeyGen in this phase. Let $u \leftarrow \mathsf{qset}_{Q_{\mathsf{pre}}+j}$.
  2. It computes $C(m)$. It simulates secret key as $\mathsf{sk}_{C,u} \leftarrow \mathsf{TgFE.S}_1((f, \mathsf{tg}, \mu) = (C, u, C(m)))$ and keeps $\mathsf{TgFE.st}$ updated. It outputs $(\mathsf{sk}_{C,u}, u)$ to $\mathcal{A}$.

- $\mathcal{A}$ outputs a bit b.

Analysis. Let $\mathcal{P}^i_{\mathcal{A}}(\lambda)$ be the probability that adversary $\mathcal{A}$ outputs 1 in Experiment i run on security parameter λ.

Lemma 5.1. For every adversary $\mathcal{A}$, parameter λ, $\mathcal{P}^0_{\mathcal{A}}(\lambda) = \mathcal{P}^1_{\mathcal{A}}(\lambda)$.

Proof. The only changes in the two experiments are syntactical, where query responses are recorded in qset and uset. Additionally, in the post challenge query phase, the recorded queries are pre-sampled in the challenge phase in exactly the same way. □

Lemma 5.2. For every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^1_{\mathcal{A}}(\lambda) - \mathcal{P}^2_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. The experiments are different if there exists a $u \in [Q]$, such that $\mathsf{uset}_u > q$. Recall that q was set to λ in the construction. For every $u \in [Q]$ and $j \in [Q]$, let $X_{j,u}$ be a random variable that is 1 if u was sampled during the j^{th} query response. For any $u \in [Q]$, let $X_u = \sum_{j=1}^{Q} X_{j,u}$. It's easy to see that $\mathcal{E}[X_{j,u}] = \frac{1}{Q}$ and by linearity of expectation, $\mathcal{E}[X_u] = 1$. By a Chernoff bound, we have that,

$$\Pr[X_u > 1 + (\lambda - 1)] \leq e^{-\frac{(\lambda-1)^2}{(\lambda+1)}}$$

This is $\leq e^{-\frac{\lambda^2/4}{2\lambda}} \leq e^{-\lambda/8}$ for $\lambda \geq 2$. The probability that there exists such a u, is bounded by the union bound to be $\leq Q \cdot e^{-\lambda/8} = \mathsf{negl}(\lambda)$. The latter holds since Q is polynomially bounded in the security parameter λ. □

Lemma 5.3. Assuming $\mathsf{tgfe} = (\mathsf{TgFE.Setup}, \mathsf{TgFE.Enc}, \mathsf{TgFE.KeyGen}, \mathsf{TgFE.Dec})$ is a simulation-secure tagged FE scheme (as per Definition 4.1), for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^2_{\mathcal{A}}(\lambda) - \mathcal{P}^3_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. The difference in these two experiments is that we use the true TgFE scheme in experiment 2 and we simulate using $\mathsf{TgFE.Sim}$ in experiment 3. This is exactly the TgFE security game and we can construct a reduction $\mathcal{B}$ that proceeds as follows.

- **Setup:** It gets $(1^n, 1^Q)$ from $\mathcal{A}$. $\mathcal{B}$ outputs $(1^n, 1^\lambda, 1^{\lceil \log Q \rceil})$ to the setup oracle. It sets mpk from the experiment and sends it to $\mathcal{A}$.
- **Pre Challenge Key Queries:** $\mathcal{A}$ asks KeyGen queries and $\mathcal{B}$ simulates them by generating a random tag u and querying its $\mathsf{TgFE.KeyGen}$ oracle on circuit C and identity u for sk. It returns (sk, u) to $\mathcal{A}$.
- **Ciphertext Queries:** $\mathcal{A}$ outputs a message m. For every $u \in [Q]$, query the $\mathsf{TgFE.Enc}$ oracle on message m and identity u to recieve ct_u. Return $(\mathsf{ct}_1, \ldots \mathsf{ct}_Q)$ to $\mathcal{A}$.
- **Post Challenge Key Queries:** $\mathcal{A}$ asks KeyGen queries and $\mathcal{B}$ simulates them by generating a random tag u and querying its $\mathsf{TgFE.KeyGen}$ oracle on circuit C and identity u for sk. It returns (sk, u) to $\mathcal{A}$.
- $\mathcal{B}$ gets a bit from $\mathcal{A}$ and outputs it.

Note that the reduction $\mathcal{B}$ works with the execution environment $\mathcal{E}$ that coordinates the information between $\mathsf{TgFE.S}_1, \mathsf{TgFE.S}_2$ for responses. We don't explicitly repeat the transcript of the exchange as the coordination of information is straightforward and described in Definition 4.1. It is easy to see that if the TgFE oracles utilize the real scheme, then we are in experiment 2, but if they are simulated, we are in experiment 3. We can observe that $\mathcal{B}$ is admissible in the tagged FE game whenever $\mathcal{A}$ is admissible in the weakly optimal game, each call to the Enc or KeyGen oracle corresponds to a single call on the same $\mathsf{TgFE.Enc}$ (we call $\mathsf{TgFE.Enc}$ multiple times, each time on a different tag) and $\mathsf{TgFE.KeyGen}$. We can see that $\mathcal{B}$ makes at most q calls to any particular tag in $[Q]$ by the conditional from Experiment 2. From the security of TgFE by Definition 4.1, we have that the distributions are computationally indistinguisable. □

Note that at the end of Experiment 3, $\mathcal{A}$ is interacting exactly with Sim defined above. Thus the output of this experiment is computationally indistinguishable from the output of Experiment 0. □

6 Upgrading Collusion Bound for Tagged FE

Now we show that a bounded-collusion tagged FE scheme where the collusion bound can be any arbitrary polynomial can be generically built from a tagged FE scheme that allows corrupting at most one key per unique tag (i.e., 1-bounded collusion secure) by relying on the client server framework from [AV19]. The ideas behind this transformation are based on the 1-bounded non-tagged FE to Q-bounded non-tagged FE transformation from [AV19].

The client server framework is formally defined later in Appendix A for completeness. Intuitively, in the client server framework, there is a single client and N servers. The computation

proceeds in two phases, an offline phase, where the client encrypts an input x of the protocol for N servers where u-th server gets $\hat{x}^u$. This is followed by an online phase which runs in Q sessions for computation on circuits $C_1, \ldots, C_Q$. In each session $j \in [Q]$, client delegates the computation of C_j by computing $\hat{C}_j^u$ for $u \in [\mathsf{N}]$ and sending $\hat{C}_j^u$ to u-th server. Now $S \subseteq [\mathsf{N}]$ where $|S| = \mathsf{p}$ servers come online and for $u \in S$, u-th server computes $\hat{y}_j^u \leftarrow \mathsf{Local}(\hat{C}_j^u, \hat{x}^u)$. Finally the S server send information back to client who computes $y \leftarrow \mathsf{Decode}(\{\hat{y}_j^u\}_{u \in S}, S)$ for each $j \in [Q]$, to compute $C_j(x)$.

The transformation in [AV19] invokes N (polynomial in Q, λ) many instantiations of the one bounded FE scheme. These N instances act like a separate server in the client server framework. Encryption simply computes the encryption of each one bounded instance on the offline computation on the inputs, i.e. encrypt under $\hat{x}^u$ for $u \in [\mathsf{N}]$ under the one bounded FE algorithm. Key generation computes the online encryption of the circuits, $\hat{C}^u$ for $u \in [\mathsf{N}]$ and picks a random subset S of size p and generates the secret keys on the 1 bounded instance for circuit $\mathsf{Local}(\hat{C}^u, \cdot)$ for $u \in S$. In our transformation, instead of having N independent instances, we instead blowup the tag space for the one tagged FE scheme and perform the key generation and encryption procedures very similarly. The analysis of the correctness and security are very similar to [AV19], except that in the $\mathsf{1TgFE}$ security game (Definition 4.1), we allow the adversary to request for multiple challenge ciphertexts (each on a different tag) and thus the security proof is tweaked adequately.

6.1 Construction

Let $\mathsf{1tgfe} = (\mathsf{1TgFE.Setup}, \mathsf{1TgFE.Enc}, \mathsf{1TgFE.KeyGen}, \mathsf{1TgFE.Dec})$ be a 1-collusion-bounded tagged FE scheme for a family of circuit classes $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$, message spaces $\{\mathcal{M}_n\}_{n \in \mathbb{N}}$, and tag space $\{\mathcal{I}_z = \{0,1\}^z\}_{z \in \mathbb{N}}$. Let $\mathsf{Protocol} = (\mathsf{CktEnc}, \mathsf{Local}, \mathsf{InpEnc}, \mathsf{Decode})$ be the client server framework (see Appendix A for the complete description) for the circuit class $\{\mathcal{F}_n\}_{n \in \mathbb{N}}$. We use $\mathsf{1TgFE}$ to build a tagged FE scheme with arbitrary (but polynomial) collusion bounded for the same circuit class, message space, and slightly shorter tag space.

Let the maximum size of the circuit be denoted by $\kappa(n)$ (denoted by κ for brevity). Let the construction parameters be $\mathsf{t} = \Theta(Q\lambda)$, $\mathsf{N} = \Theta(Q^2\mathsf{t}^2)$, $\mathsf{p} = \Theta(\mathsf{t})$.

$\mathsf{Setup}(1^\lambda, 1^n, 1^z, 1^Q) \to (\mathsf{mpk}, \mathsf{msk})$. It samples the master key pair as

$$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{1TgFE.Setup}(1^\lambda, 1^n, 1^{z+\lceil \log \mathsf{N} \rceil}, 1).$$

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{tg}, C) \to \mathsf{sk}_{C,\mathsf{tg}}$. It samples a set $S \leftarrow [\mathsf{N}]$ of cardinality p, and computes the N-server encoding of circuit C as $(\hat{C}^1, \ldots, \hat{C}^{\mathsf{N}}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C)$.

Let $\mathsf{E}^u(\cdot) = \mathsf{Local}(\hat{C}^u, \cdot)$. It generates a key for circuit E^u under $\mathsf{1TgFE}$, i.e. $\forall u \in S, \mathsf{sk}_{\mathsf{E}^u} \leftarrow \mathsf{1TgFE.KeyGen}(\mathsf{msk}, (u, \mathsf{tg}), \mathsf{E}^u)$. Output $(\{(u, \mathsf{sk}_{\mathsf{E}^u})\}_{u \in S}, S)$.

Notation. Here and throughout the paper, we use (u, tg) to denote the $(z + \lceil \log \mathsf{N} \rceil)$-bit tag, where u encodes the first $\lceil \log \mathsf{N} \rceil$-bits and tg is encoded in the remaining bits.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{tg}, m) \to \mathsf{ct}$. The encryptor first computes the input encoding $(\hat{x}^1, \ldots, \hat{x}^{\mathsf{N}}) \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, 1^\kappa, m)$, and then for every $u \in [\mathsf{N}]$, it computes $\mathsf{ct}_u \leftarrow \mathsf{1TgFE.Enc}(\mathsf{mpk}, (u, \mathsf{tg}), \hat{x}^u)$. Output $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{\mathsf{N}})$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{tg},C}, \mathsf{ct}) \to y$. Let $\mathsf{sk}_{\mathsf{tg},C} = (\{(u, \mathsf{sk}_{\mathsf{E}^u})\}_{u \in S}, S)$, and $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_{\mathsf{N}})$. For every $u \in S$, compute $\hat{y}^u \leftarrow \mathsf{1TgFE.Dec}(\mathsf{sk}_{\mathsf{E}^u}, \mathsf{ct}_u)$. Output $y \leftarrow \mathsf{Decode}(\{\hat{y}^u\}_{u \in S}, S)$.

6.2 Correctness, Efficiency, and Security

The correctness of the above scheme follows from the correctness of $\mathsf{1TgFE}$ scheme and algorithms $\mathsf{CktEnc}, \mathsf{InpEnc}, \mathsf{Local}, \mathsf{Decode}$. By the correctness of $\mathsf{1TgFE}$, we know that for every $u \in S$, $\mathsf{1TgFE.Dec}(\mathsf{sk}_{\mathsf{E}^u}, \mathsf{ct}_u) = \mathsf{Local}(\hat{C}^u, \hat{x}^u)$ (the keygen and the encryption are both on tag (u, tg) . By the correctness of $\mathsf{Protocol}$, we know that,

$$\mathsf{Decode}(\{\mathsf{Local}(\hat{C}^u, \hat{x}^u)\}_{u \in S}, S) = C(m).$$

For the efficiency, as $\mathsf{Protocol}$ consists of polynomial algorithms, it's easy to see that the runtime is polynomial in λ, n, Q, z.
To conclude, we prove the following.

Theorem 6.1. If $\mathsf{1TgFE} = (\mathsf{1TgFE.Setup}, \mathsf{1TgFE.Enc}, \mathsf{1TgFE.KeyGen}, \mathsf{1TgFE.Dec})$ is a tagged-statically-bounded-collusion simulation-secure FE scheme for collusion bound 1 (as per Definition 4.1), and $\mathsf{Protocol}$ is secure according to Definition A.1, then the above construction is a simulation-secure tagged FE scheme for collusion bound Q (as per Definition 4.1).

Proof. Let $\mathsf{1TgFE.Sim} = (\mathsf{1TgFE.S}_0, \mathsf{1TgFE.S}_1, \mathsf{1TgFE.S}_2)$ be the simulators satisfying Definition 4.1 for a 1-query collusion bound . Let $\mathsf{Protocol.Sim}$ be the simulator satisfying Definition A.1 for the MPC framework. We will construct simulators $\mathsf{Sim} = (\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2)$ that satisfy Definition 4.1 for a Q-query collusion bound. Both sets of simulators implicitly share a global state between them for coordination.

