A Simple and Generic Approach to Dynamic Collusion Model

Rachit Garg
UT Austin*

Rishab Goyal
MIT†

George Lu
UT Austin‡

Abstract

Functional Encryption (FE) is a powerful notion of encryption which enables computations and partial message recovery of encrypted data. In FE, each decryption key is associated with a function f such that decryption recovers the function evaluation $f(m)$ from an encryption of 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 $q = q(\lambda)$ decryption keys. In the last decade, numerous works have proposed many FE constructions from a wide array of algebraic and general cryptographic assumptions, and proved their security in the bounded collusion model.

However, until very recently, all these works studied bounded collusion resistance in a *static model*, where the collusion bound q was a global system parameter. While the static collusion model led to great research progress in the community, it has many major drawbacks. Very recently, Agrawal et al. (Crypto 2021) and Garg et al. (Eurocrypt 2022) independently introduced the *dynamic model* for bounded collusion resistance, where the collusion bound q was a fluid parameter that was not globally set but only chosen by each encryptor. The dynamic collusion model enabled harnessing the many virtues of the static collusion model, while avoiding its various drawbacks.

In this work, we give a simple and generic approach to upgrade any scheme from the static collusion model to the dynamic collusion model. Our result captures all existing results in the dynamic model in the form of a single unified framework, and also gives new results as simple corollaries with a lot more potential in the future. An interesting artifact of our result is that it gives a generic way to match existing lower bounds in functional encryption.

1 Introduction

Functional Encryption (FE) [SW05, BSW11] is a powerful generalization of public-key encryption [DH76] which enables fine-grained access over encrypted data. In such systems, a setup algorithm produces a master public-secret key pair $(\mathsf{mpk}, \mathsf{msk})$, where the master public key is made public and the master secret key is retained by an authority. Using mpk, any user can encrypt data m to produce a ciphertext ct. On the other hand, the authority can use the master secret key to generate arbitrarily many partial decryption keys for authorized users, where the decryption

*Email: rachg96@cs.utexas.edu.

†Email: rishab@mit.edu. Research supported in part by DARPA under Agreement No. HR00112020023, a grant from the MIT-IBM Watson AI, a grant from Analog Devices and a Microsoft Trustworthy AI grant. 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.

key sk_f for a function f enables the key holder to compute the function output $f(m)$ from the ciphertext encrypting data m while learning nothing else about m. The message space $\mathcal{M}$ and function class $\mathcal{F}$ supported depend on the expressiveness of the corresponding cryptosystem. For example, public-key encryption can be viewed as FE with $\mathcal{F}$ containing only the identity function, and identity-based encryption (IBE) [Sha85, Coc01, BF01] is equivalent to FE where $\mathcal{F}$ contains all point functions. For well over a decade, FE has shown to be a tremendously useful concept with applications ranging from all aspects of cryptography to theory of computation more generally.

The security for such systems is captured by either an indistinguishability (IND) or simulation (SIM) based security game between a challenger and an attacker. In the IND-based security game [BW07, KSW08], the challenger generates the master public key mpk, and sends it to the attacker. The attacker makes polynomially many key queries, each for a function $f_i \in \mathcal{F}$, for which it receives back the corresponding functional key sk_{f_i}. The attacker also submits two challenge messages m_0, m_1 at any point during the game with the restriction that $f_i(m_0) = f_i(m_1)$ for all queried functions $f_1, \ldots, f_q$. The challenger flips a coin b and returns a challenge ciphertext ct^* encrypting m_b. For an IND-secure FE scheme, an attacker can correctly guess b only with probability negligibly close to $\frac{1}{2}$. On the other hand, in the SIM-based security game [BSW11, O'N10], the attacker submits a single challenge message m, and the challenger flips a coin to decide whether it generates the keys $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_q}$ and ciphertext ct^* honestly, or it "simulates" them given only the evaluations $f_1(m), \ldots, f_q(m)$ and nothing more about m. Again, for a SIM-secure FE scheme, an attacker can not correctly guess with probability negligibly better than $\frac{1}{2}$.

The Collusion Boundary. While SIM-secure FE is more desirable as it closely captures the intuition that an attacker learns *at most the function evaluation but nothing more*, it is known to run into strong impossibility results [BSW11, AGVW13]. Agrawal et al. (AGVW) [AGVW13, Corollary 1.2] gave a strong unconditional lower bound where they proved that there does not exist any SIM-secure FE scheme with ciphertexts of size $o(q)$, where q is the number of colluding users. While this lower bound does not rule out IND-secure FE schemes with similar parameters, it was shown soon after, in the works of Ananth and Jain (AJ) [AJ15, AJS15] and Bitansky and Vaikuntanathan (BV) [BV15], that any IND-secure FE scheme with ciphertexts of size $o(q)$ can be generically bootstrapped into an FE scheme with optimal parameters where ciphertexts are of size $O(\log q)$. And, these bootstrapping theorems have been the pathway taken over the past several years culminating in all-powerful obfuscators from nearly standard cryptographic assumptions [JLS21].

These results clearly demarcate what we refer to as "the collusion boundary" for FE schemes. Simply put, FE schemes with ciphertexts of size $\Theta(q)$ define the boundary of feasibility in SIM-security due to the AGVW lower bound, and the boundary of *obfustopia* in IND-security as any improvement leads to obfuscation via the AJ-BV bootstrapping. Since general purpose obfuscation is known to be an incredibly powerful cryptographic primitive, thus one can not hope to cross this *collusion boundary* without relying on highly structured algebraic assumptions.

However, a natural question that has been quite extensively studied is – "*how close to this boundary can we get?*" To better understand the significance of this boundary and potentially answer this question, the notion of bounded collusion resistance in FE has been well investigated. Although the original motivation behind the bounded collusion model was to use it as a first step to gain some foothold in designing FE systems, an alternate perspective is that it is a means to understand existing lower bounds in SIM-security and explore the limits of generic unstructured assumptions in building IND-secure FE.

Bounded Collusion Model. Traditionally in the bounded collusion model, the FE system declares a bound q at the setup time, such that all the system parameters grow with q (in addition to the security parameter λ). And, the security requirement is as before with the restriction that the attacker is no longer allowed to make more than q key queries, and if the attacker corrupts more than q keys, then *no security is provided*. Despite its obvious limitations, the bounded collusion model has been well-studied with the goals of improving efficiency, post-quantum security, reducing computational assumptions, and supporting more expressive class of functions [DKXY02, SS10, GLW12, GVW12, AR17, Agr17, ISV+17, AS17, GKW18, CVW+18, AV19, GSW21, Wee21].

In this long line of research in the bounded collusion model, Ananth and Vaikuntanathan [AV19] made tremendous progress towards answering the question of how close we can get to the collusion boundary. They proposed a generic black-box compiler that, starting from any IND/SIM-secure FE scheme with ciphertexts of size $\mathsf{poly}(q)$, built an IND/SIM-secure FE scheme with ciphertexts of size $\Theta(q)$. Since the compiler is agnostic to the supported function class $\mathcal{F}$ and starting polynomial poly, this gives a mechanism to bring all existing FE constructions down to the collusion boundary (irrespective of the model of computation, the supported function class etc).

While at first glance, this seems to fully close the gap between the (linear) collusion boundary and positive results for FE, unfortunately, this is not the case! Observe that the collusion boundary, as defined by the AGVW lower bound and AJ-BV bootstrapping theorems, only dictates the ciphertext size to be $\Omega(q)$. That is, the sizes of other FE parameters such as the master key sizes, functional key sizes are not mandated to be $\Omega(q)$, and could be totally independent of the collusion bound. In other words, the collusion boundary suggests that only the FE encryption phase needs to grow with the collusion bound q. This left a major gap in our understanding of the bounded collusion model despite significant progress over the last decade.

Dynamic Collusion Model. Motivated by the above gap, the notion of dynamic collusion model was introduced in two recent concurrent works by Agrawal et al. [AMVY21] and Garg et al. [GGLW22]. They referred to the older bounded collusion models as the static collusion model.

In the dynamic collusion model, the setup algorithm no longer takes the collusion bound q as input, but instead, the encryptor selects the collusion bound q per ciphertext. That is, the setup and key generation algorithms no longer depend on q, and this lets the encryptor *dynamically* decide the maximum number of colluding users against which it desires security. As a consequence, in the dynamic collusion model, only the size of the ciphertexts grows with q, while everything else is independent of q. The security requirement is as before but with the relaxed restriction that the attacker can adaptively choose (i.e., when obtaining the challenge ciphertext instead of declaring at the beginning of the game) the maximum number of keys it will query for, and the security holds as long as it corrupts fewer than said number of keys.

With the model in place, both [AMVY21] and [GGLW22] provided FE schemes for circuits in the dynamic model assuming the minimum and necessary[1] assumption of identity-based encryption. Both the works using nearly similar intermediate tools (multi-party computation protocol in the client-server framework [AV19]) showed how to encode the collusion bound more efficiently resulting in carefully designed FE schemes in the dynamic collusion model where only the ciphertext size grows with q and that too only linearly. Furthermore, [AMVY21] also gave succinct FE schemes for circuits, and FE schemes for Turing Machines (TMs) and Nondeterministic Logspace (NL) by leveraging the hardness of learning with errors (LWE) assumption [Reg05].

[1] See [AMVY21, §7] for more details about minimality of IBE.

Although these FE schemes for circuits from IBE (almost) close the gap between the lower bounds suggested by the collusion boundary and the positive results for FE, the constructions are not fully satisfactory. The main reason is that both the works [AMVY21, GGLW22] require and perform careful surgery of existing FE schemes secure in the static collusion model to obtain their results in the dynamic model. As an unfortunate consequence, they were unable to capture all the research progress made in the static collusion model over the last decade. Therefore, in order to enjoy the benefits and different security/functionality trade-offs provided bythe numerous existing FE schemes in the static collusion model, one would need to *individually* study and try to translate every FE scheme known in the static collusion model to their dynamic counterpart. Although it should be possible to upgrade every known static collusion secure FE scheme using the core technical ideas developed in [AMVY21, GGLW22], this will be very inefficient as this would require a separate construction for upgrading each construction from the set [DKXY02, SS10, GLW12, GVW12, AR17, Agr17, ISV+17, AS17, GKW18, CVW+18, AV19, GSW21, Wee21].

In this work, we study the connection between the two collusion models at a finer level, and inspect how large the gap between the two collusion models really is. Our goal is to derive a generic mechanism to upgrade any static collusion secure FE scheme into a dynamic collusion secure scheme without requiring individual careful surgery of each underlying FE scheme.

Our Results and Techniques

The current constructions for FE in the dynamic model [AMVY21, GGLW22] follow a two-step template. First, they show how to build an intermediate static FE scheme where q is still a global system parameter, but the running times of the setup and key generation algorithms grows only poly-logarithmically with the collusion bound q. And, they show a simple generic transformation to lift such an intermediate static FE scheme with this poly-logarithmic dependence property into a fully dynamic FE scheme for the same function class, where q is no longer a global system parameter and only the encryptor selects the collusion bound. The core idea behind such a generic lifting procedure is the well-known "powers-of-two trick" [GKP+13]. In a few words, the dynamic FE scheme now sets up λ parallel static FE systems with geometrically increasing collusion bounds from $2, 4, \ldots, 2^\lambda$. And, encryptor selects exactly one of these systems to encrypt its message depending upon the desired level of collusion security, whereas the key generator generates individual keys for all λ of these static FE systems.

Now to instantiate the above template, both works build new static FE schemes for different function classes with the above desired poly-logarithmic dependence property. While the exact implementations of Agrawal et al. and Garg et al. differ here, the main idea is identical. They both rely on the same ingredients of identity-based encryption [Sha85], a special reusable multi-party computation protocols [AV19], and garbled circuits [Yao82]. The core idea is to rely on a combination of a load balancing trick from [AV19], delayed encryption trick from [GKW16], and compression properties of IBE.

The intuition behind the load balancing trick was to visualize the collusion bound as a "load" on the system, and then distribute the "load" of q colluding users into $O(q)$ buckets, each with a maximum load of just λ colluding users. Basically, each bucket corresponds to a separate static FE system with the collusion bound λ. And, the observation is if $O(q)$ of such parallel static FE schemes (or buckets) can be efficiently represented within a single FE system, then such a compressed FE system can be used to instantiate the desired static FE scheme with the poly-log dependence

property via the load balancing trick. With the above intuition, both works [AMVY21, GGLW22] focussed on building such a static FE scheme which could encode many parallel static FE schemes more efficiently. At this point, [GGLW22] explicitly introduced a new type of static FE scheme for distributing the load of colluding users, while [AMVY21] handled it implicitly. Formally, [GGLW22] introduced a new static FE notion that they referred to as Tagged Functional Encryption (or, Tagged FE for short).

A tagged FE scheme can be visualized as a compressed version of static FE, similar to how IBE is a compression of plain PKE, where each ciphertext and function key are associated with a special identity or 'tag' value. On any particular tag, regular bounded functional encryption security holds even when given oracle access to an unrestricted amount and type of function keys on any other tag (in addition to the normal collusion-bounded number of keys on the challenge tag). This allows for a succinct representation of a potentially exponential number of public-secret key pairs into a short master public-secret key pair.

The final idea in the two works was to open up existing static FE schemes such as [AV19], and surgically add this property by compressing all the key materials with the help of IBE. Broadly, both works realized that, at their core, most static FE schemes sampled a large number of public-secret key pairs for aplain public-key encryption scheme where the number of sampled schemes grew with the collusion bound q. They noticed that there were two sources of inefficiencies – one, due to the master public-secret key sizes growing with q; and two, the exponential number of parallel static FE schemes associated with a distinct tag. And, they further observed that if one replaces each PKE key pair with a publicly computable identity and its corresponding IBE secret key, then all the system inefficiencies can be handled efficiently using the separate portions of the identity space of an IBE scheme. This gives way to a static FE scheme with the desired poly-logarithmic dependence property, and eventually to a fully dynamic FE scheme by combining all these ideas.

While the above approach led to the first dynamic FE schemes, they required careful surgery of a select few static FE schemes. Since the static collusion model has already been widely adopted by the community, and appears to be simpler for developing new constructions, it is very likely that in the future, new constructions in the bounded collusion model may be provided first in the static model, and then later upgraded to dynamic model by relying on the surgical ideas similar to those developed in [GGLW22, AMVY21]. Thus, we are interested in following question –

Can we generically upgrade the security of any functional encryption scheme from the static collusion model to the dynamic collusion model?

We answer the above question in the affirmative. The main contribution of this paper is to close the gap between static and dynamic collusion models by providing a simple and generical compiler to upgrade a static scheme to a dynamic one. Our compiler is unaffected by the underlying model of computation, the class of functions and messages supported, the parameter growth in terms of the collusion bound, or any other non-generic property of the underlying static FE scheme. In this work, we prove the following.

Theorem 1.1 (Informal)**.** There exists an explicit compiler that upgrades any static λ-bounded collusion resistant functional encryption scheme into a dynamic bounded collusion resistant functional encryption scheme.

This has a fascinating consequence as this shows that all existing static collusion FE schemes be generically upgraded to be dynamic collusion secure without diving into any of the mechanics of the

underlying static FE schemes. But even more importantly, it suggests that any future constructions of a static FE scheme can be combined with our compiler to derive a dynamic FE scheme as a simple corollary via our main theorem.

Our Compiler. The main technical component of our result is a generic compiler that compresses an exponential number of static FE schemes into a single tagged FE scheme that can support an exponential number of tags. We want to highlight that constructing a tagged FE scheme (generically from a static FE scheme) is sufficient for our compiler, as by relying on the load balancing trick [AV19] and the powers-of-two technique [GKP+13] we can upgrade a tagged FE scheme into a dynamic FE scheme as done in [GGLW22]. Thus, in the following technical exposition, we focus on the showing how to compress N independent instances of a static FE scheme into a tagged FE scheme where the running time of all algorithms grows only poly-logarithmically with N. Concretely, we prove the following next:

Theorem 1.2 (Informal)**.** Assuming IBE, for any $N, q > 0$, there exists an explicit transformation from a static q-bounded collusion FE scheme to a q-bounded tagged FE scheme with tag space $[N]$.

To better understand our transformation, let us compare a static FE scheme and a tagged FE scheme again. As we discussed before, the only difference between a static FE and a tagged FE is the following. In a regular static FE scheme, each ciphertext ct_m and secret key sk_f is associated with only a message m and a function f (respectively). While in a tagged FE system, each ciphertext $\mathsf{ct}_{m,\mathsf{tag}_1}$ and secret key $\mathsf{sk}_{m,\mathsf{tag}_2}$ is also associated with a tag, tag_1 and tag_2 (respectively), in addition to a message m and a function f. Where the decryption correctness in tagged FE changes to the condition that $\mathsf{Dec}(\mathsf{sk}_{m,\mathsf{tag}_2}, \mathsf{ct}_{m,\mathsf{tag}_1})$ is equal to $f(m)$ *iff* $\mathsf{tag}_1 = \mathsf{tag}_2$, otherwise it outputs $\perp$.

Our main observation is that a tagged FE can quite simply be interpreted as an identity-based static FE scheme. That is, the relationship behind tagged FE and static FE is analogous to that between IBE and PKE. Thus, it seems like all we need to do is find a mechanism to embed identities in both the secret keys and ciphertexts of a tagged FE scheme. At a very high level, a similar problem was solved in an unrelated context of building registration-based encryption [GHMR18, GHM+19, GV20]. Although the motivation behind registration-based encryption was to solve the key-escrow problem, one could very easily visualize it as a mechanism to accumulate a large number of independently sampled PKE public keys into a short commitment where each public key is associated with an identity (and, so is the corresponding secret key as well). And, an encryptor only needs the short commitment to encrypt the message for any particular identity. Internally, these encryption schemes rely on the beautiful line of working studying non-black-box IBE constructions from simpler assumptions [DG17b, DG17a].

Thus, it seems very natural to us that these techniques can also be applied to the setting of static bounded collusion resistant FE as well. Basically, we do not see the usage of PKE in these prior works [DG17b, DG17a, GHMR18, GHM+19] as an irreplacable component, rather we view this overall line of work as providing a beautiful mechanism to embed identities in any type of functional encryption scheme, and not just in plain public-key encryption schemes. While embedding identities in fully collusion resistant FE schemes does not seem to be useful for any new applications, we point out that embedding identities is a meaningful operation for bounded collusion FE schemes. And, in this case, it leads to a simple and generic approach to dynamic FE systems. We believe that our re-visualization of Döttling-Garg [DG17b, DG17a] garbled-circuit-based tree compression techniques to encryption schemes beyond public-key encryption will find many more applications in the future.

Before moving to a high-level overview of our compiler using the garbled-circuit-based tree compression techniques, we want to highlight the aforementioned re-visualization of replacing public-key encryption with more expressive forms of encryption is not reliant on the encryption being IND-secure or SIM-secure. Although in the main body, we stick to simulation-based definitions, the security proofs work equally well in case we start with an IND-secure static FE scheme. Very briefly, the reason is the hardwiring-based proof strategy [DG17b, DG17a] only requires simulation of garbled circuits, and not the underlying encryption system. Next, we describe the main ideas and structure of our compiler.