$\mathsf{S}_0(1^\lambda, 1^n, 1^z, 1^Q)$
Let $\mathsf{mpk} \leftarrow \mathsf{1TgFE.S}_0(1^\lambda, 1^n, 1^{z+\lceil \log \mathsf{N} \rceil}, 1)$.

$\mathsf{S}_1(\mathsf{tg}, C, \mu)$

1. Let this be the j-th query to S_1 and the input query be denoted by $\mathsf{tg}_j, C_j, \mu_j$. Let this be the i-th query to tag tg_j.
2. If $i = 1$, initialize the simulator $\mathsf{Protocol.Sim}_{\mathsf{tg}_j}$ on input $(1^\lambda, 1^Q, 1^\kappa)$.
3. **Pre Challenge Key Queries:** If $\mu_j = \perp$,
   (a) Compute $(\hat{C}_j^1, \ldots, \hat{C}_j^{\mathsf{N}}) \leftarrow \mathsf{Protocol.Sim}_{\mathsf{tg}_j}(C_j)$. Let $\mathsf{E}_j^u(\cdot) = \mathsf{Local}(\hat{C}_j^u, \cdot)$.
   (b) Sample a set $S_i^{\mathsf{tg}_j} \leftarrow [\mathsf{N}]$ of cardinality p.
   (c) Generate a key for circuit E_j^u under the one tagged scheme, $\forall u \in S_i^{\mathsf{tg}_j}$, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u, \mathsf{tg}_j), \mathsf{E}_j^u, \perp)$. Output $(\{(u, \mathsf{sk}_{\mathsf{E}_j^u})\}_{u \in S_i^{\mathsf{tg}_j}}, S_i^{\mathsf{tg}_j})$.
4. **Post Challenge Key Queries:** Else,
   (a) Compute the simulated circuit, $(\hat{C}_j^1, \ldots, \hat{C}_j^{\mathsf{N}})$ and $\{\hat{y}_i^u\}_{u \in S_i^{\mathsf{tg}_j}}$ by running $\mathsf{Protocol.Sim}_{\mathsf{tg}_j}$ on $(i, C_j, C_j(m^{\mathsf{tg}_j}))$. Note that $\mu_j = C_j(m^{\mathsf{tg}_j})$. Let $\mathsf{E}_j^u(\cdot) = \mathsf{Local}(\hat{C}_j^u, \cdot)$ for $u \in [\mathsf{N}]$.
   (b) Generate a key for circuit E_j^u under the one tagged scheme, i.e. $\forall u \in S^{\mathsf{tg}_j}$,
      i. If $u \in \mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}_j}$, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u, \mathsf{tg}_j), \mathsf{E}_j^u, \perp)$.[7]
      ii. Else, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u, \mathsf{tg}_j), \mathsf{E}_j^u, \hat{y}_i^u)$.

[7] Note that on these tags, no ciphertext query is made.

Output $(\{(u, \mathsf{sk}_{\mathsf{E}_j^u})\}_{u \in S_i^{\mathsf{tg}_j}}, S_i^{\mathsf{tg}_j})$.

$\mathsf{S}_2(\mathsf{tg}^*, \Pi^{m^{\mathsf{tg}^*}})$

1. Let $Q_{\mathsf{pre}}^{\mathsf{tg}^*}$ be the number of key generation queries made to tag tg^*. Let the set $\Pi^{m^{\mathsf{tg}^*}}$ denote the set of all $(C_j, C_j(m^{\mathsf{tg}^*}))$ where for $j \in [Q]$, (tg^*, C_j) was the key generation query.
2. For every $i \in [Q_{\mathsf{pre}}^{\mathsf{tg}^*} + 1, Q]$, sample a set $S_i^{\mathsf{tg}^*}$ of size p uniformly at random from $[\mathsf{N}]$.
3. Construct a corrupted set $\mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}$ that includes all tags that are corruped as follows,

$$\mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*} = \{u \in [\mathsf{N}] \ : \ \exists i, j \in [Q], i \neq j \text{ and } u \in S_i^{\mathsf{tg}^*} \cup S_j^{\mathsf{tg}^*}\}.$$

   If $|\mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}| > \mathsf{t}$, output $\perp$.
4. If $Q_{\mathsf{pre}}^{\mathsf{tg}^*} = 0$, initialize the simulator $\mathsf{Protocol.Sim}_{\mathsf{tg}^*}$. Compute $\mathsf{Protocol.Sim}_{\mathsf{tg}^*}(1^{|m^{\mathsf{tg}^*}|}, \mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}, \Pi^{m^{\mathsf{tg}^*}})$ [8] to obtain $\{\hat{x}^u\}_{u \in \mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}}$ and evalutations $\{\hat{y}_i^u\}_{i \in [Q_{\mathsf{pre}}^{\mathsf{tg}^*}], u \in S_i^{\mathsf{tg}^*}}$. Additionally we have that $\hat{y}_i^u = \mathsf{Local}(\hat{C}_j^u, \hat{x}^u)$ for $u \in S_i^{\mathsf{tg}^*} \cap \mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}$ where $i \in [Q_{\mathsf{pre}}^{\mathsf{tg}^*}]$ and j is the global query number (recall i is the query number on tg^*).
5. For every $u \in [\mathsf{N}]$,
   (a) If $u \in \mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}$, run the normal encryption procedure, i.e.

$$\mathsf{ct}_u \leftarrow \mathsf{1TgFE.Enc}(\mathsf{mpk}, (u, \mathsf{tg}^*), \hat{x}^u).$$

   (b) Else, query $\mathsf{1TgFE.S}_2$ for the ciphertexts. Let the shares we've previously output be noted down in $\Pi'_u = \{(\mathsf{E}_j^u, \hat{y}_i^u)\}_{i \in [Q_{\mathsf{pre}}^{\mathsf{tg}^*}], u \in [\mathsf{N}]}$ (j is the total query number to S_1 where this is the i-th query on tg_j) and compute,

$$\mathsf{ct}_u \leftarrow \mathsf{1TgFE.S}_2((u, \mathsf{tg}^*), \Pi'_u).$$

The security proof proceeds in a sequence of hybrids, where we move from the real experiment to the simulated experiment. First, we switch the order of sampling sets S^{tg} on a post challenge query, i.e. on a challenge tag, tg^*, we sample the post challenge sets S^{tg^*} identically and use these for post challenge queries. Then, we set the experiment to fail if the size of the corrupted set $\mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}$ exceeds a threshold t. This guarantees that enough servers are not corrupted for the security of the MPC protocol to hold. This failure happens with a negligible probability using an information theoretic lemma. Now, we can rely on the $\mathsf{1tgfe}$ simulator to remove some information about the message. Note that for the corrupted set for each challenge tag tg^*, we don't request a challenge query and crucially rely on the security of the non-corrupted set. Finally, the remaining information about the message (except the circuit evaluations) is removed by relying on the security of the MPC protocol, which works at a high level by relying on only the message length $|m|$ in the offline phase and the circuit evaluations during the online phase. Thus we end up we in the simulated experiment. The detailed proof with complete hybrid experiments is described later in Appendix C.

□

[8] We can assume simulator knows the message length as we have the empty function description, see Footnote 6.

7 Building 1-Bounded Collusion Tagged FE from IBE

Here we construct a tagged FE scheme that achieves security in the 1-bounded collusion model and, as we discussed in the previous section, this is sufficient to build a general bounded-collusion tagged FE scheme. Our construction is itself split into two components where first we provide a simple construction using garbled circuits and IBE but it only achieves non-adaptive security, and later we show how to generically upgrade it to full adaptive security by relying on non-committing encryption techiques.

7.1 Non-Adaptive 1-Bounded Tagged FE from Garbled Circuits and IBE

The non-adaptive construction is a close adaptation of the traditional construction of 1-bounded FE from public key encryption and garbled circuits found in [SS10, GVW12]. The idea is to simply encrypt all the wire labels of a garbling of a universal circuit using IBE and only give out select IBE secret keys of the wires corresponding to the circuit. For simplicity, we assume that the functionality class $\mathcal{F}_n$ includes all circuits of size n (the circuit description is n bits long).

$\mathsf{Setup}(1^\lambda, 1^n, 1^z) \to (\mathsf{mpk}, \mathsf{msk})$. Sample an IBE master key pair as $(\mathsf{ibe.pk}, \mathsf{ibe.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda, 1^{z+\lceil \log n \rceil + 1})$, and output $\mathsf{mpk} = \mathsf{ibe.pk}, \mathsf{msk} = \mathsf{ibe.msk}$.

Notation. Here and throughout the paper, we use (b, i, tg) to denote the $(z + \lceil \log n \rceil + 1)$-bit identity, where b is a single bit, i encodes $\lceil \log n \rceil$)-bits, and tg is encoded in the remaining z bits. Basically, each bit-index-tag tuple is uniquely and efficiently mapped into the identity space.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{tg}, m \in \mathcal{M}_n) \to \mathsf{ct}$. Let $\mathcal{U}$ be the universal circuit for the family of size n circuits on inputs in $\mathcal{M}_n$ (i.e., $\mathcal{U}(C, m) = C(m)$). Now in the following garble $\mathcal{U}(\cdot, m)$ as $(\hat{\mathcal{U}}, \{w_{i,b}\}_{i \le n, b \in \{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathcal{U})$, and encrypt the labels as

$$\forall i \in n, b \in \{0,1\}, \quad \mathsf{ct}_{i,b} \leftarrow \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b, i, \mathsf{tg}), w_{i,b})$$

It finally outputs $\mathsf{ct} = (\hat{\mathcal{U}}, \{\mathsf{ct}_{i,b}\}_{i \le n, b \in \{0,1\}})$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{tg}, C \in \{0,1\}^n) \to \mathsf{sk}_{\mathsf{tg},C}$. Let $C[1], C[2], \ldots C[n]$ denote the bit representation of circuit C. It samples n IBE secret keys as $\mathsf{sk}_i = \mathsf{IBE.KeyGen}(\mathsf{msk}, (C[i], i, \mathsf{tg}))$ for $i \in [n]$, and outputs $\mathsf{sk}_{\mathsf{tg},C} = \{\mathsf{sk}_i\}_{i \in [n]}$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{tg},C}, \mathsf{ct}) \to y$. It parses the secret key and ciphertext as above. It first decrypts the wire keys as $w_{i,C[i]} \leftarrow \mathsf{IBE.Dec}(\mathsf{sk}_i, \mathsf{ct}_{i,C[i]})$ for $i \in [n]$, and then outputs $y = \mathsf{GC.Eval}(\hat{\mathcal{U}}, \{w_{i,C[i]}\}_{i \in [n]})$.

Theorem 7.1. If ibe is a secure IBE scheme per Definition 2.2 and GC is a secure garbled circuit Definition 2.1, then the above is a non-adaptively secure tagged 1-bounded FE scheme.

Proof. Consider the following simulator for the above construction.

$\mathsf{S}_0(1^\lambda, 1^n, 1^z)$ - Setup is not changed in simulation, we simply run Setup, outputting the Setup public key (which is an IBE public key) as the public key and storing the IBE master secret key as internal state.

$\mathsf{S}_1(\mathsf{tg}, C, \perp)$ - Simulation of key generation also remains the same, simply running the KeyGen algorithm using the stored msk state and handing out a subset of IBE secret keys.

$\mathsf{S}_2(\mathsf{tg}^*, \Pi^m = (C, C(m)))$ - Here, given the output to the single functional key queried, we can simulate the garbled circuit, and encrypt the simulated labels under the IBE keys. Since this only generates half the labels a normal garbling scheme would, the other half of the labels are replaced with garblings of 0.

The security proof proceeds in a sequence of hybrids, where we move from the real experiment to the simulated experiment. In our real experiment encrypt outputs encryptions of label for each possible circuit bit. Since this is a non-adaptive experiment, we switch the encryption to output encryptions of zero, wherever the tags don't match the circuit bit (circuit to be queried is known while making the ciphertext). This step uses the IBE security as observe that IBE secret keys for these tags will never be requested. Finally, to remove information about the message and by relying on garbled circuit security with the evaluation $C(m)$ we can simulate the remaining labels. At this point there is no information remaining about m, except the evaluation $C(m)$ and we are in the simulated experiment. The detailed proof with complete hybrid experiments is described later in Appendix D.

□

7.2 Upgrading to Adaptive Security

We can transform any non-adaptive 1-bounded tagged FE scheme to an adaptive one using IBE. This is an analogue of the traditional method of using weak non-committing encryption (which is constructable from plain public key encryption) to make 1-bounded FE adaptive. Here, the encryption has two modes — a 'normal mode', where the scheme functions like a normal public key/IBE scheme, and a non-committing mode, where the encryptor can produce secret keys which equivocate to any value. This enables us to delay simulating the ciphertext of a adaptive key queries until the secret key is requested.

$\mathsf{Setup}(1^\lambda, 1^n, 1^z) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm runs the underlying tagged FE and IBE setup algorithms as $(\mathsf{natgfe.pk}, \mathsf{natgfe.msk}) \leftarrow \mathsf{NATgFE.Setup}(1^\lambda, 1^n, 1^z)$, and $(\mathsf{ibe.pk}, \mathsf{ibe.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda, 1^{z+\lceil \log n' \rceil + 1})$ where n' denotes the length of natgfe ciphertexts with the above parameters.