From Static to Tagged FE via Garbled Circuits. As described above, we rely on the beautiful tree compression techniques of Döttling-Garg [DG17b, DG17a]. We note that these techniques are useful in delegating computation to the decryption algorithm via encrypting a sequence of garbled circuits. In this work, we rely on the notion of One-Time Signature with Encryption (OTSE) introduced in [DGHM18] for building IBE from plain PKE using garbled circuits. Briefly, an OTSE scheme is a one-time signature scheme that is equipped with an encryption and decryption algorithm, where encryption is performed w.r.t. a verification key and a pair of message bit and index such that the resulting ciphertext can be decrypted using any signature that where the message bit matches at the corresponding index position. That is, a ciphertext ct is associated with an index-bit pair (i, b), and decryption only needs a signature σ on any message x such that $x_i = b$.

Let us start with a simpler goal which is of *compressing* and *embedding* N independent instantiations of a (static untagged) functional encryption scheme into a tagged FE scheme where the parameter sizes grow only poly-logarithmically with N, but the running time of the setup and key generation algorithms is allowed to grow polynomially with N. The idea is to (independently) simply sample N fresh key pairs $(\mathsf{mpk}_i, \mathsf{msk}_i)$ for the untagged system during the setup, and create a short digest for all N public keys by placing each mpk_i into a leaf node of a Merkle tree and the root node of the tree will be the short master public key for the compressed/tagged FE system. Basically, the root value of the Merkle tree serves as a short commitment to the sequence of N master public keys, and each leaf node can be succinctly opened w.r.t. the root node. To encrypt a message m for the tag^{th} untagged FE system (i.e., under key $\mathsf{mpk}_{\mathsf{tag}}$), the encryptor needs to search the Merkle tree to obtain the corresponding public key from the root node. Since an encryptor cannot perform this search operation during encryption time, thus it defers it to the decryption phase by generating a sequence of garbled circuits. The final garbled circuit outputs the encryption of message m under the correct key $\mathsf{mpk}_{\mathsf{tag}}$, as the intermediate garbled circuits perform step-by-step verification of the root-to-leaf path where at each step only one bit of the tag value tag is read and verified. And, all the garbled wire keys are encrypted using the OTSE scheme such that the encryptor enforces the final garbled circuit to only be evaluated on $\mathsf{mpk}_{\mathsf{tag}}$. The key generation process involves computing the untagged FE decryption key for the function f w.r.t. master key $\mathsf{msk}_{\mathsf{tag}}$, and a sequence of signatures such that they enable evaluation of the intermediate garbled circuits along the appropriate root-to-leaf path. And, for decryption, one needs to decrypt the encrypted wire keys and evaluate the corresponding garbled circuit alternatively such that the it finally obtains the wire keys corresponding to the key $\mathsf{mpk}_{\mathsf{tag}}$, and then create the untagged FE ciphertext using the last garbled circuit which is finally decrypted by using the tagged FE decryption algorithm.

While the above idea is only useful for compression of a polynomial sized sequence of master public keys, this can be easily extended to compress an exponential number of master keys by

using a simple lazy sampling technique where the setup algorithm only samples a PRF key, and uses the PRF key to sample all the untagged FE keys. Thus, the running time does not grow with the number of underlying FE systems, and consistency of the Merkle tree is ensured by the PRF key. This concludes a high-level overview of our main compiler, and more details are provided later in Section 4. By combining this with the simple black-box transformations [GGLW22], we get our final result of upgrading any static FE scheme to its dynamic FE counterpart.

New Results and Future Impact

As we hinted earlier, our generic compiler not only provides a much simpler approach to all the dynamic FE constructions in [AMVY21, GGLW22], but it goes above beyond them as it readily gives new results not captured by them. Very recently, there have new constructions for attribute-based encryption in static bounded collusion model for TMs from IBE [GSW21] and DFAs from LWE [Wee21]. Using our compiler, we can immediately upgrade them to get the following new results as simple corollaries.

Corollary 1.1 (Informal)**.** Assuming the existence of an Identity Based Encryption scheme, there exists an Attribute Based Encryption scheme for Turing Machines in the dynamic collusion model.

Corollary 1.2 (Informal)**.** Assuming the existence of an Identity Based Encryption scheme and Learning with Errors assumption, there exists an Attribute Based Encryption scheme for DFA's in the dynamic collusion model.

We remark that this already shows the value of our compiler as it gives new results as mere corollaries. And it also has huge potential in future research in the bounded collusion model, as this gives the community a simpler target to aim for by showing it will be sufficient to build a static FE scheme rather than a dynamic FE scheme. Because the latter can be generically obtained by using our compiler without any further surgical procedures similar to [AMVY21, GGLW22].

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 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 Pseudorandom Functions

A pseudorandom function (PRF) is a funcion that takes inputs in domain $\mathcal{D}_\lambda$, samples a PRF seed of λ bits and computes an output in the range, $\mathcal{R}_\lambda$. The evaluation function runs polynomially in λ and satisfies the following pseudorandomness property.

Definition 2.1 (Pseudorandomness)**.** A PRF scheme is said to be secure if for every stateful PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\mathcal{D}_\lambda, \mathcal{R}_\lambda$ the following holds:

$$\Pr\left[\mathcal{A}^{\mathsf{PRF}(s,\cdot)}(r_b) = b : \begin{array}{c} s \leftarrow \{0,1\}^\lambda,\ b \leftarrow \{0,1\} \\ x^* \in \mathcal{D}_\lambda \leftarrow \mathcal{A}^{\mathsf{PRF}(s,\cdot)}(1^\lambda) \\ r_0 \leftarrow \mathcal{R}_\lambda,\ r_1 = \mathsf{PRF}(s, x^*) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where $\mathcal{A}$ must not query the challenge input x^* to the evaluation oracle $\mathsf{PRF}(s, \cdot)$.

2.2 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 2.2. 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:

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

The following corollary follows from the definition.

Corollary 2.1. If GC is a secure garbling scheme for a class of circuits $\mathcal{C} = \{\mathcal{C}_n\}_n$, then for all n, $C \in \mathcal{C}_n$ and $x \in \{0,1\}^n$, the following distributions are computationally indistinguishable:

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

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

2.3 One-Time Signature with Encryption

A One-Time Signature with Encryption (OTSE) scheme, defined by [DG17a], is a one-time signature scheme with an additional encryption and decryption functionality. An OTSE scheme, consists of the following five algorithms, (SSetup, SGen, SSign, SEnc, SDec)[2].

$\mathsf{SSetup}(1^\lambda, \ell) \to \mathsf{pp}$. The setup algorithm takes as input the security parameter λ and a message length parameter ℓ. It outputs a public parameter pp.

$\mathsf{SGen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$. Once parameters pp are setup, we can sample a verification and signing key.

$\mathsf{SSign}(\mathsf{pp}, \mathsf{sk}, x) \to \sigma$. On input the public parameters, a signing key and a message $x \in \{0,1\}^\ell$, we output a signature σ.

$\mathsf{SEnc}(\mathsf{pp}, (\mathsf{vk}, i, b), m) \to \mathsf{ct}$. On input the public parameters, and a verification key vk, a position $i \in [\ell]$, a bit $b \in \{0,1\}$ and a plaintext m, we output a ciphertext ct.

$\mathsf{SDec}(\mathsf{pp}, (\mathsf{vk}, \sigma, x), \mathsf{ct}) \to m'$. On input the public parameters, a verification key vk, a signature σ, a message x and ciphertext ct, we output a plaintext m'.

The scheme satisfies the following properties,

Correctness. For all $\lambda \in \mathbb{N}, \ell \in \mathbb{N}, x \in \{0,1\}^\ell$ and plaintext m, if $\mathsf{pp} \leftarrow \mathsf{SSetup}(1^\lambda, 1^\ell)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SGen}(\mathsf{pp})$ and $\sigma \leftarrow \mathsf{SSign}(\mathsf{pp}, \mathsf{sk}, x)$, then,

$$\mathsf{SDec}(\mathsf{pp}, (\mathsf{vk}, \sigma, x), \mathsf{SEnc}(\mathsf{pp}, (\mathsf{vk}, i, x_i), m)) = m.$$

Succinctness. For $\mathsf{pp} \leftarrow (1^\lambda, 1^\ell)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SGen}(\mathsf{pp})$, it holds that vk is a polynomial function in λ and independent of ℓ.

Selective Security. For every PPT Adversary $\mathcal{A}$, there exists a negligible function negl such that for all $\lambda \in \mathbb{N}, \ell \in \mathbb{N}$, the following holds.

$$\Pr\left[\mathcal{A}(\mathsf{pp}, \mathsf{vk}, \sigma, \mathsf{ct}^*) = b : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{SSetup}(1^\lambda, \ell) \\ x \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{SGen}(\mathsf{pp}); \quad \sigma \leftarrow \mathsf{SSign}(\mathsf{pp}, \mathsf{sk}, x) \\ (i, m_0^*, m_1^*) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{vk}, \sigma); \quad b \leftarrow \{0,1\} \\ \mathsf{ct}^* \leftarrow \mathsf{SEnc}(\mathsf{pp}, (\mathsf{vk}, i, 1-x_i), m_b^*) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda),$$

where the probability depends on the randomness of the different OTSE algorithms. In our proof we use the multi-message security, wich is implied by a single message security via a standard hybrid argument. See [DG17a] for details.

3 Functional Encryption: Definitions

In this section, we revisit the notion of functional encrypion (FE) in the bounded setting [SS10, GVW12]. Recent works of [AMVY21, GGLW22] proposed a collusion bound in the dynamic setting where the scheme's setup and key generation algorithms are independent of the collusion bound and instead, we can specify the collusion bound during encryption. We follow the formal security definitions from [GGLW22] almost verbatim and describe them below.

[2]There is no need for a verification algorithm for the signature as one can run the encryption and decryption algorithm to check validity of the signature.

3.1 Static Collusion Model

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

$\mathsf{Setup}(1^\lambda, 1^n, 1^q) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm takes as input the security parameter λ, the functionality index n[3] and the collusion bound 1^q. It 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 $y \in \mathcal{R}_n$.