It outputs the master keys as $(\mathsf{mpk}, \mathsf{msk}) = ((\mathsf{natgfe.pk}, \mathsf{ibe.pk}), (\mathsf{natgfe.msk}, \mathsf{ibe.msk}))$.

Notation. Here and throughout the paper, we use (b, i, tg) to denote the $(z + \lceil \log n' \rceil + 1)$-bit identity, where b is a single bit, i encodes $\lceil \log n' \rceil$)-bits, and tg is encoded in the remaining z bits. Basically, each bit-index-tag tuple is uniquely and efficiently mapped into the identity space.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{tg}, m) \to \mathsf{ct}$. Let $\mathsf{mpk} = (\mathsf{natgfe.pk}, \mathsf{ibe.pk})$. It encrypts m using tagged FE as $\mathsf{ct}' \leftarrow \mathsf{NATgFE.Enc}(\mathsf{natgfe.pk}, \mathsf{tg}, m)$, and it encrypts ct' bit-by-bit under IBE as $c_{b,j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b, j, \mathsf{tg}), \mathsf{ct}'[j])$ for $b \in \{0, 1\}, j \in [n']$. It then outputs $\mathsf{ct} = \{c_{b,j}\}_{b \in \{0,1\}, j \in [n']}$.

$\mathsf{KeyGen}(\mathsf{msk}, \mathsf{tg}, C \in \{0,1\}^n) \to \mathsf{sk}_{\mathsf{tg},C}$. Let $\mathsf{msk} = (\mathsf{natgfe.msk}, \mathsf{ibe.msk})$. It samples n' random bits $b_1, b_2, \ldots, b_{n'} \leftarrow \{0,1\}$, and computes secret keys for the underlying systems as:

$$\begin{aligned} &\mathsf{natgfe.sk}_{\mathsf{tg},C} \leftarrow \mathsf{NATgFE.KeyGen}(\mathsf{natgfe.msk}, \mathsf{tg}, C), \\ \forall j \in [n'], \quad &\mathsf{ibe.sk}_{b_j,j} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{ibe.msk}, (b_j, j, \mathsf{tg})). \end{aligned}$$

And it outputs $\mathsf{sk}_{\mathsf{tg},C} = (\mathsf{natgfe.sk}_{\mathsf{tg},C}, \{(b_j, \mathsf{ibe.sk}_{j,b_j})\}_{j\in[n']})$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{tg},C}, \mathsf{ct}) \to y$**.** It parses the secret key and ciphertext as above. It first decrypts the IBE ciphertexts as $\mathsf{ct}'[j]$ as $\mathsf{IBE.Dec}(\mathsf{ibe.sk}_{b_j,j}, \mathsf{ct}_{b,j})$ for $j \in [n']$, and then computes $y = \mathsf{NATgFE.Dec}(\mathsf{natgfe.sk}_{\mathsf{tg},C}, \mathsf{ct}')$ where $\mathsf{ct}'[i]$ is the i-th bit of ct'.

Theorem 7.2. If ibe is a secure IBE scheme and natgfe is a non-adaptively secure 1-bounded tagged FE scheme, then the above is an adaptively secure 1-bounded tagged FE scheme.

Proof. Consider the following simulator for the above construction. The simulators jointly maintain an internal state st. Since this simulator calls the natgfe.Sim which is also stateful, consider this simulator to implicitly track the internal state of natgfe.Sim in addition to the explicit state values. Since the simulators shares an internal state, it can track whether, on each tag the adversary queried the keygen oracle first or the ciphertext oracle, and classify queries on said tag as non-adaptive or adaptive respectively.

$\mathsf{S}_0(1^\lambda, 1^n, 1^z)$: Run $\mathsf{natgfe.S}_0$ and and ibe.Setup with identity space input as 1^z and $1^{z+\lceil \log n' \rceil + 1}$ respectively, outputting the public keys to both as the public key of the scheme. Retain both secret keys in the internal state.

Non-adaptive Queries: The execution of the simulators S_1 and S_2 here exactly follows the honest execution of KeyGen and Enc. Simply replacing the underlying calls to natgfe.KeyGen and natgfe.Enc with calls to $\mathsf{natgfe.S}_1$ and $\mathsf{natgfe.S}_2$ suffices.

Adaptive Queries: When the adversary first requests an oracle call to Enc, instead of calling natgfe.Enc, the simulator S_2 randomly encrypts one of $\{0,1\}$ to each of $\mathsf{ct}_{i,0}$ and $\mathsf{ct}_{i,1}$ for each position. Now, S_1 is called, it not only runs the $\mathsf{natgfe.S}_1$ call to simulate the natgfe.KeyGen, but also makes the delayed call to $\mathsf{natgfe.S}_2$ with the aid of the $\mu = C(m)$ information, and gives out IBE keys only to the corresponding $\mathsf{ct}_{i,b}$ which match the bit of the simulated ciphertext.

The security proof proceeds in a sequence of hybrids, where we move from the real experiment to the simulated experiment. For a high level intuition we assume that the adversary only makes adaptive secret key queries as we already have a non-adaptively secure primitive with us. First we switch the order of sampling $\{b_1, \ldots, b_{n'}\}$ on a post challenge query, i.e. on a challenge tag, tg, we sample $\{b_1, \ldots, b_{n'}\}$ identically for post challenge and store it in the global st. Then, we change how the secret keys are generated in the adaptive case. Instead of using b_j as the random bit, we instead use $b_j \oplus \mathsf{ct}'[j]$ as the random bit. We argue that the distributions remain same information theortecally as b_j is random and $\mathsf{ct}'[j]$ is independent. We can now use IBE security to switch the IBE encryption on the encryption where we do not request a key. Thus on tag $\overline{b_j \oplus \mathsf{ct}'[j]}$ we instead output encryption of the complement bit $\overline{\mathsf{ct}'[j]}$. Our end goal is to remove information about the

message and now we can set the ciphertext without the knowledge of the message by outputting encryptions of 0 and 1 (appropriately). To coordinate with the adaptive keygen query, we still have to generate ct' in the post challenge phase using the message m so that the decryption can compute ct' and thus $C(m)$. This trick where we set the ciphertext to an encryption of both 0 and 1 and then set key query according to the value of ct' generated during the post challenge query is why we are adaptively secure. The technique in this sub-hybrid change is very similar to that of weak non-commiting encryption [CFGN96, DN00, KO04] which is built from public-key encryption. The final step is to remove the message information by relying on non-adaptive security. Observe that we never make an adaptive query to the non-adaptive primitive as both operations of generating the key and the ciphertext happen in the post challenge phase. Thus we end up we in the simulated experiment. The detailed proof with complete hybrid experiments is described later in Appendix E. □

Acknowledgements. We thank the anonymous reviewers for CRYPTO 2021 for useful feedback regarding our abstractions.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (h)ibe in the standard model. In *Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'10, pages 553–572, Berlin, Heidelberg, 2010. Springer-Verlag.

[Agr17] Shweta Agrawal. Stronger security for reusable garbled circuits, general definitions and attacks. In *Annual International Cryptology Conference*, 2017.

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

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Cryptology ePrint Archive, Report 2015/730, 2015.

[AMVY21] Shweta Agrawal, Monosij Maitra, Narasimha Sai Vempati, and Shota Yamada. Functional encryption for turing machines with dynamic bounded collusion from lwe. To appear in CRYPTO, 2021.

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

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 174–198. Springer, 2019.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, 2001.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS '12*, 2012.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. In *Proceedings of the 8th conference on Theory of cryptography*, TCC'11, pages 253–273, Berlin, Heidelberg, 2011. Springer-Verlag.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 171–190, 2015.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *Proceedings of the 4th conference on Theory of cryptography*, TCC'07, pages 535–554, Berlin, Heidelberg, 2007. Springer-Verlag.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multiparty computation. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 639–648. ACM, 1996.

[CHH+07] Ronald Cramer, Goichiro Hanaoka, Dennis Hofheinz, Hideki Imai, Eike Kiltz, Rafael Pass, Abhi Shelat, and Vinod Vaikuntanathan. Bounded cca2-secure encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 502–518. Springer, 2007.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 523–552, 2010.

[Coc01] Clifford Cocks. An identity based encryption scheme based on Quadratic Residues. In *Cryptography and Coding, IMA International Conference*, volume 2260 of LNCS, pages 360–363, 2001.

[CVW+18] Yilei Chen, Vinod Vaikuntanathan, Brent Waters, Hoeteck Wee, and Daniel Wichs. Traitor-tracing from lwe made simple and attribute-based. In *TCC*, 2018.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography, 1976.

[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.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Annual International Cryptology Conference*, pages 432–450. Springer, 2000.

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

[GKW16] Rishab Goyal, Venkata Koppula, and Brent Waters. Semi-adaptive security and bundling functionalities made generic and easy. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, 2016.

[GKW18] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In *STOC*, 2018.

[GLW12] Shafi Goldwasser, Allison Lewko, and David A Wilson. Bounded-collusion ibe from key homomorphism. In *Theory of Cryptography Conference*, 2012.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GSW21] Rishab Goyal, Ridwan Syed, and Brent Waters. Bounded collusion abe for tms from ibe. Cryptology ePrint Archive, Report 2021/709, 2021.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.

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

[KMUW18] Lucas Kowalczyk, Tal Malkin, Jonathan Ullman, and Daniel Wichs. Hardness of non-interactive differential privacy from one-way functions, 2018.

[KO04] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *Annual International Cryptology Conference*, pages 335–354. Springer, 2004.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, 2008.

[Sha85] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[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*, CCS '10, pages 463–472, New York, NY, USA, 2010. ACM.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, pages 475–484, 2014.

[Yao86] Andrew Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167, 1986.

A Client Server framework

This section is copied almost verbatim from [AV19], barring small notational changes. We include the framework definition and security requirements for completeness.

Let $\mathsf{Protocol}$ be a multi party protocol for a family of circuit classes $\{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$, message spaces $\{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$. It consists of a single client and $\mathsf{N}(\lambda, Q)$ servers where λ is the security parameter and Q is the number of sessions. Additionally the construction admits parameters $\mathsf{p}(\lambda, Q)$ and $\mathsf{t}(\lambda, Q)$. Let the maximumm size of the circuit $|\mathcal{F}_\lambda|$ be denoted by $\kappa(\lambda)$. The framework is described in two phases.

Offline Phase In this phase, the client takes as input the number of sessions Q, size of the circuit delegated κ, input x and executes a PPT algorithm InpEnc that outputs correlated input encodings $(\hat{x}^1, \ldots, \hat{x}^{\mathsf{N}})$. It sends the encoding $\hat{x}^u$ to the u^{th} server.

$$(\hat{x}^1, \ldots, \hat{x}^{\mathsf{N}}) \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, 1^\kappa, x)$$

Online Phase This phase is executed for Q sessions. In each session, the client delegates the computation of a circuit C on x to the servers in the following steps.

- **Client Delegation:** This is performed by the client computing a PPT algorithm $\mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C)$ to obtain $(\hat{C}^1, \ldots, \hat{C}^{\mathsf{N}})$. It sends the circuit encoding $\hat{C}^u$ to the u^{th} server.
$$(\hat{C}^1, \ldots, \hat{C}^{\mathsf{N}}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C).$$
- **Local Computation by the Servers:** Upon receiving the circuit encodings from the client, a subset S of servers come online and the u^{th} server in the set S computes $\mathsf{Local}(\hat{C}^u, \hat{x}^u)$ to obtain the u^{th} output encoding $\hat{y}^u$.
$$\forall u \in S, \quad \hat{y}^u \leftarrow \mathsf{Local}(\hat{C}^u, \hat{x}^u).$$
- **Decoding:** Finally, the output is recovered by computing a PPT algorithm Decode on $(\{\hat{y}^u\}_{u\in S}, S)$.
$$y \leftarrow \mathsf{Decode}(\{\hat{y}^u\}_{u\in S}, S).$$

Correctness. The protocol $\mathsf{Protocol}$ is said to be correct if for all $\lambda, \in \mathbb{N}$, circuits $C \in \mathcal{F}_\lambda$ with circuit size bounded by κ, messages $x \in \mathcal{M}_\lambda$ we have that, for $(\hat{C}^1, \ldots, \hat{C}^{\mathsf{N}}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C)$ and $(\hat{x}^1, \ldots, \hat{x}^{\mathsf{N}}) \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, 1^\kappa, x)$,

$$\Pr\left[\mathsf{Decode}(\{\mathsf{Local}(\hat{C}^u, \hat{x}^u)\}_{u\in S}, S) = C(x)\right] = 1.$$

Security. The security allows an adversary to be able to corrupt a subset of the servers. Once a server is compromised, the entire state of the server is leaked to the adversary. Formally the notion is defined below in terms of two experiments Expt_0 and Expt_1 defined below.