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

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

And, it is said to be efficient if the running time of the algorithms is a fixed polynomial in the parameters λ, n and q.

Static bounded collusion security. This is formally captured via the following 'simulation based' security definition as follows.

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

[3] 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.

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

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

$$\approx_c$$

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

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 $\Pi^m = \big((f_1, f_1(m)), \ldots, (f_{q_{\mathsf{pre}}}, f_{q_{\mathsf{pre}}}(m))\big)$.
- $\mathcal{A}$ 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.

3.2 Dynamic Collusion Model

In a "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 $(\mathsf{mpk}, \mathsf{msk})$.

$\mathsf{Enc}(\mathsf{mpk}, m \in \mathcal{M}_n, 1^q) \to \mathsf{ct}$. 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 as an input. It outputs a ciphertext ct.

Efficiency. The runtime and output of $\mathsf{Setup}, \mathsf{KeyGen}$ is polynomial in λ, n. The runtime and output of Enc, the runtime of Dec is polynomial in λ, n, q.

Dynamic bounded collusion security. We define a 'simulation based' security noion similar to the static security definition (Definition 3.1).

Definition 3.2 (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 Sim such that for every stateful PPT adversary $\mathcal{A}$, the following distributions are computationally indistinguishable:

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

$$\approx_c$$

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

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

3.3 Tagged Functional Encryption

Next, we recall the notion of tagged FE from [GGLW22]. Tagged FE is simply a collection of different instances of a functional encryption scheme, where each instance is denoted by a $\mathsf{tag} \in \mathcal{I}_z$ ($|\mathcal{I}_z|$ is the total number of FE instances bundled together). A tagged bounded functional encryption scheme FE for a family of function classes $\{\mathcal{F}_n\}_{n\in\mathbb{N}}$, message spaces $\{\mathcal{M}_n\}_{n\in\mathbb{N}}$ and tag spaces $\mathcal{I} = \{\mathcal{I}_z\}_{z\in\mathbb{N}}$ consists of four polynomial-time algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ with the following semantics.

$\mathsf{Setup}(1^\lambda, 1^n, 1^z, 1^q) \to (\mathsf{mpk}, \mathsf{msk})$. 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 $\mathcal{I}_z$.

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

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

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

Definition 3.3 (Correctness)**.** We say the scheme is correct if for all $\lambda, n \in \mathbb{N}, z, q \in \mathsf{poly}(\lambda)$, functions $f \in \mathcal{F}_n$, messages $m \in \mathcal{M}_n$ and $\mathsf{tag} \in \mathcal{I}_z$, we have that for $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, 1^z, 1^q)$, the following holds true,

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

where the probability is taken over the coins of setup, key generation and encryption algorithms.

Definition 3.4 (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 $(f_i, \mathsf{tag}_i, \mu_i)$, where the function value is either $\mu_i = \perp$ (signalling that the adversary has not yet made any encryption query on tag tag_i), or $(m^{\mathsf{tag}_i}, \mathsf{tag}_i)$ has already been queried for encryption (for some message m^{tag_i}), and $\mu_i = f_i(m^{\mathsf{tag}_i})$, the oracle outputs a simulated key $\mathsf{sk}_{f_i,\mathsf{tag}_i}$.

$\mathsf{S}_2(\cdot,\cdot)$, upon a call to generate ciphertext on a tag-list tuple $(\mathsf{tag}_i, \Pi^{m^{\mathsf{tag}_i}})$, where the list $\Pi^{m^{\mathsf{tag}_i}}$ is a possibly empty list of the form $\Pi^{m^{\mathsf{tag}_i}} = (f_1^{\mathsf{tag}_i}, f_1^{\mathsf{tag}_i}(m^{\mathsf{tag}_i}))$, $\ldots, (f_{q_{\mathsf{pre}}}^{\mathsf{tag}_i}, f_{q_{\mathsf{pre}}}^{\mathsf{tag}_i}(m^{\mathsf{tag}_i}))$ (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 $\mathsf{ct}_{\mathsf{tag}_i}$.

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

Upon a ciphertext query on a message-tag pair (m_i, tag_i), if the adversary made an encryption query on the same tag tag_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 $\Pi^{m_i} = \left((f_1^{\mathsf{tag}_i}, f_1^{\mathsf{tag}_i}(m^{\mathsf{tag}_i})), \ldots, (f_{q_{\mathsf{pre}}}^{\mathsf{tag}_i}, f_{q_{\mathsf{pre}}}^{\mathsf{tag}_i}(m^{\mathsf{tag}_i}))\right)$ where $(f_j^{\mathsf{tag}_i}, \mathsf{tag}_i)$ are stored in $\mathcal{E}$'s local state, and removes all such pairs $(f_j^{\mathsf{tag}_i}, \mathsf{tag}_i)$ from its local state. $\mathcal{E}$ then queries ciphertext oracle S_2 on tuple $(\mathsf{tag}_i, \Pi^{m_i})$, and simply forwards oracle's simulated ciphertext $\mathsf{ct}_{\mathsf{tag}_i}$ 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 tag_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 tag_i such that it also submitted an encryption query for tag tag_i.

4 From Static FE to Tagged FE Generically

Our tagged FE Accumulator is a generic compiler which can be used to combine 2^z many instances of a functional encryption scheme to a tagged functional encryption scheme with tag space $\mathcal{I}_z = \{0,1\}^z$ (see Section 3.3 for a definition). Our compiler preserves the bound on the number of Keygen queries that can be made on a single tag, i.e. if we start with a q-bounded functional encryption scheme where $q = \mathsf{poly}(\lambda)$, we get a q-bounded tagged functional encryption scheme. Additionally, our compiler can support uniform models of computation.

The construction and security proof are similar to the IBE construction of [DG17a] and builds on techniques from [DG17b, DG17a, BLSV18, GHMR18]. In the aforementioned list of works, the delegation trick is done using a variety of primitives – chameleon hash functions, one time signatures with encryption (OTSE), batch encryption, hash garbling, etc. We build our tagged FE using the notion of compact One-Time Signature with Encryption (OTSE). IBE implies compact OTSE [DG17a] and we use the weaker primitive of compact OTSE here to make our transformation more general. And, indeed OTSE along with public key encryption gives an IBE scheme [DG17a]. Additionally, in [DGHM18], the authors showed how to build compact OTSE from public key encryption and hash encryption. Thus compact OTSE can be built from a variety of primitives such as Learning with Errors (LWE), PKE with Learning Parity with Noise (LPN) [DGHM18] or Computational Diffie-Hellman (CDH) [DG17a].

4.1 Construction

Let $\mathsf{BFE} = (\mathsf{BFE.Setup}, \mathsf{BFE.KeyGen}, \mathsf{BFE.Enc}, \mathsf{BFE.Dec})$ be a q-bounded functional encryption scheme for $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ a family of functions and message space $\mathcal{M} = \{0,1\}^*$, let $\mathsf{OTSE} = (\mathsf{SSetup}, \mathsf{SGen}, \mathsf{SSign}, \mathsf{SEnc}, \mathsf{SDec})$ be a One Time Signature with Encryption scheme. Note that $\mathcal{F}$ can either follow a uniform/non-uniform model of computation and our tagged construction would hold on the same functionality $\mathcal{F}$. Let PRF be a pseudorandom function with key size λ and can take in inputs of less than z bits and outputs two pseudorandom strings PRF_1 and PRF_2 such that $\mathsf{PRF}(s \in \{0,1\}^\lambda, v \in \{0,1\}^{\leq z}) = \mathsf{PRF}_1(s,v)||\mathsf{PRF}_2(s,v)$ where $|\mathsf{PRF}_1(s,v)|$ is the length of the randomness used for the SSetup alorithm an $|\mathsf{PRF}_2(s,v)|$ is the length of the randomness used for the $\mathsf{BFE.Setup}$ algorithm.

We construct a q-bounded tagged functional encryption scheme on tag space $\mathcal{I}_z = \{0,1\}^z$, which combines $|\mathcal{I}_z|$ many instances of a q-bounded functional encryption scheme.

$\mathsf{Setup}(1^\lambda, 1^n, 1^z, 1^q)$.

1. Let $\ell = \mathsf{poly}(\lambda)$ be the length of the $\mathsf{BFE.mpk}$ when called on security parameter λ. Let $\mathsf{pp} \leftarrow \mathsf{SSetup}(1^\lambda, \ell)$. [4]
2. Let $s \leftarrow \{0,1\}^\lambda$ be the PFF seed.
3. Run $(\mathsf{vk}_\epsilon, \sigma_\epsilon, x_\epsilon) \leftarrow \mathsf{NodeGen}(\mathsf{pp}, \epsilon, s)$
4. Output $\mathsf{mpk} = (\mathsf{pp}, \mathsf{vk}_\epsilon), \mathsf{msk} = s$.

$\mathsf{KeyGen}(\mathsf{msk} = s, \mathsf{tag} \in \{0,1\}^z, f \in \mathcal{F}_n)$

[4] Recall from the succinctness property of OTSE, the length of the verification key $|\mathsf{vk}|$ is some polynomial in λ and independent of ℓ. We assume we can choose the security parameter of $\mathsf{BFE.Setup}$ by some polynomial factor such that $\ell \geq 2|\mathsf{vk}|$.

$\mathsf{LeafGen}(\mathsf{pp}, v, s, f)$

Inputs: OTSE public parameters pp; Leaf Index $v \in \{0,1\}^z$;
PRF Seed $s \in \{0,1\}^\lambda$; Function $f \in \mathcal{F}_n$

Output: OTSE verification key vk_v; OTSE signature σ_v;
Public Key of v^{th} instance $\mathsf{BFE.mpk}_v$; Secret Key $\mathsf{BFE.sk}_f$

1. $(\mathsf{vk}_v, \mathsf{sk}_v) \leftarrow \mathsf{SGen}(\mathsf{pp}, \mathsf{PRF}_1(s, v))$.
2. $(\mathsf{BFE.mpk}_v, \mathsf{BFE.msk}_v) \leftarrow \mathsf{BFE.Setup}(1^\lambda; \mathsf{PRF}_2(s, v))$, i.e. compute the v^{th} instance of the base scheme by running the setup on randomness $\mathsf{PRF}_s(s, v)$.
3. $\mathsf{BFE.sk}_f \leftarrow \mathsf{BFE.KeyGen}(\mathsf{BFE.msk}_v, f)$
4. $\sigma_v = \mathsf{SSign}(\mathsf{pp}, \mathsf{sk}_v, \mathsf{BFE.mpk}_v)$
5. Output $(\mathsf{vk}_v, \sigma_v, \mathsf{BFE.mpk}_v, \mathsf{BFE.sk}_f)$

Figure 1: Routine $\mathsf{LeafGen}$

$\mathsf{NodeGen}(pp, v, s)$

Inputs: OTSE public parameters pp; Node Index $v \in \{0,1\}^{z'}$ where $z' < z$;
PRF Seed $s \in \{0,1\}^\lambda$

Output: OTSE verification key vk_v; OTSE signature σ_v; Auxiliary value x_v

1. $(\mathsf{vk}_v, \mathsf{sk}_v) \leftarrow \mathsf{SGen}(\mathsf{pp}, \mathsf{PRF}_1(s, v))$, $(\mathsf{vk}_{v||0}, \mathsf{sk}_{v||0}) \leftarrow \mathsf{SGen}(\mathsf{pp}, \mathsf{PRF}_1(s, v||0))$ and $(\mathsf{vk}_{v||1}, \mathsf{sk}_{v||1}) \leftarrow \mathsf{SGen}(\mathsf{pp}, \mathsf{PRF}_1(s, v||1))$.
2. $x_v = \mathsf{vk}_{v||0}||vk_{v||1}$.
3. $\sigma_v = \mathsf{SSign}(\mathsf{pp}, \mathsf{sk}_v, x_v)$.
4. Output $(\mathsf{vk}_v, \sigma_v, x_v)$

Figure 2: Routine $\mathsf{NodeGen}$

Leaf Encryption Circuit $\mathsf{T}[m](\mathsf{BFE.mpk})$

Inputs: Public key of leaf $\mathsf{BFE.mpk}$; **Constants:** Message $m \in \mathcal{M}_n$;
Output: Ciphertext ct

1. Compute and output $\mathsf{BFE.Enc}(\mathsf{BFE.mpk}, m)$.

Figure 3: Circuit T

1. For $j \in [0, z]$, let v_j denote a prefix of tag, i.e. first j bits of tag. Note that v_0 is ϵ and $v_z = \mathsf{tag}$.
2. For $j \in [0, z-1]$, compute $(\mathsf{vk}_{v_j}, \sigma_{v_j}, x_{v_j}) \leftarrow \mathsf{NodeGen}(\mathsf{pp}, v_j, s)$.
3. Let $(\mathsf{vk}_{\mathsf{tag}}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}, \mathsf{BFE.sk}_{\mathsf{tag},f}) \leftarrow \mathsf{LeafGen}(\mathsf{pp}, \mathsf{tag}, s)$.

Garbled Label Encryption Circuit $\mathsf{Q}[\mathsf{pp}, \beta, \ell', \mathsf{e}](\mathsf{vk})$

Inputs: OTSE verification key vk

Constants: OTSE public parameters pp; Bit $\beta \in \{0,1\}$; Number of circuit labels $\ell' \in \mathbb{N}$; Labels of a garbled circuit $\mathsf{e} = \{(Y_{\iota,0}, Y_{\iota,1})\}_{\iota \in [\ell']}$

Output: Encrypted Labels, $\hat{\mathsf{e}}^{\beta}$

1. Compute and output $\{\mathsf{SEnc}(\mathsf{pp}, (\mathsf{vk}, \beta \cdot \ell' + \iota, b), Y_{\iota,b})\}_{\iota \in [\ell'], b \in \{0,1\}}$.

Figure 4: Circuit Q

4. Output $\left(\{(\sigma_{v_j}, x_{v_j})\}_{j \in [0, z-1]}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}, \mathsf{BFE.sk}_{\mathsf{tag},f}\right)$.

$\mathsf{Enc}(\mathsf{mpk} = (\mathsf{pp}, \mathsf{vk}_\epsilon), \mathsf{tag} \in \{0,1\}^z, m \in \mathcal{M}_n) \to \mathsf{ct}$.

1. For $j \in [z]$, let v_j denote the prefix of tag and ℓ be the length of $\mathsf{BFE.mpk}$.
2. $(\tilde{\mathsf{T}}, \mathsf{e}_{\mathsf{T}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{T}[m])$ (Figure 3).
3. Let $(\tilde{\mathsf{Q}}^{(z)}, \mathsf{e}_{\mathsf{Q}}^{(z)}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Q}[\mathsf{pp}, 0, \ell, \mathsf{e}_{\mathsf{T}}])$ (Figure 4).
4. For $j = z-1, \ldots, 0$, let ℓ' be $|\mathsf{vk}_\epsilon|$,
   (a) $(\tilde{\mathsf{Q}}^{(j)}, \mathsf{e}_{\mathsf{Q}}^{(j)}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Q}[\mathsf{pp}, \mathsf{tag}_{j+1}, \ell', \mathsf{e}_{\mathsf{Q}}^{(j+1)}])$ (note that $\mathsf{Q}^{(j+1)}$ consists of labels corresponding to the verification key).
5. Parse $\mathsf{e}_{\mathsf{Q}}^{(0)} = \{Y_{\iota,0}, Y_{\iota,1}\}_{\iota \in [\ell']}$.
6. Let y_ι denote the ι^{th} bit of vk_ϵ. Let $\tilde{\mathsf{y}}^{(0)} \leftarrow \{Y_{\iota,\mathsf{y}_\iota}\}_{\iota \in [\ell']}$.
7. Output $\left(\tilde{\mathsf{y}}^{(0)}, \tilde{\mathsf{Q}}^{(0)}, \ldots, \tilde{\mathsf{Q}}^{(z)}, \tilde{\mathsf{T}}\right)$.

$\mathsf{Dec}(\mathsf{sk}_{\mathsf{tag},f}, \mathsf{ct}) \to \mathcal{R}_n$.

1. Parse $\mathsf{sk}_{\mathsf{tag},f} = \left(\{(\sigma_{v_j}, x_{v_j})\}_{j \in [0, z-1]}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}, \mathsf{BFE.sk}_{\mathsf{tag},f}\right)$.
2. Parse $\mathsf{ct} = \left(\tilde{\mathsf{y}}^{(0)}, \tilde{\mathsf{Q}}^{(0)}, \ldots, \tilde{\mathsf{Q}}^{(z)}, \tilde{\mathsf{T}}\right)$.
3. Recall, for $j \in [z]$, let v_j denote the prefix of tag, ℓ be the length of $\mathsf{BFE.mpk}$ and ℓ' be length of vk for otse scheme called on λ, ℓ.
4. For $j \in [0, z-1]$
   (a) $\{\hat{\mathsf{e}}_{\iota,b}^{(j)}\}_{\iota \in [\ell'], b \in \{0,1\}} \leftarrow \mathsf{GC.Eval}(\tilde{\mathsf{Q}}^{(j)}, \tilde{\mathsf{y}}^{(j)})$.
   (b) $\left\{\tilde{\mathsf{y}}^{(j+1)}\right\} \leftarrow \left\{\mathsf{SDec}\left(\mathsf{pp}, \left(\mathsf{vk}_{v_j}, \sigma_{v_j}, x_{v_j}\right), \hat{\mathsf{e}}_{\iota,(x_{v_j})_\iota}^{(j)}\right)\right\}_{\iota \in [\ell']}$
5. $\{\hat{\mathsf{e}}_{\iota,b}^{(z)}\}_{\iota \in [\ell], b \in \{0,1\}} \leftarrow \mathsf{GC.Eval}(\tilde{\mathsf{Q}}^{(z)}, \tilde{\mathsf{y}}^{(z)})$.
6. Let pk_ι denote the ι^{th} bit of $\mathsf{BFE.mpk}_{\mathsf{tag}}$.
7. Let $\mathsf{y}^{\mathsf{T}} = \{\mathsf{SDec}\left(\mathsf{pp}, \left(\mathsf{vk}_{\mathsf{tag}}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}\right), \hat{\mathsf{e}}_{\iota,\mathsf{pk}_\iota}^{(z)}\right)\}_{\iota \in [\ell]}$.
8. Let $\mathsf{ct}_{\mathsf{BFE}} = \mathsf{Eval}(\tilde{\mathsf{T}}, \mathsf{y}^{\mathsf{T}})$. Output $\mathsf{BFE.Dec}(\mathsf{BFE.sk}_{\mathsf{tag},f}, \mathsf{ct}_{\mathsf{BFE}})$.

4.2 Correctness

We say the scheme is correct if it satisfies Definition 3.3. By correctness of the garbling scheme, we have that $\tilde{\mathsf{Q}}^{(0)}$, when run on garbled input $\tilde{\mathsf{y}}^{(0)}$ computes $Q[\mathsf{pp}, 0, \ell, \cdot](\mathsf{vk}_\epsilon)$ and outputs encrypted labels to $\tilde{\mathsf{Q}}^{(1)}$. By correctness of the OTSE encryption scheme, we can decrypt the labels corresponding to tag_1 to compute $\tilde{y}^{(1)}$. Similarly, iteratively calling the security of the garbled circuit and the signature scheme, we compute the garbled inputs to circuit $\tilde{\mathsf{T}}$ corresponding to $\mathsf{BFE.mpk}_{\mathsf{tag}}$. Thus we finally compute $\mathsf{ct}_{\mathsf{BFE}} \leftarrow \mathsf{T}[m](\mathsf{BFE.mpk}_{\mathsf{tag}})$ by the correctness of the garbled circuit algorithm. Finally, we run,

$$\mathsf{BFE.Dec}(\mathsf{BFE.sk}_{\mathsf{tag},f}, \mathsf{BFE.Enc}(\mathsf{BFE.mpk}_{\mathsf{tag}}, m)) = f(m),$$

by the correctness of the tag^{th} instance of BFE scheme.