$\mathsf{Expt}_0^{\mathcal{A},\mathsf{Ch}}(1^\lambda)$

- $\mathcal{A}$ outputs a query bound Q, maximum circuit size κ, total number of parties N, number of parties p participating in any session, threshold t.
- **Circuit Queries:** $\mathcal{A}$ is allowed to make circuit queries. First it makes Q_{pre} circuit queries $C_1, \ldots, C_{Q_{\mathsf{pre}}}$. For the i^{th} circuit query C_i, it sends the set $S_i \subseteq [\mathsf{N}]$, such that $|S_i| = \mathsf{p}$ andCh computes,
$$(\hat{C}_i^1, \ldots, \hat{C}_i^{\mathsf{N}}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C_i)$$
and sends this to $\mathcal{A}$.
- **Challenge Input Query:** $\mathcal{A}$ can choose to output a corruption set $\mathcal{S}_{\mathsf{corr}} \subseteq [\mathsf{N}]$ and input x.
If $Q_{\mathsf{pre}} > Q$ or $|\mathcal{S}_{\mathsf{corr}}| > \mathsf{t}$, then experiment aborts.
It outputs sets $S_{Q_{\mathsf{pre}}+1}, \ldots, S_Q \subseteq [\mathsf{N}]$ such that $|S_i| = \mathsf{p}$ for all $i \in [Q_{\mathsf{pre}}+1, Q]$. S_i is the set of parties participating in the i^{th} session.
Ch generates $(\hat{x}^1, \ldots, \hat{x}^{\mathsf{N}}) \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, 1^\kappa, x)$.
Ch sends
$$\left(\{\hat{x}^u\}_{u\in\mathcal{S}_{\mathsf{corr}}}, \left\{\mathsf{Local}(\hat{C}_i^u, \hat{x}_i^u)\right\}_{i\in[Q_{\mathsf{pre}}], u\in S_i}\right)$$
to $\mathcal{A}$ i.e. the inputs with the corrupted servers and the share of the honest servers due to the circuit queries made so far.
- **Circuit queries:** $\mathcal{A}$ then makes adaptive queries $Q_{\mathsf{pre}}+1, \ldots, Q$. Ch computes $(\hat{C}_i^1, \ldots, \hat{C}_i^{\mathsf{N}}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C_i)$ and sends,
$$\left\{\left((\hat{C}_i^1, \ldots, \hat{C}_i^{\mathsf{N}}), \mathsf{Local}(\hat{C}_i^u, \hat{x}^u)\right)\right\}_{i\in[Q_{\mathsf{pre}}+1,Q], u\in S_i}$$
to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b.

$\mathsf{Expt}_1^{\mathcal{A},\mathsf{Sim}(1^\lambda)}$

- $\mathcal{A}$ outputs a query bound Q, maximum circuit size κ, total number of parties N, number of parties p participating in any session, threshold t.
- Ch initializes the simulator Sim on input $(1^\lambda, 1^Q, 1^\kappa)$.
- **Circuit Queries:** $\mathcal{A}$ is allowed to make circuit queries. First it makes Q_{pre} circuit queries $C_1, \ldots, C_{Q_{\mathsf{pre}}}$. For the i^{th} circuit query C_i, it sends the set $S_i \subseteq [\mathsf{N}]$, such that $|S_i| = \mathsf{p}$ and Ch computes,
$$(\hat{C}_i^1, \ldots, \hat{C}_i^{\mathsf{N}}) \leftarrow \mathsf{Sim}(C_i)$$
and sends this to $\mathcal{A}$.

- **Challenge Input Query:** $\mathcal{A}$ can choose to output a corruption set $\mathcal{S}_{\mathsf{corr}} \subseteq [\mathsf{N}]$ and input x.
  If $Q_{\mathsf{pre}} > Q$ or $|\mathcal{S}_{\mathsf{corr}}| > \mathsf{t}$, then experiment aborts.
  It outputs sets $S_{Q_{\mathsf{pre}}+1}, \ldots, S_Q \subseteq [\mathsf{N}]$ such that $|S_i| = \mathsf{p}$ for all $i \in [Q_{\mathsf{pre}}+1, Q]$. S_i is the set of parties participating in the i^{th} session.
  Construct a set $\mathcal{V} = \{C_i, C_i(x) : i \in [Q_{\mathsf{pre}}]\}$. Ch runs the simulator to get the challenge output.
  $$\left(\{\hat{x}^u\}_{u\in\mathcal{S}_{\mathsf{corr}}}, \{\hat{y}_i^u\}_{i\in[Q_{\mathsf{pre}}],u\in S_i}\right)$$
  such that $\hat{y}_i^u = \mathsf{Local}(\hat{C}_i^u, \hat{x}^u)$ for every $u \in S_i \cap \mathcal{S}_{\mathsf{corr}}$. It sends this to $\mathcal{A}$.
- **Circuit queries:** $\mathcal{A}$ then makes adaptive queries $Q_{\mathsf{pre}}+1, \ldots, Q$. Ch computes for every $i \in [Q_{\mathsf{pre}}+1, Q]$,
  $$((\hat{C}_i^1, \ldots, \hat{C}_i^{\mathsf{N}}), \hat{y}_i^u) \leftarrow \mathsf{Sim}(i, C_i, C_i(x).$$
  It sends,
  $$\left\{((\hat{C}_i^1, \ldots, \hat{C}_i^{\mathsf{N}}), \hat{y}_i^u)\right\}_{i\in[Q_{\mathsf{pre}}+1,Q],u\in S_i}$$
  to $\mathcal{A}$.
- $\mathcal{A}$ outputs a bit b.

Definition A.1 (Security)**.** A protocol $\mathsf{Protocol}$ is secure if for every PPT adversary $\mathcal{A}$, there exists a PPT simulator Sim such that for some negligible function negl, for all $\lambda \in \mathbb{N}$,

$$\left|\Pr[1 \leftarrow \mathsf{Expt}_0^{\mathcal{A},\mathsf{Ch}}(1^\lambda)] - \Pr[1 \leftarrow \mathsf{Expt}_1^{\mathcal{A},\mathsf{Sim}}(1^\lambda)]\right| \leq \mathsf{negl}(\lambda).$$

Remark A.1. Note that the definition above differs from the security definition of the framework in [AV19] in the following two ways -

- The servers participating in the i^{th} session S_i can be output by the adversary on the i^{th} query rather than the start. Additionally, the corrupted set $\mathcal{S}_{\mathsf{corr}}$ is output during the challenge query by the adversary.

  Note that in the security proof for the client server framework in Section 5.2.2 of [AV19], the simulator uses S_i first during the i^{th} session and $\mathcal{S}_{\mathsf{corr}}$ during the challenge query and not before this. Thus the same security proof holds.

- The experiment allows more than Q circuit queries and can choose to output a challenge input query or not. If no challenge input query is made then Q_{pre} can be greater than Q in the real and ideal setting.

  This change doesn't effect the simulator as the simulator can run the real procedure during the pre-challenge phase, and since there is no challenge input query, there is no need for embedding the challenge in the different query responses.

We port the following theorem directly from [AV19].

Theorem A.1 ([AV19])**.** Let $Q \in \mathbb{N}$. Let $\mathsf{N} > \Theta(\lambda \cdot Q^2)$ and $\mathsf{t} < \lfloor \frac{n}{3} \rfloor$ and $\mathsf{p} < \mathsf{N}$. Assuming the existence of pseudorandom generators, there exists a secure protocol $\mathsf{Protocol}$ with parameters $\mathsf{t}, \mathsf{p}, \mathsf{N}$ for the circuit class $\mathsf{P/poly}$. Moreover, the construction makes black box use of the pseudorandom generators.

B Full Proof of Theorem 3.1

Proof. Consider the following simulator for construction of Section 3.2.1. The simulators jointly maintain an internal state. Since this simulator calls a couple instances of, Static-FE.Sim which is also stateful, consider this simulators to implicitly track the internal state of Static-FE.Sim in addition to the explicit state values.

$\mathsf{S}_0(1^\lambda, 1^n)$ - The simulator initializes λ independent instances of the $\{\mathsf{Static\text{-}FE.Sim}^i\}_{i\in[\lambda]}$ and runs all the $\mathsf{Static\text{-}FE.S}_0^i$ simulators with increasing values of the static collusion bound q as follows:

$$\forall i \in [\lambda], \quad \mathsf{mpk}_i \leftarrow \mathsf{Static\text{-}FE.S}_0^i(1^\lambda, 1^n, q = 2^i).$$

It then outputs the public keys as an λ-tuple of all these keys, i.e. $\mathsf{mpk} = (\mathsf{mpk}_i)_{i\in[\lambda]}$.

$\mathsf{S}_1(f)$ - The simulator runs all λ independent instances of the $\mathsf{Static\text{-}FE.S}_1^i$ key generation simulator as $\mathsf{sk}_{i,f} \leftarrow \mathsf{Static\text{-}FE.S}_1^i(f)$ for $i \in [\lambda]$. It outputs the secret key sk as $\mathsf{sk} = (\mathsf{sk}_{i,f})_{i\in[\lambda]}$.

$\mathsf{S}_2(\Pi^m, 1^Q)$ - The encryption algorithm simply simulates the ciphertext using the $\lceil \log Q \rceil$-th Static-FE simulator as $\mathsf{ct} \leftarrow \mathsf{Static\text{-}FE.S}_2^{\lceil \log Q \rceil}(\Pi^m)$. (It also includes Q as part of the ciphertext, and tracks it in it's internal state.)

$\mathsf{S}_3^{U_m(\cdot)}(f)$ - For all $i \in [\lambda] \backslash \{\lceil \log Q \rceil\}$, the simulator proceeds excatly as in S_1, generating $\mathsf{sk}_{i,f} \leftarrow \mathsf{Static\text{-}FE.S}_1^i(f)$. For $\mathsf{sk}_{\lceil \log Q \rceil, f}$, this key component is generated as the output of $\mathsf{Static\text{-}FE.S}_3^{\lceil \log Q \rceil}(f)$. The secret key is output as $\mathsf{sk} = (\mathsf{sk}_{i,f})_{i\in[\lambda]}$.

We can show that the adversary cannot distinguish between interacting with the above simulator and the real scheme through a series of experiments.

Experiment j for $j \in [0, \lambda]$.

- **Setup Phase:** Adversary sends (1^n). For $i \in [j+1, \lambda]$, generate $\mathsf{mpk}_i \leftarrow \mathsf{S\text{-}FE.Setup}(1^\lambda, 1^n, q = 2^i)$. For $i \in [j]$, initializes an instance of the weakly optimal simulator $\mathsf{Static\text{-}FE.Sim}^i$ and generate $\mathsf{mpk}_i \leftarrow \mathsf{Static\text{-}FE.S}_0^i(1^\lambda, 1^n, q = 2^i)$. Return $\{\mathsf{mpk}_i\}_{i\in[Q]}$ to the adversary.
- **Pre-Challenge Key Queries:** Adversary sends Keygen queries of a function f. For $i \in [j+1, \lambda]$, generate $\mathsf{sk}_{i,f} \leftarrow \mathsf{S\text{-}FE.KeyGen}(\mathsf{msk}_i, f)$. For $i \in [j]$, instead run the corresponding weakly optimal simulator $\mathsf{Static\text{-}FE.Sim}^i$ and generate $\mathsf{mpk}_i \leftarrow \mathsf{Static\text{-}FE.S}_1^i(f)$. Return $\{\mathsf{sk}_{i,f}\}_{i\in[Q]}$ to the adversary.
- **Challenge Ciphertext:** Adversary sends message m and collusion bound 1^Q. If $\lceil \log Q \rceil > j$, generate ct as $\mathsf{S\text{-}FE.Enc}(\mathsf{mpk}_{\lceil \log Q \rceil}, m)$. Otherwise, set ct as $\mathsf{Static\text{-}FE.S}_2^{\lceil \log Q \rceil}(\Pi^m, 1^Q)$. Return (ct, Q) to adversary.
- **Post-Challenge Key Queries:** Adversary sends Keygen queries of a function f. For $i \in [j+1, Q]$, generate $\mathsf{sk}_{i,f} \leftarrow \mathsf{S\text{-}FE.KeyGen}(\mathsf{msk}_i, f)$. For $i \in [j]$, instead run the corresponding weakly optimal simulator $\mathsf{Static\text{-}FE.Sim}^i$. If $i = \lceil \log Q \rceil$, generate $\mathsf{mpk}_i \leftarrow \mathsf{Static\text{-}FE.S}_3^i(f)$. Otherwise, generate $\mathsf{mpk}_i \leftarrow \mathsf{Static\text{-}FE.S}_1^i(f)$. Return $\{\mathsf{sk}_{i,f}\}_{i\in[Q]}$ to the adversary.

- **Output:** Adversary outputs a bit $\in \{0,1\}$

Lemma B.1. Assuming Static-FE is a weakly-optimal static-bounded-collusion simulation-secure FE scheme, for every PPT adversary $\mathcal{A}$, every $j \in [\lambda]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{j-1}(\lambda) - \mathcal{P}_{\mathcal{A}}^{j}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. Observe that the only difference between experiments $j-1$ and j is how mpk_j (and corresponding secret keys and ciphertexts) are replaced with their simulated counterparts. Suppose $\mathcal{A}$ is a distinguisher between experiments $j-1$ and j which succeeds with noticeable probability. Then the following $\mathcal{A}'$ is a reduction which breaks strong simulation-security of Static-FE with noticeable probability.

$\mathcal{A}'(1^\lambda)$

- **Setup:** Run $\mathcal{A}(1^\lambda)$ and receive 1^n. Forward $(1^n, 1^{2^j})$ to your challenger and receive challenge public key mpk^*. Generate mpk via experiment $j-1$, except set $\mathsf{mpk}_j = \mathsf{mpk}^*$, and return to $\mathcal{A}$
- **Pre-Challenge Key Queries:** $\mathcal{A}$ sends Keygen queries of a function f. Query your challenger's KeyGen oracle on f and receive secret key sk_f^*. Generate sk_f via experiment $j-1$, but set $\mathsf{sk}_{j,f} = \mathsf{sk}_f^*$ and return to $\mathcal{A}$.
- **Ciphertext Queries:** Adversary sends message m and collusion bound 1^Q. If $j \neq \lceil \log Q \rceil$, generate ct as in experiment $j-1$. Otherwise, query for a challenge ciphertext on message m nad collusion bound Q, receiving ct^*, setting $\mathsf{ct} = (\mathsf{ct}^*, Q)$ and returning to $\mathcal{A}$.
- **Post-Challenge Key Queries:** Same as pre-challenge queries.
- $\mathcal{A}$ returns bit b, Output b.