4.3 Efficiency

The different algorithms $\mathsf{Setup}, \mathsf{KeyGen}$ run polynomial in λ, z, n, q and this can be seen easily from the construction. Encryption algorihm runs polynomial in λ, z, n, q and the message m that is chosen during encryption time. Crucially, we observe here that our tagged FE accumulator is agnostic to the model of computation of the base BFE scheme. Consider a uniform model of computation where the encryption algorithm is a family of circuits, different for each input. The plaintext can be in $\{0,1\}^*$ and the key generation algorithm can take in a TM description. The description of our algorithm Enc garbles a circuit T which computes the encryption of one instance of BFE scheme on a hardcoded message $m \in \{0,1\}^*$. Since during encrypt, we know the message, we can hardcode the circuit at m such that the resulting encryption procedure can be described as a circuit $\mathsf{T}[m]$. Thus if the base scheme BFE can support uniform models of computation, so can our transformation.

4.4 Security

Theorem 4.1. If PRF is a secure pseudorandom function, GC is a secure garbling scheme, OTSE is a secure one-time signature with encryption scheme, $\mathsf{BFE} = (\mathsf{BFE.Setup}, \mathsf{BFE.Enc}, \mathsf{BFE.KeyGen}, \mathsf{BFE.Dec})$ is a bounded-collusion simulation-secure FE scheme (as per Definition 3.1), then the above scheme is a tagged-statically-bounded-collusion simulation-secure FE scheme (as per Definition 3.4).

Proof. Let $\mathsf{BFE.Sim} = (\mathsf{BFE.S}_0, \mathsf{BFE.S}_1, \mathsf{BFE.S}_2)$ be the simulators satisfying Definition 3.1. We will construct simulators $\mathsf{Sim} = (\mathsf{S}_0, \mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3)$ that share a global state st and use the simulated routines Figure 5 and Figure 6 that satisfy Definition 3.4 as follows.

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

Sample $\mathsf{pp} \leftarrow \mathsf{SSetup}(1^\lambda, \ell)$. Run $(\mathsf{vk}_\epsilon, \mathsf{sk}_\epsilon, x_\epsilon) \leftarrow \mathsf{Sim.NodeGen}(\mathsf{pp}, \epsilon, \mathsf{st})$. Output $\mathsf{mpk} = (\mathsf{pp}, \mathsf{vk}_\epsilon)$.

$\mathsf{S}_1(\mathsf{tag}, f, \mu)$

1. For $j \in [0, z]$, let v_j denote the prefix of tag, i.e. first j bits of tag. Note that v_0 is ϵ and $v_z = \mathsf{tag}$.
2. For $j \in [0, z-1]$, compute $(\mathsf{vk}_{v_j}, \sigma_{v_j}, x_{v_j}) \leftarrow \mathsf{Sim.NodeGen}(\mathsf{pp}, v_j, \mathsf{st})$.
3. Let $(\mathsf{vk}_{\mathsf{tag}}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}, \mathsf{BFE.sk}_{\mathsf{tag},f}) \leftarrow \mathsf{Sim.LeafGen}(\mathsf{pp}, \mathsf{tag}, \mathsf{st}, f, \mu)$.

$\mathsf{Sim.LeafGen}(\mathsf{pp}, v, \mathsf{st}, f, \mu)$

Inputs: OTSE public parameters pp; Leaf Index $v \in \{0,1\}^z$; Simulator State $\mathsf{st} \in \{0,1\}^*$; Function $f \in \mathcal{F}_n$; Function Values μ

Output: OTSE verification key vk_v; OTSE signature σ_v; Public Key of v^{th} instance $\mathsf{BFE.mpk}_v$; Secret Key $\mathsf{BFE.sk}_f$

1. If $(v, \mathsf{vk}_v, \mathsf{sk}_v)$ does not exist in st, then $(\mathsf{vk}_v, \mathsf{sk}_v) \leftarrow \mathsf{SGen}(\mathsf{pp})$. Add $(v, \mathsf{vk}_v, \mathsf{sk}_v)$ to st.
2. Let $(\mathsf{BFE.mpk}_v, \mathsf{BFE}^v.\mathsf{st}_0) \leftarrow \mathsf{BFE.S}_0(1^\lambda, 1^n, q)$.
3. If $\mu = \perp$, run $\mathsf{BFE.sk}_f \leftarrow \mathsf{BFE.S}_1(\mathsf{BFE}^v.\mathsf{st}_0, f)$ and update state $\mathsf{BFE}^v.\mathsf{st}_1$ in st. Else, get $\mathsf{BFE.st}_2^v$ from st and run $\mathsf{BFE.sk}_f \leftarrow \mathsf{BFE.S}_3^\mu(\mathsf{BFE}^v.\mathsf{st}_2, f)$ where μ contains the list of all previous function evaluations.
4. $\sigma_v = \mathsf{SSign}(\mathsf{pp}, \mathsf{sk}_v, \mathsf{BFE.mpk}_v)$.
5. Output $(\mathsf{vk}_v, \sigma_v, \mathsf{BFE.mpk}_v, \mathsf{BFE.sk}_f)$.

Figure 5: Routine Simulated LeafGen

$\mathsf{Sim.NodeGen}(\mathsf{pp}, v, \mathsf{st})$

Inputs: OTSE public parameters pp; Node Index $v \in \{0,1\}^{z'}$ where $z' < z$; Simulator state $\mathsf{st} \in \{0,1\}^*$

Output: OTSE verification key vk_v; OTSE signature σ_v; Auxiliary value x_v

1. If $(v, \mathsf{vk}_v, \mathsf{sk}_v)$ does not exist in st, then $(\mathsf{vk}_v, \mathsf{sk}_v) \leftarrow \mathsf{SGen}(\mathsf{pp})$. Add $(v, \mathsf{vk}_v, \mathsf{sk}_v)$ to st. Similarly, set/check the parameters in st for $v||0$ and $v||1$.
2. Let $x_v = \mathsf{vk}_{v||0}||vk_{v||1}$.
3. $\sigma_v = \mathsf{SSign}(\mathsf{pp}, \mathsf{sk}_v, x_v)$.
4. Output $(\mathsf{vk}_v, \sigma_v, x_v)$.

Figure 6: Routine Simulated NodeGen

Simulated Garbled Label Encryption Circuit $\mathsf{Sim.Q}[\mathsf{pp}, \beta, \ell', \mathsf{y}](\mathsf{vk})$

Inputs: OTSE verification key vk

Constants: OTSE public parameters pp; Bit $\beta \in \{0,1\}$; Number of circuit labels $\ell' \in \mathbb{N}$; Labels of a garbled circuit $\mathsf{y} = \{(Y_\iota)\}_{\iota \in [\ell']}$

Output: Encrypted Labels, $\hat{\mathsf{e}}^\beta$

1. Compute and output $\{\mathsf{SEnc}(\mathsf{pp}, (\mathsf{vk}, \beta \cdot \ell' + \iota, b), Y_\iota)\}_{\iota \in [\ell'], b \in \{0,1\}}$.

Figure 7: Simulation Circuit Sim.Q

4. Output $\left(\{(\sigma_{v_j}, x_{v_j})\}_{j \in [0, z-1]}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}, \mathsf{BFE.sk}_{\mathsf{tag},f}\right)$.

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

1. For $j \in [z]$, let v_j denote the prefix of tag and ℓ be the length of $\mathsf{BFE.mpk}$.
2. If $\mathsf{BFE}^{\mathsf{tag}^*}.\mathsf{st}_1$ is not in st, setup $(\mathsf{BFE.mpk}_{\mathsf{tag}^*}, \mathsf{BFE}^{\mathsf{tag}^*}.\mathsf{st}_0) \leftarrow \mathsf{BFE.S}_0(1^\lambda, 1^n, q)$. Let $\mathsf{BFE}^{\mathsf{tag}^*}.\mathsf{st}_1 = \mathsf{BFE}^{\mathsf{tag}^*}.\mathsf{st}_0$. Run $\mathsf{ct}^*_{\mathsf{BFE}} \leftarrow \mathsf{BFE.S}_2(\mathsf{BFE}^{\mathsf{tag}^*}.\mathsf{st}_1, \Pi^{m^{\mathsf{tag}^*}})$. Store $\mathsf{BFE}^{\mathsf{tag}^*}.\mathsf{st}_2$ in st. ($\Pi^{m^{\mathsf{tag}^*}}$ contains the functions and their evaluations at m^{tag^*} for the tag tag^*).
3. $(\tilde{\mathsf{T}}, \tilde{\mathsf{y}}^{\mathsf{T}}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^\ell, 1^{|\mathsf{T}[0]|}, \mathsf{ct}^*_{\mathsf{BFE}})$ (Figure 3)[5].
4. For $j \in [0, z]$, let v_j denote the prefix, i.e. first j bits of tag. If $(v_j, \mathsf{vk}_{v_j}, \mathsf{sk}_{v_j})$ does not exist in st, then $(\mathsf{vk}_{v_j}, \mathsf{sk}_{v_j}) \leftarrow \mathsf{SGen}(\mathsf{pp})$. Add $(v_j, \mathsf{vk}_{v_j}, \mathsf{sk}_{v_j})$ to st.
5. Compute $\hat{\mathsf{y}}^{\mathsf{T}} \leftarrow \mathsf{Sim.Q}[\mathsf{pp}, 0, \ell, \tilde{\mathsf{y}}^{\mathsf{T}}](\mathsf{vk}_{\mathsf{tag}^*})$.
6. Let $(\tilde{\mathsf{Q}}^{(z)}, \tilde{\mathsf{y}}^{(z)}_{\mathsf{Q}}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^\ell, 1^{|\mathsf{Q}|}, \hat{\mathsf{y}}^{\mathsf{T}})$ (Figure 4).
7. For $j = z-1, \ldots, 0$, let ℓ' be $|\mathsf{vk}_\epsilon|$,
   (a) Compute $\hat{\mathsf{y}}^{(j)}_{\mathsf{Q}} \leftarrow \mathsf{Sim.Q}[\mathsf{pp}, \mathsf{tag}_{j+1}, \ell', \tilde{\mathsf{y}}^{(j+1)}_{\mathsf{Q}}](\mathsf{vk}_{v_j})$.
   (b) Let $(\tilde{\mathsf{Q}}^{(j)}, \tilde{\mathsf{y}}^{(j)}_{\mathsf{Q}}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell'}, 1^{|\mathsf{Q}|}, \hat{\mathsf{y}}^{(j)}_{\mathsf{Q}})$ (Figure 4).
8. Output $\left(\tilde{\mathsf{y}}^{(0)}, \tilde{\mathsf{Q}}^{(0)}, \ldots, \tilde{\mathsf{Q}}^{(z)}, \tilde{\mathsf{T}}\right)$.