Observe that when the challenge keys/ciphertexts are generated by the real scheme, this is exactly experiment $j-1$, and when they are generated by the simulated scheme, this is experiment j. Thus, all that remains to be checked is that the queries $\mathcal{A}'$ sends satisfy the admissibility constraints of strong simulation security.

First, we can see if $j \neq \lceil \log Q \rceil$, the challenge ciphertext is never requested, so by Definition 3.3, the number of key queries can be unbounded. Alternatively, if $j = \lceil \log Q \rceil$, recall we set the static collusion bound to $2^j = 2^{\lceil \log Q \rceil} \geq Q$. Since the $\mathcal{A}'$ makes at most Q queries to the key generation oracle, this case is admissible as well.

□

Note that experiment 0 is identical to the experiment played against the real scheme, while experiment λ is exactly the simulated experiment. Thus, we can conclude the two are computationally indistinguishable.

□

C Full proof of Theorem 6.1

We show this through a sequence of experiments, we first show that the scheme is simulation-secure.

Experiment 0. This is the experiment with adversary $\mathcal{A}$ and simulation secure tagged FE scheme. The experiment is parameterized by $\lambda \in \mathbb{N}$.

$$\left\{ \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot,\cdot),\mathsf{Enc}(\mathsf{mpk},\cdot,\cdot)}(\mathsf{mpk}) : \begin{array}{c} (1^n, 1^Q, 1^z) \leftarrow \mathcal{A}(1^\lambda) \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^z, 1^q) \end{array} \right\}_{\lambda \in \mathbb{N}}$$

- **Setup:** $(1^n, 1^Q, 1^z) \leftarrow \mathcal{A}(1^\lambda)$. $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{1TgFE.Setup}(1^\lambda, 1^n, 1^{z+\lceil \log \mathsf{N} \rceil}, 1)$. Send mpk to $\mathcal{A}$. $\mathcal{A}$ is given access to two oracles, KeyGen and Enc.
- **Key Queries:** Let $\mathcal{A}$ query KeyGen on tag $\mathsf{tg} \in \mathcal{I}_z$ and circuit C.
  1. Let this be the j-th query to the oracle and the input query be denoted by $\mathsf{tg}_j \in \mathcal{I}_z, C_j$. Let this be the i-th query to tag tg_j.
  2. Sample a set $S_i^{\mathsf{tg}_j} \leftarrow [\mathsf{N}]$ of cardinality p.
  3. Compute $(\hat{C}_j^1, \ldots, \hat{C}_j^{\mathsf{N}}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C_j)$. Let $\mathsf{E}_j^u(\cdot) = \mathsf{Local}(\hat{C}_j^u, \cdot)$.
  4. Generate a key for circuit E_j^u under the one tagged scheme, i.e. $\forall u \in S_i^{\mathsf{tg}_j}$, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.KeyGen}(\mathsf{msk}, (u, \mathsf{tg}_j), \mathsf{E}_j^u)$. Output $(\{(u, \mathsf{sk}_{\mathsf{E}_j^u})\}_{u \in S_i^{\mathsf{tg}_j}}, S_i^{\mathsf{tg}_j})$.
- **Ciphertext Queries:** Let $\mathcal{A}$ query Enc on tag $\mathsf{tg}^* \in \mathcal{I}_z$ and message $m^{\mathsf{tg}^*} \in \mathcal{M}_n$.
  1. Let $Q_{\mathsf{pre}}^{\mathsf{tg}^*}$ be the number of key generation queries made to tag tg^*. Let the set $\Pi^{m^{\mathsf{tg}^*}}$ denote the set of all $(C_j, C_j(m^{\mathsf{tg}^*}))$ where for $j \in [Q]$, (tg^*, C_j) was the key generation query.
  2. Compute $(\hat{x}^1, \ldots, \hat{x}^{\mathsf{N}}) \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, 1^\kappa, m^{\mathsf{tg}^*})$.
  3. For every $u \in [\mathsf{N}]$, compute $\mathsf{ct}_u \leftarrow \mathsf{1TgFE.Enc}(\mathsf{mpk}, (u, \mathsf{tg}^*), \hat{x}^u)$. Output $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{\mathsf{N}})$.
- $\mathcal{A}$ outputs a bit b.

We've included additional notational indexing to help with our hybrids.

Experiment 1. In this experiment, we pre sample the post challenge sets $S_i^{\mathsf{tg}^*}$ for queries that come after the challenge tag tg^*. The sampled sets are then used to answer post challenge key generation queries.

- **Key Queries:** Let $\mathcal{A}$ query KeyGen on tag $\mathsf{tg}_j \in \mathcal{I}_z$ and circuit C_j.
  1. **Pre Challenge Key Queries:** If tg_j was not queried by $\mathcal{A}$ to S_2, perform computation exactly as in previous experiment.
  2. **Post Challenge Key Queries:** Else, use $S_i^{\mathsf{tg}_j}$ as sampled in the challenge phase.
- **Ciphertext Queries:** Let $\mathcal{A}$ query Enc on tag $\mathsf{tg}^* \in \mathcal{I}_z$ and message $m^{\mathsf{tg}^*} \in \mathcal{M}_n$.
  1. For every $i \in [Q_{\mathsf{pre}}^{\mathsf{tg}^*} + 1, Q]$, sample a set $S_i^{\mathsf{tg}^*}$ of size p uniformly at random from $[\mathsf{N}]$.
  2. Compute ct exactly as same as previous experiment.

Experiment 2. In this experiment, we output $\perp$ if the number of queries on a challenge tag tg^*, we have that the number of corrupted servers or the set $\mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$ which denotes pairwise intersection of $\{S_i^{\mathsf{tg}^*}\}_{i\in[Q]}$ exceeds some threshold t.

- **Ciphertext Queries:** Let $\mathcal{A}$ query Enc on tag $\mathsf{tg}^* \in \mathcal{I}_z$ and message $m^{\mathsf{tg}^*} \in \mathcal{M}_n$.
  1. For every $i \in [Q^{\mathsf{tg}^*}_{\mathsf{pre}} + 1, Q]$, sample a set $S_i^{\mathsf{tg}^*}$ of size p uniformly at random from $[\mathsf{N}]$.
  2. Construct a corrupted set $\mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$ that includes all tags that are corruped as follows,
  $$\mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}} = \{u \in [\mathsf{N}] \ : \ \exists i, j \in [Q], i \neq j \text{ and } u \in S_i^{\mathsf{tg}^*} \cup S_j^{\mathsf{tg}^*}\}.$$
  If $|\mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}| > \mathsf{t}$, output $\perp$.
  3. Compute ct exactly as same as previous experiment.
- $\mathcal{A}$ outputs a bit b.

Experiment 3. We rely on $\mathsf{1TgFE}$ security to simulate $\mathsf{1TgFE}$.

- **Setup:** $(1^n, 1^Q, 1^z) \leftarrow \mathcal{A}(1^\lambda)$. Let $\mathsf{mpk} \leftarrow \mathsf{1TgFE.S}_0(1^\lambda, 1^n, 1^{z+\lceil \log \mathsf{N} \rceil}, 1)$. Send mpk to $\mathcal{A}$.
- **Key Queries:** Let $\mathcal{A}$ query KeyGen on tag $\mathsf{tg}_j \in \mathcal{I}_z$ and circuit C_j.
  1. **Pre Challenge Key Queries:**
     (a) Compute $(\hat{C}_j^1, \ldots, \hat{C}_j^{\mathsf{N}}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C_j)$. Let $\mathsf{E}_j^u(\cdot) = \mathsf{Local}(\hat{C}_j^u, \cdot)$.
     (b) Sample a set $S_i^{\mathsf{tg}_j} \leftarrow [\mathsf{N}]$ of cardinality p.
     (c) Generate a key for circuit E_j^u under the one tagged scheme, $\forall u \in S_i^{\mathsf{tg}_j}$, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u, \mathsf{tg}_j), \mathsf{E}_j^u, \perp)$. Output $(\{(u, \mathsf{sk}_{\mathsf{E}_j^u})\}_{u \in S_i^{\mathsf{tg}_j}}, S_i^{\mathsf{tg}_j})$.
  2. **Post Challenge Key Queries:**
     (a) Compute $(\hat{C}_j^1, \ldots, \hat{C}_j^{\mathsf{N}}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, 1^\kappa, C_j)$. Let $\mathsf{E}_j^u(\cdot) = \mathsf{Local}(\hat{C}_j^u, \cdot)$ for $u \in [\mathsf{N}]$.
     (b) Generate a key for circuit E_j^u under the one tagged scheme, i.e. $\forall u \in S^{\mathsf{tg}_j}$,
        i. If $u \in \mathcal{S}^{\mathsf{tg}_j}_{\mathsf{corr}}$, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u, \mathsf{tg}_j), \mathsf{E}_j^u, \perp)$.[9]
        ii. Else, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u, \mathsf{tg}_j), \mathsf{E}_j^u, \hat{y}_i^u)$ where $\hat{y}_i^u = \mathsf{Local}(\hat{C}_j^u, \hat{x}^u)$ ($\hat{x}^u$ is known from the challenge message to S_2).
     Output $(\{(u, \mathsf{sk}_{\mathsf{E}_j^u})\}_{u \in S_i^{\mathsf{tg}_j}}, S_i^{\mathsf{tg}_j})$.
- **Ciphertext Queries:**
  1. Perform the following change while computing ct. For every $u \in [\mathsf{N}]$,
     (a) If $u \in \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$ compute $\mathsf{ct}_u \leftarrow \mathsf{1TgFE.Enc}(\mathsf{mpk}, (u, \mathsf{tg}), \hat{x}^u)$.

[9] Note that on these tags, no ciphertext query is made.

(b) If $u \notin \mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}$ let the shares previously computed be $\Pi'_u = \{(\mathsf{E}_j^u, \hat{y}_i^u)\}_{i\in[Q_{\mathsf{pre}}^{\mathsf{tg}^*}],u\in[\mathsf{N}]}$ (where j is the query number to S_1 and it is the i-th query on tg_j) and compute, $\mathsf{ct}_u \leftarrow \mathsf{1TgFE.S}_2((u,\mathsf{tg}),\Pi'_u)$.

Output $(\mathsf{ct}_1,\ldots,\mathsf{ct}_\mathsf{N})$.

- $\mathcal{A}$ outputs a bit b.

Experiment 4_k**.** Let each tg that is queried by adversary to both KeyGen and Enc oracle be recorded in the set tgset. Let $|\mathsf{tgset}| = \mathsf{TotalQtg}$ be the total number of unique tags queried to the oracles. Let tgset have a well defined ordering on the tags that is defined by the time the query was received by S_1 or S_2. Let tgset_k denote the k-th tag in this set. Thus $\mathsf{tgset}_1 > \mathsf{tgset}_2$ if tgset_1 was received later than tgset_2.

In this experiment, for $k \in [\mathsf{TotalQtg}+1]$, we will switch $\mathsf{tgset}_1,\ldots,\mathsf{tgset}_{k-1}$ to simulate on the client server framework. In the end, we remove all information about m^{tg^*} for every challenge tag.

- $\mathcal{A}$ is given access to two oracles, KeyGen and Enc. Let tgset be a set that keeps track of the unique tags that it sees.
- **Key Queries:**
  1. Let this be the j-th query to the oracle and the input query be denoted by $\mathsf{tg}_j \in \mathcal{I}_z, C_j$. Let this be the i-th query to tag tg_j.
  2. If tg_j has not been queried before, add it to tgset. . If $\mathsf{tg}_j \geq \mathsf{tgset}_k$ (statement is false if $|\mathsf{tgset}| < k$), then proceed exactly as in the previous experiment. Otherwise continue below.
  3. If $i = 1$, initialize the simulator $\mathsf{Protocol.Sim}_{\mathsf{tg}_j}$ on input $(1^\lambda, 1^Q, 1^\kappa)$.
  4. **Pre Challenge Key Queries:**
     (a) Compute $(\hat{C}_j^1,\ldots,\hat{C}_j^\mathsf{N}) \leftarrow \mathsf{Protocol.Sim}_{\mathsf{tg}_j}(C_j)$. . Let $\mathsf{E}_j^u(\cdot) = \mathsf{Local}(\hat{C}_j^u,\cdot)$.
     (b) Sample a set $S_i^{\mathsf{tg}_j} \leftarrow [\mathsf{N}]$ of cardinality p.
     (c) Generate a key for circuit E_j^u under the one tagged scheme, $\forall u \in S_i^{\mathsf{tg}_j}$, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u,\mathsf{tg}_j),\mathsf{E}_j^u,\perp)$. Output $(\{(u,\mathsf{sk}_{\mathsf{E}_j^u})\}_{u\in S_i^{\mathsf{tg}_j}}, S_i^{\mathsf{tg}_j})$.
  5. **Post Challenge Key Queries:**
     (a) Compute the simulated circuit, $(\hat{C}_j^1,\ldots,\hat{C}_j^\mathsf{N})$ and $\{\hat{y}_i^u\}_{u\in S_i^{\mathsf{tg}_j}}$ by running $\mathsf{Protocol.Sim}_{\mathsf{tg}_j}$ on $(i, C_j, C_j(m^{\mathsf{tg}_j}))$. Note we can are allowed to compute $C_j(m^{\mathsf{tg}_j})$. . Let $\mathsf{E}_j^u(\cdot) = \mathsf{Local}(\hat{C}_j^u,\cdot)$ for $u \in [\mathsf{N}]$.
     (b) Generate a key for circuit E_j^u under the one tagged scheme, i.e. $\forall u \in S^{\mathsf{tg}_j}$,
        i. If $u \in \mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}_j}$, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u,\mathsf{tg}_j),\mathsf{E}_j^u,\perp)$.[10]
        ii. Else, $\mathsf{sk}_{\mathsf{E}_j^u} \leftarrow \mathsf{1TgFE.S}_1((u,\mathsf{tg}_j),\mathsf{E}_j^u,\hat{y}_i^u)$ ~~where $\hat{y}_i^u = \mathsf{Local}(\hat{C}_j^u,\hat{x}^u)$ ($\hat{x}^u$ is known from the challenge message to S_2)~~.

     Output $(\{(u,\mathsf{sk}_{\mathsf{E}_j^u})\}_{u\in S_i^{\mathsf{tg}_j}}, S_i^{\mathsf{tg}_j})$.

[10] Note that on these tags, no ciphertext query is made.

- **Ciphertext Queries:** Let $\mathcal{A}$ query Enc on tag $\mathsf{tg}^* \in \mathcal{I}_z$ and message $m^{\mathsf{tg}^*} \in \mathcal{M}_n$.
  1. Let the variables and computation on $\mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$ be defined similarly as in the previous experiment.
  2. If tg^* has not been queried before, add it to tgset.
  3. If $\mathsf{tg}^* \geq \mathsf{tgset}_k$ (statement is false if $|\mathsf{tgset}| < k$), then do exactly as the previous experiment. Otherwise proceed below.
  4. Compute $\mathsf{Protocol.Sim}_{\mathsf{tg}^*}(1^{|m^{\mathsf{tg}^*}|}, \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}, \Pi^{m^{\mathsf{tg}^*}})$ [11] to obtain $\{\hat{x}^u\}_{u \in \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}}$ and evalutations $\{\hat{y}^u_i\}_{i \in [Q^{\mathsf{tg}^*}_{\mathsf{pre}}], u \in S^{\mathsf{tg}^*}_i}$. Additionally we have that $\hat{y}^u_i = \mathsf{Local}(\hat{C}^u_j, \hat{x}^u)$ for $u \in S^{\mathsf{tg}^*}_i \cap \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$ where $i \in [Q^{\mathsf{tg}^*}_{\mathsf{pre}}]$ and j is the global query number (recall i is the query number on tg^*). Note that we have completely removed the dependence on $\hat{x}^u$ for $u \notin \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$.
  5. Proceed similarly to the previous experiment i.e. for every $u \in [\mathsf{N}]$,
     (a) If $u \in \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$ compute
     $$\mathsf{ct}_u \leftarrow \mathsf{1TgFE.Enc}(\mathsf{mpk}, (u, \mathsf{tg}), \hat{x}^u).$$
     (b) If $u \notin \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$ let the shares we've previously output be noted down in $\Pi'_u = \{(\mathsf{E}^u_j, \hat{y}^u_i)\}_{i \in [Q^{\mathsf{tg}^*}_{\mathsf{pre}}], u \in [\mathsf{N}]}$ (j is the total query number to S_1 where this is the i-th query on tg_j) and compute,
     $$\mathsf{ct}_u \leftarrow \mathsf{1TgFE.S}_2((u, \mathsf{tg}), \Pi'_u).$$
  6. Output $(\mathsf{ct}_1, \ldots, \mathsf{ct}_{\mathsf{N}})$.
- $\mathcal{A}$ outputs a bit b.

Analysis. Let $\mathcal{P}^i_{\mathcal{A}}(\lambda)$ be the probability that adversary $\mathcal{A}$ outputs 1 on Experiment i run on security parameter λ.

Lemma C.1. For every adversary $\mathcal{A}$, $\lambda \in \mathbb{N}$, $\mathcal{P}^0_{\mathcal{A}}(\lambda) = \mathcal{P}^1_{\mathcal{A}}(\lambda)$.

Proof. The only changes in the two experiments are syntactical, where post challenge key generation sets $\mathcal{S}^{\mathsf{tg}_j}_{\mathsf{corr}}$ are sampled during the challenge phase. Since the distribution is identical in both experiments, the probability of $\mathcal{A}$ outputting 1 is identical. □

Lemma C.2. For every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^1_{\mathcal{A}}(\lambda) - \mathcal{P}^2_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. The statistical distance between the two experiments is negligible in security parameter λ. We use the following lemma proven in [GVW12].

Lemma C.3 (Small Pairwise Intersection Lemma [GVW12, AV19])**.** Let $Q \in \mathbb{N}$, set $\mathsf{t} = \Theta(Q^2\lambda), \mathsf{N} = \Theta(Q^2\mathsf{t}^2)$ and $\mathsf{p} = \Theta(\mathsf{t})$. Then for some negligible function λ,
$$\Pr\left[\left|\cup_{i_1 \neq i_2} (S_{i_1} \cap S_{i_2})\right| > \mathsf{t}\right] \leq \mathsf{negl}(\lambda).$$

[11] We can assume simulator knows the message length as we have the empty function description, see Footnote 6.

In Experiment 2, we denote this set by $\mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$ for challenge tag tg^* and output $\perp$ for every challenge tag. Since the number of encryption challenge queries are polynomial in λ, Q, from union bounding the probability of outputting bot is still $\mathsf{negl}(\lambda)$. □

Lemma C.4. If $\mathsf{1TgFE} = (\mathsf{1TgFE.Setup}, \mathsf{1TgFE.Enc}, \mathsf{1TgFE.KeyGen}, \mathsf{1TgFE.Dec})$ is a simulation-secure tagged FE scheme for collusion bound 1 (as per Definition 4.1), then for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^2_{\mathcal{A}}(\lambda) - \mathcal{P}^3_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. The difference in these two experiments is that we use $\mathsf{1TgFE.Sim}$ during Setup, KeyGen and Enc to simulate. Note that the tag space for $\mathsf{1TgFE}$ is $1^{z+\lceil \log \mathsf{N} \rceil}$ and for a challenge identity $\mathsf{tg}^* \in \mathcal{I}_z$, we only simulate on $u \notin \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$, thus we are not restricted to making only one query to $\mathsf{1TgFE.S}_1$ for $u \in \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$. Thus this is exactly the $\mathsf{1TgFE}$ security experiment and we can construct a reduction $\mathcal{B}$ that proceeds as follows. The reduction $\mathcal{B}$ coordinates with an environment $\mathcal{E}$ as defined in Definition 4.1. The environment simply ensures that relevant state information exists so the simulator can run. We don't explicitly show the environment transcript below for simplicity.

- **Setup:** It gets $(1^n, 1^Q, \mathcal{I}_z)$ from $\mathcal{A}$. $\mathcal{B}$ outputs $(1^n, 1, 1^{z+\lceil \log \mathsf{N} \rceil})$. It sets mpk from the experiment and sends it to $\mathcal{A}$.
- **Key Queries:** Let $\mathcal{A}$ query KeyGen on tag $\mathsf{tg}_j \in \mathcal{I}_z$ and circuit C_j. Run as in Experiment 2, but instead replace calls to $\mathsf{1TgFE.KeyGen}$ are replaced with calls to challenge KeyGen oracle.
- **Ciphertext Queries:** Let $\mathcal{A}$ query Enc on tag $\mathsf{tg}^* \in \mathcal{I}_z$ and message $m^{\mathsf{tg}^*} \in \mathcal{M}_n$. Run as in Experiment 2, but replace calls to $\mathsf{1TgFE.Enc}$ with calls to the Enc oracle when the index $u \notin \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$.

We make the following observations on a tag tg^* that was queried to S_2 by $\mathcal{A}$.

- For $u \in \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$, we never query $\mathsf{1TgFE.S}_2$ on tag (u, tg^*).
- For $u \notin \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$, we only query $\mathsf{1TgFE.S}_2$ on tag (u, tg^*) on message $\hat{x}^u$ once.
- We make at most 1 query to $\mathsf{1TgFE.S}_1$ for all $u \notin \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$.
- Since the $\mathsf{1TgFE}$ experiment does not place restrictions on the number of queries $\mathcal{B}$ can make on non-challenge tags, for $u \in \mathcal{S}^{\mathsf{tg}^*}_{\mathsf{corr}}$, $\mathcal{B}$ is allowed to make more than one query on tag (u, tg^*).
- Note that $\hat{y}^u_i = \mathsf{F}^u_j(\hat{x}^u)$ where $\hat{x}^u$ was the challenge message on tag (u, tg^*) output by the simulator. Thus these are valid settings for calling $\mathsf{1TgFE}$ simulator.

If $\mathcal{A}$ distinguishes between Experiments 2 and 3 with non-negligible probability, then $\mathcal{B}$ is a valid algorithm that distinguishes between Experiments in Definition 4.1 with non-negligible probability. Hence the advantage of $\mathcal{A}$ is negligible. □

Lemma C.5. If $\mathsf{Protocol}$ is a secure client server framework according to Definition A.1 then for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^{4_k}_{\mathcal{A}}(\lambda) - \mathcal{P}^{4_{k+1}}_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. The difference in these two experiments is that in the latter experiment we use $\mathsf{Protocol.Sim}_{\mathsf{tgset}_k}$ during KeyGen and Enc to simulate. On every query to the $\mathsf{KeyGen}, \mathsf{Enc}$ oracles on tgset_k, instead o using algorithms $\mathsf{CktEnc}, \mathsf{InpEnc}$, we simulate the circuit queries and the challenge input query. This is exactly the $\mathsf{Protocol}$ security experiment on servers denoted by $S_1^{\mathsf{tgset}_k}, \ldots, S_Q^{\mathsf{tgset}_k}$, corrupted set $\mathcal{S}_{\mathsf{corr}}^{\mathsf{tgset}_k}$ and input query m^{tgset_k}. We construct a reduction $\mathcal{B}_k$ that proceeds as follows.

- **Setup:** It gets $(1^n, 1^Q, \mathcal{I}_z)$ from $\mathcal{A}(1^\lambda)$. $\mathcal{B}_k$ outputs a query bound Q, maximum circuit size κ [12], total number of parties N, number of parties p participating in any session, threshold t. Let $\mathsf{mpk} \leftarrow \mathsf{1TgFE.S}_0(1^\lambda, 1^n, 1^{z+\lceil \log \mathsf{N} \rceil}, 1)$. Send mpk to $\mathcal{A}$.
- **Key Queries:** Let $\mathcal{A}$ query KeyGen on tag $\mathsf{tg}_j \in \mathcal{I}_z$ and circuit C_j. Run as in Experiment 4^k, but on $\mathsf{tg}_j = \mathsf{tgset}_k$, generate $(\hat{C}_i^1, \ldots, \hat{C}_i^{\mathsf{N}})$ from the $\mathsf{Protocol}$ challenger rather than CktEnc. Additionally, for the post challenge phase, the $\mathsf{Protocol}$ simulator outputs $\{\hat{y}_i^u\}_{u \in S_i^{\mathsf{tg}_j}}$ which will be used to simulate $\mathsf{1TgFE}$ simulator.
- **Ciphertext Queries:** Let $\mathcal{A}$ query Enc on tag $\mathsf{tg}^* \in \mathcal{I}_z$ and message $m^{\mathsf{tg}^*} \in \mathcal{M}_n$. Run as in Experiment 4^k, but on $\mathsf{tg}^* = \mathsf{tgset}_k$, obtain $\{\hat{x}^u\}_{u \in \mathcal{S}_{\mathsf{corr}}^{\mathsf{tg}^*}}$ and evalutations $\{\hat{y}_i^u\}_{i \in [Q_{\mathsf{pre}}^{\mathsf{tg}^*}], u \in S_i^{\mathsf{tg}^*}}$ from $\mathsf{Protocol}$ experiment.
- $\mathcal{B}_k$ gets bit b from $\mathcal{A}$ and outputs it.

We make the following observations on a tag tgset_k.

- If there are no S_2 queries on tgset_k, then the experiment can make $Q_{\mathsf{pre}}^{\mathsf{tgset}_k}$ queries which may be more than Q.
- If there are queries on S_2, then we ensure from the security experiment that $\mathcal{A}$ does not make more than Q queries on tgset_k.

If $\mathcal{A}$ distinguishes between Experiments 4_k and 4_{k+1} with non-negligible probability, then $\mathcal{B}_k$ is a valid algorithm that distinguishes between Experiments in Definition A.1 with non-negligible probability. Hence the advantage of $\mathcal{A}$ is negligible. Since $\mathsf{TotalQtg}$ is polynomial in λ, we have that the advantage between Experiments 4_1 and Experiments $4_{\mathsf{TotalQtg}+1}$ is negligible.

□

It is easy to see that Experiment 4_1 is exactly the same as Experiment 3 and Experiment $4_{\mathsf{TotalQtg}+1}$ is exactly the same as interacting with simulators $\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2$ (barring some syntactical changes of keeping the appropriate state recorded). Thus the experiments in Definition 4.1 for collusion bound Q are computationally indistinguishable experiments.