We will show through a sequence of experiments that the real and simulated games of Theorem 4.1 are computationally indistinguishable. We only note down the steps that are different from the previous experiment in red, rest of the operations and notations remain the same.

Experiment 0. This is the experiment with adversary $\mathcal{A}$ and tagged-statically-bounded-collusion 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:** $(1^n, 1^q, 1^z) \leftarrow \mathcal{A}(1^\lambda)$.
  1. Let $\mathsf{pp} \leftarrow \mathsf{SSetup}(1^\lambda, \ell)$ (ℓ is length of $\mathsf{BFE.mpk}$ called on securtiy parameter λ, functionality index n and collusion bound q).
  2. Sample $s \leftarrow \{0,1\}^\lambda$ as PRF seed.
  3. Compute $(\mathsf{vk}_\epsilon, \sigma_\epsilon, x_\epsilon) \leftarrow \mathsf{NodeGen}(\mathsf{pp}, \epsilon, s)$.
  4. Set $\mathsf{mpk} = (\mathsf{pp}, \mathsf{vk}_\epsilon)$, $\mathsf{msk} = s$.

  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 $\mathsf{tag} \in \{0,1\}^z$ and function f.
  1. For $j \in [0, z]$, let v_j denote the prefix of tag, i.e. first j bits of tag. Note that v_0 is ϵ and $v_z = \mathsf{tag}$.

[5] Any hardcoded public message $\in \mathcal{M}_n$ will work as input to T. Here we set it to 0 for simplicity. In future simulation algorithms, for notational simplicity we omit the hardcoded values and just mention the size as $1^{|\mathsf{T}|}$ or $1^{|\mathsf{Q}|}$. Assume that the size can be set appropriately as a polynomial in λ, ℓ, ℓ'.

2. For $j \in [0, z-1]$, compute $(\mathsf{vk}_{v_j}, \sigma_{v_j}, x_{v_j}) \leftarrow \mathsf{NodeGen}(\mathsf{pp}, v_j, s)$.
3. Let $(\mathsf{vk}_{\mathsf{tag}}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}, \mathsf{BFE.sk}_{\mathsf{tag},f}) \leftarrow \mathsf{LeafGen}(\mathsf{pp}, \mathsf{tag}, s)$.
4. Output $\left(\{(\sigma_{v_j}, x_{v_j})\}_{j\in[0,z-1]}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}, \mathsf{BFE.sk}_{\mathsf{tag},f}\right)$.

- **Ciphertext Queries:** Let $\mathcal{A}$ query Enc on tag $\mathsf{tag}^* \in \mathcal{I}_z$ and message $m^{\mathsf{tag}^*} \in \mathcal{M}_n$.
  1. $(\tilde{\mathsf{T}}, \mathsf{e}_{\mathsf{T}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{T}[m])$ (Figure 3).
  2. Let $(\tilde{\mathsf{Q}}^{(z)}, \mathsf{e}_{\mathsf{Q}}^{(z)}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Q}[\mathsf{pp}, 0, \ell, \mathsf{e}_{\mathsf{T}}])$ (Figure 4).
  3. For $j = z-1, \ldots, 0$, let ℓ' be $|\mathsf{vk}_\epsilon|$,
     (a) $(\tilde{\mathsf{Q}}^{(j)}, \mathsf{e}_{\mathsf{Q}}^{(j)}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Q}[\mathsf{pp}, \mathsf{tag}^*_{j+1}, \ell', \mathsf{e}_{\mathsf{Q}}^{(j+1)}])$ (note that $\mathsf{Q}^{(j+1)}$ consists of labels corresponding to the verification key).
  4. Parse $\mathsf{e}_{\mathsf{Q}}^{(0)} = \{Y_{\iota,0}, Y_{\iota,1}\}_{\iota\in[\ell']}$.
  5. Let y_ι denote the ι^{th} bit of vk_ϵ. Let $\tilde{\mathsf{y}}^{(0)} \leftarrow \{Y_{\iota,\mathsf{y}_\iota}\}_{\iota\in[\ell']}$.
  6. Output $\left(\tilde{\mathsf{y}}^{(0)}, \tilde{\mathsf{Q}}^{(0)}, \ldots, \tilde{\mathsf{Q}}^{(z)}, \tilde{\mathsf{T}}\right)$.
- $\mathcal{A}$ outputs a bit b.

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

Experiment 1. In this experiment, we replace the PRF function with a truly random function. We additionally describe a routine that partially simulates $\mathsf{LeafGen}$.

$\mathsf{PartSim.LeafGen}(\mathsf{pp}, v, \mathsf{st}, f)$

Inputs: OTSE public parameters pp; Leaf Index $v \in \{0,1\}^z$; Simulator State $\mathsf{st} \in \{0,1\}^*$; Function $f \in \mathcal{F}_n$

Output: OTSE verification key vk_v; OTSE signature σ_v; Public Key of v^{th} instance $\mathsf{BFE.mpk}_v$; Secret Key $\mathsf{BFE.sk}_f$

1. If $(v, \mathsf{vk}_v, \mathsf{sk}_v)$ does not exist in st, then $(\mathsf{vk}_v, \mathsf{sk}_v) \leftarrow \mathsf{SGen}(\mathsf{pp})$. Add $(v, \mathsf{vk}_v, \mathsf{sk}_v)$ to st.
2. Let $(\mathsf{BFE.mpk}_v, \mathsf{BFE.msk}_v) \leftarrow \mathsf{BFE.Setup}(1^\lambda, 1^n, q)$. Store $(v, \mathsf{BFE.mpk}_v, \mathsf{BFE.msk}_v)$ in the state st.
3. $\mathsf{BFE.sk}_f \leftarrow \mathsf{BFE.KeyGen}(\mathsf{BFE.msk}_v, f)$
4. $\sigma_v = \mathsf{SSign}(\mathsf{pp}, \mathsf{sk}_v, \mathsf{BFE.mpk}_v)$
5. Output $(\mathsf{vk}_v, \sigma_v, \mathsf{BFE.mpk}_v, \mathsf{BFE.sk}_f)$

Figure 8: Routine Partially Simulated $\mathsf{LeafGen}$

- **Setup:** $(1^n, 1^q, 1^z) \leftarrow \mathcal{A}(1^\lambda)$.
  ~~Sample $s \leftarrow \{0,1\}^\lambda$ as PRF seed.~~
  Let st be the global state. Compute $(\mathsf{vk}_\epsilon, \sigma_\epsilon, x_\epsilon) \leftarrow \mathsf{Sim.NodeGen}(\mathsf{pp}, \epsilon, \mathsf{st})$.
- **Key Queries:** Let $\mathcal{A}$ query KeyGen on tag $\mathsf{tag} \in \{0,1\}^z$ and function f.
  For $j \in [0, z-1]$, compute $(\mathsf{vk}_{v_j}, \sigma_{v_j}, x_{v_j}) \leftarrow \mathsf{Sim.NodeGen}(\mathsf{pp}, v_j, \mathsf{st})$.
  Let $(\mathsf{vk}_{\mathsf{tag}}, \sigma_{\mathsf{tag}}, \mathsf{BFE.mpk}_{\mathsf{tag}}, \mathsf{BFE.sk}_{\mathsf{tag},f}) \leftarrow \mathsf{PartSim.LeafGen}(\mathsf{pp}, \mathsf{tag}, \mathsf{st})$.

Experiment $2k$**.** Here k varies from 1 to z. In this experiment, we change how $\tilde{\mathsf{Q}}^{(k-1)}$ and the garbled labels $\tilde{\mathsf{y}}^{(k-1)}$ are computed by calling the garbled circuit simulator.
Ciphertext Queries: Let $\mathcal{A}$ query Enc on tag $\mathsf{tag}^* \in \mathcal{I}_z$ and message $m^{\mathsf{tag}^*} \in \mathcal{M}_n$.