D Full proof of Theorem 7.1

We will show through a sequence of experiments that the real scheme and simulated scheme are computationally indistinguishable.

[12] The maximum size of family of circuits $\mathcal{F}_n$.

Experiment 0. This is the real non-adaptive security Experiment played between adversary $\mathcal{A}$ and challenger $\mathcal{C}$.

- **Setup:** Adversary generates $(1^n, 1^q, 1^z)$. Compute $(\mathsf{ibe.pk}, \mathsf{ibe.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda, 1^{z+\lceil \log n' \rceil + 1})$, giving $\mathsf{ibe.pk}$ to adversary and retaining $\mathsf{ibe.msk}$ in state.
- **Key Queries:** Adversary sends Keygen queries of the form (tg, C). For $i \in [n]$, set $\mathsf{sk}_i = \mathsf{IBE.KeyGen}(\mathsf{msk}, (C[i], i, \mathsf{tg}))$, and return $\{\mathsf{sk}_i\}_{i\in[n]}$ to adversary
- **Ciphertext Queries:** Adversary sends message m and tag tg^*. Compute a garbled circuit $(\hat{\mathcal{U}}, \{w_{i,b}\}_{i\leq n, b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathcal{U})$[13]. For all $(i, b) \in [n] \times \{0, 1\}$, set $\mathsf{ct}_{i,b} \leftarrow \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b, i, \mathsf{tg}^*), w_{i,b})$. Return $\mathsf{ct} = (\hat{\mathcal{U}}, \{\mathsf{ct}_{i,b}\}_{i\leq n, b\in\{0,1\}})$ to adversary.
- **Output:** Adversary outputs a bit $\in \{0, 1\}$

Experiment 1. In this experiment, we use the security of IBE to replace ciphertexts for keys which are not given. $\overline{b}$ is used to indicate the flip bit of b.

- **Ciphertext Queries:** The ciphertext components $\mathsf{ct}_{i,b}$ are generated differently:
  Let (tg^*, C^*) be the pre-challenge query made on tag tg^*. For all $i \in [n]$, set $\mathsf{ct}_{i,\overline{C_i^*}} \leftarrow \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (\overline{C_i^*}, i, \mathsf{tg}^*), 0^{|w_i|})$, an encryption of zero (all other encryptions are computed as before).

Experiment 2. In this experiment, we simulate the garbled circuit rather than using it directly. By this point, the ciphertext queries are generated from $(C, C(m))$ and can be simulated.

- **Ciphertext Queries:** Compute a garbled circuit $(\hat{\mathcal{U}}, \{w_{i,C_i}\}_{i\leq n}) \leftarrow \mathsf{GC.Sim}(1^\lambda, \mathcal{U}, C(m))$

Analysis. Let $\mathcal{P}^i_{\mathcal{A}}(\lambda)$ be the probability an adversary $\mathcal{A}$ returns 1 on experiment i.

Lemma D.1. Assuming ibe is a secure IBE scheme for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^0_{\mathcal{A}}(\lambda) - \mathcal{P}^1_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. We will proceed through a sequence of subhybrids where we incrementally replace portions of the challenge ciphertexts with encryptions of 0. Let $(\{(\mathsf{tg}^*_j, C^{*j})\}_{j\in N})$ be a list of queries made by $\mathcal{A}$ to Enc. Define Experiment $0^{i^*,j^*}$ to be the following:

- **Setup, Key Queries:** Are played as Experiment 0
- **Ciphertext Queries:** Adversary sends message m_j and tag tg^*_j - For all ciphertext components where $j > j^*$ or $j = j^* \wedge i \geq i^*$, the components $\mathsf{ct}_{i,\overline{C_i^{*j}}}$ are generated as Experiment 0. Otherwise, said ciphertext components are replaced with an encryption of 0.
- **Output:** Played as Experiment 0

Claim D.1. For every adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^0_{\mathcal{A}}(\lambda) - \mathcal{P}^{0^{1,1}}_{\mathcal{A}}(\lambda)| = 0$

[13] The structure of this universal circuit U is the same as defined in algorithm description.

Proof. Since i and j are at least 1, the conditional never occurs, and this is Experiment 0. □

Claim D.2. Assuming ibe is a secure IBE scheme, for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{0^{i^*,j^*}}(\lambda) - \mathcal{P}_{\mathcal{A}}^{0^{i^*+1,j^*}}(\lambda)| = \mathsf{negl}(\lambda)$

Proof. Notice that Experiments $0^{i^*,j^*}$ and $0^{i^*+1,j^*}$ only differ in how $\mathsf{ct}_{i,\overline{C_i^{*j}}}$ is generated. Suppose $\mathcal{A}$ is a distinguisher between Experiments $0^{i^*,j^*}$ and $0^{i^*+1,j^*}$ which succeeds with noticeable probability. Then we can break IBE security with the same probability as follows:

$\mathcal{A}'(1^\lambda)$

- **Setup:** Receive $\mathsf{ibe.mpk}$ from IBE challenger and send to $\mathcal{A}$
- **Key Queries:** Answer the key queries of $\mathcal{A}$ by querying the IBE challenger for secret keys whenever needed
- **Ciphertext Queries:** For all ciphertexts except for $\mathsf{ct}_{i,\overline{C_i^{*j}}}$, run as experiment $0^{i^*,j^*}$. On this ciphertext, request an IBE challenge on identity $(\overline{C_i^{*j}}, i, \mathsf{tg}^*)$ with $m_0 = w_{i,\overline{C_i^{*j}}}$ and $m_1 = 0^{|w_i|}$.
- $\mathcal{A}$ returns bit b, Output b.

Observe that this is exactly Experiment $0^{i^*,j^*}$ when the message m_b is m_0 in the IBE game and Experiment $0^{i^*+1,j^*}$ when $m_b = m_1$. In addition, recall that by 1-bounded static security, since $\mathcal{A}$ queries Enc on tag tg^{*j}, $\mathcal{A}$ can only make a single key query on tg^{*j} before the challenge. Since $\mathcal{A}$ queries on C^{*j}, we can verify that no key requests are made to identity $(\overline{C_i^{*j}}, i, \mathsf{tg}^{*j})$, and all other queries are on different tg's, which will similarly result in different key queries. Thus, we can see that this is an admissible IBE game and $\mathcal{A}'$ distinguishes m_0 and m_1 with the same success $\mathcal{A}$ distinguishes Experiment $0^{i^*,j^*}$ and Experiment $0^{i^*+1,j^*}$

□

Claim D.3. For every adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{0^{n+1,j^*}}(\lambda) - \mathcal{P}_{\mathcal{A}}^{0^{1,j^*+1}}(\lambda)| = 0$.

Proof. Since i is bounded above by n, these are the same Experiment. □

Claim D.4. For every adversary PPT $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $\exists N \in \mathsf{poly}(\lambda) : \mathcal{P}_{\mathcal{A}}^{0^{0,N}}(\lambda) = \mathcal{P}_{\mathcal{A}}^{1}(\lambda)$.

Proof. Since adversary $\mathcal{A}$ is polynomially bounded in λ, it can only make polynomially many queries to Enc. If N exceeds said number of queries, then the Experiments are identical. □

Taken together, the above four claims give us a polynomial sequence of experiments with negligible difference between Experiments 0 and 1, and so Experiments 0 and 1 must be negligibly close as well.

□

Lemma D.2. Assuming GC is a secure garbled circuit scheme, for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{1}(\lambda) - \mathcal{P}_{\mathcal{A}}^{2}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. We will replace the garbled circuits requested in each challenge ciphertext with its simulation. Let $(\{(\mathsf{tg}^*_j, C^{*j})\}_{j\in N})$ be a list of queries made by $\mathcal{A}$ to Enc. Define Experiment 1^{j^*} to be the following:

- **Setup**, **Key Queries:** Are played as Experiment 1
- **Ciphertext Queries:** Adversary simulates the first j^* garbled circuit queries as $\mathsf{GC.Sim}(1^\lambda, \mathcal{U}, C^*(m))$. All others garbled circuits are generated normally.
- **Output:** Played as Experiment 1

Claim D.5. For every adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^1_{\mathcal{A}}(\lambda) - \mathcal{P}^{1^1}_{\mathcal{A}}(\lambda)| = 0$

Proof. Since j is at least 1, the conditional never occurs, and this is Experiment 1. □

Claim D.6. Assuming GC is a secure garbled circuit scheme for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^{1^{j^*}}_{\mathcal{A}}(\lambda) - \mathcal{P}^{1^{j^*+1}}_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$

Proof. Note the only difference between these Experiments are whether the j^th garbled circuit is simulated or not. Suppose $\mathcal{A}$ is a distinguisher between Experiments 1^{j^*} and 1^{j^*+1} which succeeds with noticeable probability. Then the following $\mathcal{A}'$ is a reduction which breaks garbled circuit security with the same probability.

$\mathcal{A}'(1^\lambda)$

- **Setup:**, **Key Queries:** Run as in Experiment 1
- **Ciphertext Queries:** Simulate the first $j^* - 1$ garbled circuits via $\mathsf{GC.Sim}$. For the j^{th} garbled circuit, request a sample from either $\mathsf{GC.Garble}$ or $\mathsf{GC.Sim}$ as per the garbled circuit security Experiment. Generate all other garbled circuits using $\mathsf{GC.Garble}$ and proceed with the rest of ciphertext generation as per Experiment 1.
- $\mathcal{A}$ returns bit b, Output b.

It's easy to see that this is Experiment 1^{j^*} when the circuit on j^* comes from $\mathsf{GC.Garble}$, and is Experiment 1^{j^*+1} when j^*'s garbled circuit comes from $\mathsf{GC.Sim}$

□

Claim D.7. For every PPT adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $\exists N \in \mathsf{poly}(\lambda) : \mathcal{P}^{1^N}_{\mathcal{A}}(\lambda) = \mathcal{P}^2_{\mathcal{A}}(\lambda)$.

Proof. Since adversary $\mathcal{A}$ is polynomially bounded in λ, it can only make polynomially many queries to Enc. If N exceeds said number of queries, then the experiments are identical. □

Taken together, the above three claims give us a polynomial sequence of experiments with negligible difference between Experiments 1 and 2, and so Experiments 1 and 2 must be negligibly close as well.

□

By the two lemmas above, Experiments 0 and 2 are indistinguishable, which exactly correspond to the real and simulated experiments.

E Full proof of Theorem 7.2

We will show through a sequence of Experiments that the real Experiment and simulated Experiment are computationally indistinguishable.

Experiment 0. This is the real adaptive security Experiment played between adversary $\mathcal{A}$ and challenger $\mathcal{C}$.

- **Setup:** Adversary sends $(1^n, 1^q, 1^z)$
  1. $(\mathsf{natgfe.pk}, \mathsf{natgfe.msk}) \leftarrow \mathsf{NATgFE.Setup}(1^\lambda, 1^n, 1^z)$
  2. $(\mathsf{ibe.pk}, \mathsf{ibe.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda, 1^{z+\lceil \log n' \rceil + 1})$[14]
  3. Return $(\mathsf{mpk} = (\mathsf{natgfe.pk}, \mathsf{ibe.pk}))$ to adversary.
- **Key Queries:** Adversary sends Keygen queries of the form (tg, C)
  1. Let $\mathsf{natgfe.sk}_{\mathsf{tg},C} \leftarrow \mathsf{NATgFE.KeyGen}(\mathsf{natgfe.msk}, \mathsf{tg}, C)$
  2. Sample $b_1, b_2, \ldots, b_{n'} \leftarrow \{0,1\}$
  3. For $j \in [n']$, compute $\mathsf{ibe.sk}_{j,b_j} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{ibe.msk}, (b_j, j, \mathsf{tg}))$
  4. Return $\mathsf{sk}_{\mathsf{tg},C} = (\mathsf{natgfe.sk}_{\mathsf{tg},C}, \{b_j, \mathsf{ibe.sk}_{j,b_j}\}_{j\in[n']})$ to the adversary.
- **Ciphertext Queries:** Adversary sends message m and tag tg.
  1. Run $\mathsf{ct}' \leftarrow \mathsf{NATgFE.Enc}(\mathsf{natgfe.pk}, \mathsf{tg}, m)$
  2. For all $b \in \{0,1\}, j \in [n']$, set $c_{b,j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b, j, \mathsf{tg}), \mathsf{ct}'[j])$
  3. Return $\mathsf{ct} = \{c_{b,j}\}_{b\in\{0,1\},j\in[n']}$ to the adversary.
- **Output:** Adversary outputs a bit $\in \{0,1\}$

Experiment 1. This experiment simply introduces some syntactic changes. Let st be the global state across the different phases.

- **Key Queries:**
  3. **Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Adaptive}, \mathsf{ct}') \in \mathsf{st}$
     (a) For $j \in [n']$, compute $\mathsf{ibe.sk}_j \leftarrow \mathsf{IBE.KeyGen}(\mathsf{ibe.msk}, (b_j, j, \mathsf{tg}))$
  4. **Non-Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Adaptive}) \notin \mathsf{st}$
     (a) For $j \in [n']$, compute $\mathsf{ibe.sk}_j \leftarrow \mathsf{IBE.KeyGen}(\mathsf{ibe.msk}, (b_j, j, \mathsf{tg}))$
     (b) Add $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Non-Adaptive})$ to st
  5. Return $\mathsf{sk}_{\mathsf{tg},C} = (\mathsf{natgfe.sk}_{\mathsf{tg},C}, \{b_j, \mathsf{ibe.sk}_j\}_{j\in[n']})$ to the adversary.
- **Ciphertext Queries:**
  1. **Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Non-Adaptive}) \notin \mathsf{st}$

[14] Recall n' is the length of the natgfe ciphertext.

(a) Sample $b_1, b_2, \ldots, b_{n'} \leftarrow \{0,1\}$
(b) Run $\mathsf{ct}' \leftarrow \mathsf{NATgFE.Enc}(\mathsf{natgfe.pk}, \mathsf{tg}, m)$
(c) Add $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Adaptive}, \mathsf{ct}')$ to st
(d) For all $b \in \{0,1\}, j \in [n']$, set $c_{b,j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b, j, \mathsf{tg}), \mathsf{ct}'[j])$

2. **Non-Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Non-Adaptive}) \in \mathsf{st}$
(a) Run $\mathsf{ct}' \leftarrow \mathsf{NATgFE.Enc}(\mathsf{natgfe.pk}, \mathsf{tg}, m)$
(b) For all $b \in \{0,1\}, j \in [n']$, set $c_{b,j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b, j, \mathsf{tg}), \mathsf{ct}'[j])$
3. Return $\mathsf{ct} = \{c_{b,j}\}_{b \in \{0,1\}, j \in [n']}$ to the adversary.

Experiment 2. This experiment changes how the secret keys are sampled.

- **Key Queries:**

3. **Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Adaptive}, \mathsf{ct}') \in \mathsf{st}$
(a) For $j \in [n']$, compute $\mathsf{ibe.sk}_{j,b_j} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{ibe.msk}, (b_j \oplus \mathsf{ct}'[j], j, \mathsf{tg}))$

Experiment 3. This experiment changes the way the ciphertext is sampled.

- **Ciphertext Queries:**

1. **Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Non-Adaptive}) \notin \mathsf{st}$
(d) For all $j \in [n']$
– Set $c_{b_j \oplus \mathsf{ct}'[j], j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b_j \oplus \mathsf{ct}'[j], j, \mathsf{tg}), \mathsf{ct}'[j])$
– Set $c_{\overline{b_j \oplus \mathsf{ct}'[j]}, j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (\overline{b_j \oplus \mathsf{ct}'[j]}, j, \mathsf{tg}, \overline{\mathsf{ct}'[j]}))$

Experiment 4. This experiment reorders the way the underlying non-adaptive FE is sampled when handling adaptive queries.

- **Key Queries:**

3. **Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Adaptive}, m) \in \mathsf{st}$
(a) Run $\mathsf{ct}' \leftarrow \mathsf{NATgFE.Enc}(\mathsf{natgfe.pk}, \mathsf{tg}, m)$
(b) For $j \in [n']$, compute $\mathsf{ibe.sk}_{j,b_j} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{ibe.msk}, (b_j \oplus \mathsf{ct}'[j], j, \mathsf{tg}))$

- **Ciphertext Queries:**

1. **Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Non-Adaptive}) \notin \mathsf{st}$
(b) ~~Run $\mathsf{ct}' \leftarrow \mathsf{NATgFE.Enc}(\mathsf{natgfe.pk}, \mathsf{tg}, m)$~~
(c) Add $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Adaptive}, m)$ to st
(d) For all $j \in [n']$
– Set $c_{b_j,j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b_j, j, \mathsf{tg}), 0)$
– Set $c_{\overline{b_j},j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (\overline{b_j}, j, \mathsf{tg}), 1)$

Experiment 5. This experiment simulates the non-adaptive FE.

- **Setup:**
  1. $(\mathsf{natgfe.pk}, \mathsf{natgfe.msk}) \leftarrow \mathsf{natgfe.S}_0(1^\lambda, 1^n, 1^z)$
- **Key Queries:**
  1. Let $\mathsf{natgfe.sk}_{\mathsf{tg},C} \leftarrow \mathsf{natgfe.S}_1(\mathsf{tg}, C, \perp)$
  3. **Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Adaptive}) \in \mathsf{st}$
     (a) Run $\mathsf{ct}' \leftarrow \mathsf{natgfe.S}_2(\mathsf{natgfe.pk}, \mathsf{tg}, (C, C(m)))$
- **Ciphertext Queries:**
  2. **Non-Adaptive Case:** If $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}\}, \text{Non-Adaptive}) \in \mathsf{st}$
     (a) Run $\mathsf{ct}' \leftarrow \mathsf{natgfe.S}_2(\mathsf{natgfe.pk}, \mathsf{tg}, \Pi^m)$
- **Output:** Adversary outputs a bit $\in \{0, 1\}$

Lemma E.1. For every adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^0_{\mathcal{A}}(\lambda) - \mathcal{P}^1_{\mathcal{A}}(\lambda)| = 0$.

Proof. The changes in Experiment 1 are entirely syntactic and do not affect the execution of the Experiment. □

Lemma E.2. For every adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$,$|\mathcal{P}^1_{\mathcal{A}}(\lambda) - \mathcal{P}^2_{\mathcal{A}}(\lambda)| = 0$.

Proof. In this Experiment, rather than sampling a random bit b_j to give out IBE key under identity (b_j, j, tg), we instead use $(b_j \oplus \mathsf{ct}'[j], j, \mathsf{tg})$. However, since b_j is still a uniformly random bit independent of $\mathsf{ct}'[j]$, the distribution of $b_j \oplus \mathsf{ct}'[j]$ is still uniform and independently random. □

Lemma E.3. If ibe is a secure IBE scheme, for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^2_{\mathcal{A}}(\lambda) - \mathcal{P}^3_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. We will proceed through a sequence of subhybrids where we incrementally flip the bit under portions of the challenge ciphertexts for IBE keys we do not give out. Let $\mathsf{tg}_1, \ldots, \mathsf{tg}_N$ be the list of tags for which $\mathcal{A}$ makes a Key query proceeded by a ciphertext query.

Define Experiment $2^{i^*,j^*}$ to be the following:

Experiment $2^{i^*,j^*}$. This experiment changes the way the ciphertext is sampled

1. **Setup: Key Queries:** Play as Experiment 2.
2. **Ciphertext Queries:** Adversary sends message m_i and tag tg_i which produces ciphertext ct'^i - For all ciphertext components where $j > j^*$ or $j = j^* \wedge i \geq i^*$, the components $\mathsf{ct}_{\overline{b_j \oplus \mathsf{ct}'^i[j]}, j}$ are encryptions of $\mathsf{ct}'^i[j]$ (as in Experiment 2). Otherwise, said ciphertext components are replaced with an encryption of $\overline{\mathsf{ct}'^i[j]}$ (as in Experiment 3).
3. **Output:** Played as Experiment 2

Claim E.1. For every adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$,$|\mathcal{P}^2_{\mathcal{A}}(\lambda) - \mathcal{P}^{2,1}_{\mathcal{A}}(\lambda)| = 0$

Proof. Since i and j are at least 1, the conditional never occurs, and the Experiments are identical. □

Claim E.2. If ibe is a secure IBE scheme, for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{2^{i^*,j^*}}(\lambda) - \mathcal{P}_{\mathcal{A}}^{2^{i^*,j^*+1}}(\lambda)| = \mathsf{negl}(\lambda)$

Proof. Notice that Experiments $2^{i^*,j^*}$ and $2^{i^*,j^*+1}$ only differ in how $\mathsf{ct}_{\overline{b_{j^*} \oplus \mathsf{ct}'^{i^*}[j^*]},j^*}$ is generated. Suppose $\mathcal{A}$ is a distinguisher between Experiments $2^{i^*,j^*}$ and $2^{i^*,j^*+1}$ which succeeds with noticeable probability. Then the following $\mathcal{A}'$ is a reduction which breaks IBE security with the same probability.

$\mathcal{A}'(1^\lambda)$

- **Setup:** Receive $\mathsf{ibe.mpk}$ from IBE challenger and send to $\mathcal{A}$
- **Key Queries:** Answer the key queries of $\mathcal{A}$ by querying the IBE challenger for secret keys whenever needed
- **Ciphertext Queries:** For all ciphertexts except for $\mathsf{ct}_{\overline{b_{j^*} \oplus \mathsf{ct}'^{i^*}[j^*]},j^*}$, run as experiment $2^{i^*,j^*}$. On this ciphertext, request an IBE challenge on identity $(\overline{b_j \oplus \mathsf{ct}'^{i}[j]}, j, \mathsf{tg}_{i^*})$ with $m_0 = \mathsf{ct}'^{i^*}[j^*]$ and $m_1 = \overline{\mathsf{ct}'^{i^*}[j^*]}$.
- $\mathcal{A}$ returns bit b, Output b.

Observe that this is exactly Experiment $2^{i^*,j^*}$ when the message m_b is m_0 in the IBE game and Experiment $2^{i^*,j^*+1}$ when $m_b = m_1$. Note that since the ciphertext query adds $(\mathsf{tg}, \{b_1, b_2, \ldots, b_{n'}, 1, \mathsf{ct}'\})$ to st, any future calls to the keygen query will follow the first branch of the conditional, so keygen will only request IBE keys on $(b_{j^*} \oplus \mathsf{ct}'_{j^*}, j^*, \mathsf{tg}_{i^*})$ in that component, and not on $(b_{j^*} \oplus \mathsf{ct}'_{j^*}, j^*, \mathsf{tg}_{i^*})$. Since $(\mathsf{tg}, \{b_1, b_2, \ldots b_{n'}\}, 0) \notin \mathsf{st}$, we know there were no previous keygen queries on tg_{i^*}, and hence no IBE keygen queries to any identities of the form $(\cdot, \cdot, \mathsf{tg}_{i^*})$. Thus, we can see that this is an admissible IBE game and $\mathcal{A}'$ distinguishes m_0 and m_1 with the same success $\mathcal{A}$ distinguishes Experiment $2^{i^*,j^*}$ and Experiment $2^{i^*,j^*+1}$.

□

Claim E.3. For every adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{2^{i^*,n'+1}}(\lambda) - \mathcal{P}_{\mathcal{A}}^{2^{i^*1,1}}(\lambda)| = 0$

Proof. Since j is bounded above by n', these are the same experiment. □

Claim E.4. For every adversary PPT $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $\exists N \in \mathsf{poly}(\lambda) : \mathcal{P}_{\mathcal{A}}^{2^{0,N}}(\lambda) = \mathcal{P}_{\mathcal{A}}^{3}(\lambda)$.

Proof. Since adversary $\mathcal{A}$ is polynomially bounded in λ, it can only make polynomially many queries to Enc. If N exceeds said number of queries, then the experiments are identical. □

Taken together, the above four claims give us a polynomial sequence of experiments with negligible difference between Experiments 2 and 3, and so Experiments 2 and 3 must be negligibly close as well.

□

Lemma E.4. For every adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{3}(\lambda) - \mathcal{P}_{\mathcal{A}}^{4}(\lambda)| = 0$.

Proof. The changes in Experiment 4 are entirely syntactic. Observe that setting

- $c_{b_j \oplus \mathsf{ct}'[j],j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b_j \oplus \mathsf{ct}'[j], j, \mathsf{tg}), \mathsf{ct}'[j])$
- $c_{\overline{b_j \oplus \mathsf{ct}'[j]},j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (\overline{b_j \oplus \mathsf{ct}'[j]}, j, \mathsf{tg}), \overline{\mathsf{ct}'[j]})$

is equivalent to setting

- $c_{b_j,j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (b_j, j, \mathsf{tg}), 0)$
- $c_{\overline{b_j},j} = \mathsf{IBE.Enc}(\mathsf{ibe.pk}, (\overline{b_j}, j, \mathsf{tg}), 1)$

for either bit value of ct_j. This allows us to delay the generation of ct' to KeyGen, (note that the ct' is still generated in the same way however).

□

Lemma E.5. If natgfe is a non-adaptively secure 1-bounded tagged FE scheme, for every PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^4_{\mathcal{A}}(\lambda) - \mathcal{P}^5_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. Observe that in Experiment 5, we simply replace the calls of $\mathsf{natgfe.Setup}$, $\mathsf{natgfe.Enc}$ and $\mathsf{natgfe.KeyGen}$ with $\mathsf{natgfe.S}_0$, $\mathsf{natgfe.S}_1$, and $\mathsf{natgfe.S}_2$. To see that these Experiments are indistinguishable, we simply need to check that the calls to satisfy the admissibility constraints of the non-adaptive simulator.

We break our analysis into two cases for each tag - the adaptive and non-adaptive.

1. If tg is a tag where $\mathcal{A}$ asks for a key query before a ciphertext query.
   Observe in this case, the key query sets a $(\mathsf{tg}, \{b_1, b_2, \dots b_{n'}\}, \text{Non-adaptive})$ into st, so the non-adaptive key query and non-adaptive ciphertext query each happen in the corresponding adaptive key and ciphertext query.

2. If tg is a tag where $\mathcal{A}$ asks for a key query after a ciphertext query.
   Observe in this case, the ciphertext query sets a $(\mathsf{tg}, \{b_1, b_2, \dots b_{n'}\}, \text{Adaptive})$ into st, so the non-adaptive key and ciphertext oracles are called exactly once in that order in the adaptive ciphertext query (and not at all in the adaptive key query).

Thus, the we can argue the challenger of the above security Experiment is an admissible adversary in the non-adaptive 1-bounded tagged FE Experiment, and so they must be negligibly close.

□

Note that by Experiment 5, the key and ciphertext queries are generated exactly as in the simulators described, where ciphertext queries are generated using only Π^m. By the lemmas above, Experiments 0 and 5 are indistinguishable, which exactly correspond to the real and simulated Experiments.