- For $j = z-1, \ldots, k$, let ℓ' be $|\mathsf{vk}_\epsilon|$,
  1. $(\tilde{\mathsf{Q}}^{(j)}, \mathsf{e}_{\mathsf{Q}}^{(j)}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Q}[\mathsf{pp}, \mathsf{tag}^*_{j+1}, \ell', \mathsf{e}_{\mathsf{Q}}^{(j+1)}])$ (note that $\mathsf{Q}^{(j+1)}$ consists of labels corresponding to the verification key).
- For $j = (k-1)$,
  1. Compute $\hat{\mathsf{e}}_{\mathsf{Q}}^{(j)} \leftarrow \mathsf{Q}[\mathsf{pp}, \mathsf{tag}_{j+1}, \ell', \mathsf{e}_{\mathsf{Q}}^{(j+1)}](\mathsf{vk}_{v_j})$.
  2. Let $(\tilde{\mathsf{Q}}^{(j)}, \tilde{\mathsf{y}}_{\mathsf{Q}}^{(j)}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell'}, 1^{|\mathsf{Q}|}, \hat{\mathsf{e}}_{\mathsf{Q}}^{(j)})$.
- For $j = (k-2), \ldots, 0$,
  1. Compute $\hat{\mathsf{y}}_{\mathsf{Q}}^{(j)} \leftarrow \mathsf{Sim.Q}[\mathsf{pp}, \mathsf{tag}_{j+1}, \ell', \tilde{\mathsf{y}}_{\mathsf{Q}}^{(j+1)}](\mathsf{vk}_{v_j})$.
  2. Let $(\tilde{\mathsf{Q}}^{(j)}, \tilde{\mathsf{y}}_{\mathsf{Q}}^{(j)}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell'}, 1^{|\mathsf{Q}|}, \hat{\mathsf{y}}_{\mathsf{Q}}^{(j)})$.

Experiment $2k+1$**.** Here k varies from 1 to z. In this experiment, instead of using Q for encrypting the labels $\mathsf{e}_{\mathsf{Q}}^{(k)}$, instead we use $\mathsf{Sim.Q}$ to compute $\hat{\mathsf{y}}^{(k-1)}$ and use that inside $\mathsf{GC.Sim}$. In the **Ciphertext Queries** phase, we make the following change,
For $j = (k-1)$,

1. Let $\mathsf{e}_{\mathsf{Q}}^{(j+1)}$ be denoted by the set of labels $\{(Y_{\iota,0}, Y_{\iota,1})\}_{\iota \in [\ell']}$. Let y_ι be the ι^{th} bit of $\mathsf{vk}_{v_{j+1}}$. Use $\mathsf{y}_{\mathsf{Q}}^{(j+1)} = \{Y_{\iota,\mathsf{y}_\iota}\}_{\iota \in [\ell']}$ as the labels.
2. Compute $\hat{\mathsf{y}}_{\mathsf{Q}}^{(j)} \leftarrow \mathsf{Sim.Q}[\mathsf{pp}, \mathsf{tag}_{j+1}, \ell', \mathsf{y}_{\mathsf{Q}}^{(j+1)}](\mathsf{vk}_{v_j})$.
3. Let $(\tilde{\mathsf{Q}}^{(j)}, \tilde{\mathsf{y}_{\mathsf{Q}}^{(j)}}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell'}, 1^{|\mathsf{Q}|}, \hat{\mathsf{y}}_{\mathsf{Q}}^{(j)})$.

Experiment $2z+2$**.** In this experiment, we change how $\tilde{\mathsf{Q}}^{(z)}$ and the garbled labels $\tilde{\mathsf{y}}^{(z)}$ are computed by calling the garbled circuit simulator. In the **Ciphertext Queries** phase, we make the following change -
Let $(\tilde{\mathsf{Q}}^{(z)}, \tilde{\mathsf{y}}_{\mathsf{Q}}^{(z)}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell'}, 1^{|\mathsf{Q}|}, \mathsf{e}^{\mathsf{T}})$ and use $\tilde{\mathsf{y}}_{\mathsf{Q}}^{(z)}$ instead of $\mathsf{e}_{\mathsf{Q}}^{(z)}$ inside $\mathsf{Sim.Q}$.

Experiment $2z+3$**.** In this experiment, instead of using Q for encrypting the labels y^{T}, instead we use $\mathsf{Sim.Q}$ to compute $\hat{\mathsf{y}}^{\mathsf{T}}$. In the **Ciphertext Queries** phase, we make the following change - Let e_{T} be denoted by the set of labels $\{(Y_{\iota,0}, Y_{\iota,1})\}_{\iota \in \ell}$. Let y_ι be the ι^{th} bit of $\mathsf{BFE.mpk}_{\mathsf{tag}^*}$. Use $\mathsf{y}_{\mathsf{T}} = \{Y_{\iota,\mathsf{y}_\iota}\}_{\iota \in \ell}$ as the labels. Compute $\hat{\mathsf{y}}^{\mathsf{T}} \leftarrow \mathsf{Sim.Q}[\mathsf{pp}, 0, \ell, \mathsf{y}_{\mathsf{T}}](\mathsf{vk}_{\mathsf{tag}^*})$. Additionally, use $\hat{\mathsf{y}}^{\mathsf{T}}$ instead of e^{T} as the input to the garble simulator algorithm.

Experiment $2z+4$**.** In this experiment, we simulate T using the garbled circuit simulator hard-coded with the $\mathsf{BFE.ct}$. In the **Ciphertext Queries** phase, we make the following change -
Let $\mathsf{ct}^*_{\mathsf{BFE}} \leftarrow \mathsf{BFE.Enc}(\mathsf{BFE.mpk}_{\mathsf{tag}^*}, m)$. Compute $(\tilde{\mathsf{T}}, \tilde{\mathsf{y}}^{\mathsf{T}}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^\ell, 1^{|\mathsf{T}[m]|}, \mathsf{ct}^*_{\mathsf{BFE}})$.

Experiment $2z+5$**.** In this experiment, we simulate $\mathsf{BFE.Enc}$ and $\mathsf{BFE.KeyGen}$ to remove all information about the message m.
In the **Key Queries** phase, use routine $\mathsf{Sim.LeafGen}$ instead of $\mathsf{PartSim.LeafGen}$.
In the **Ciphertext Queries** phase, change how we sample $\mathsf{ct}^*_{\mathsf{BFE}}$ and set it as $\mathsf{ct}^*_{\mathsf{BFE}} \leftarrow \mathsf{BFE.S}_2(\mathsf{BFE}^{\mathsf{tag}^*}.\mathsf{st}_1, \Pi^{m^{\mathsf{tag}^*}})$. ($\Pi^{m^{\mathsf{tag}^*}}$ contains the functions and their evaluations at m^{tag^*} for the tag tag^*).

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 4.1. If PRF is a secure pseudorandom functions, then for every 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. Suppose there existed an adversary $\mathcal{A}$ that distinguishes with some non-negligible probability, then we can construct an adversary $\mathcal{B}$ that distinguishes between the a truly random function and a PRF with non-negligible probability. Experiment 0 consists of computation corresponding to the PRF and Experiment 1 consists of computation corresponding to the truly random function. Thus we can argue the lemma from PRF security. □

Lemma 4.2. If GC is a secure garbling scheme, then for k from 1 to z, for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^{2k-1}_{\mathcal{A}}(\lambda) - \mathcal{P}^{2k}_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. Suppose there existed an adversary $\mathcal{A}$ that distinguishes with some non-negligible probability, then we can construct an adversary $\mathcal{B}$ that distinguishes between the garbling security game with non-negligible probability. Consider the circuit $\tilde{\mathsf{Q}}^{(k-1)}$. In experiment $2k-1$, we compute the circuit using the algorithm $(\tilde{\mathsf{Q}}^{(k-1)}, \mathsf{e}^{(k-1)}_{\mathsf{Q}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Q}[\mathsf{pp}, v_k, \ell', \mathsf{e}^{(k)}_{\mathsf{Q}}])$. In experiment $2k$, we compute the same circuit from a garbled circuit simulator, where we compute $(\tilde{\mathsf{Q}}^{(k-1)}, \tilde{\mathsf{y}}^{(k-1)}_{\mathsf{Q}}) \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell'}, 1^{|\mathsf{Q}|}, \hat{\mathsf{e}}^{(k-1)}_{\mathsf{Q}})$ where the labels $\tilde{\mathsf{y}}^{(k-1)}_{\mathsf{Q}}$ correspond to the labels $\mathsf{vk}_{v_{k-1}}$. Observe that in both games, we only proceed with computation on $\tilde{\mathsf{y}}^{(k-1)}_{\mathsf{Q}}$, specifically in experiment $2k-1$, we compute $\mathsf{e}^{(k-1)}_{\mathsf{Q}}$ and only compute on labels corresponding to $\mathsf{vk}_{v_{k-1}}$. Thus by garbled security on $\mathsf{Q}[\mathsf{pp}, v_k, \ell', \mathsf{e}^{(k)}_{\mathsf{Q}}]$ and input $\mathsf{vk}_{v_{k-1}}$, the advantage of $\mathcal{B}$ is the same as advantage of $\mathcal{A}$. □

Lemma 4.3. If OTSE is a secure one time encryption scheme, then for k from 1 to z, for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}^{2k}_{\mathcal{A}}(\lambda) - \mathcal{P}^{2k+1}_{\mathcal{A}}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. We can assume that our proof proceeds through a sequence of sub-experiments where we switch one challenge ciphertext at a time. The loss in the security experiment will be bounded by the number of queries we can make and thus will be q times the loss between one of the sub-experiments. Assume that we switch the ciphertext on challenge tag tag^*.

Suppose there existed an adversary $\mathcal{A}$ that distinguishes with some non-negligible probability, then we can construct an adversary $\mathcal{B}$ that distinguishes between the OTSE security game with non-negligible probability. Consider the routines $\mathsf{Q}[\mathsf{pp}, \mathsf{tag}_k, \ell', \mathsf{e}^{(k)}_{\mathsf{Q}}](\mathsf{vk}_{v_{k-1}})$ and $\mathsf{Sim.Q}[\mathsf{pp}, \mathsf{tag}_k, \ell', \mathsf{y}^{(k)}_{\mathsf{Q}}](\mathsf{vk}_{v_{k-1}})$ used to compute $\hat{\mathsf{y}}^{(k)}_{\mathsf{Q}}$. Let $\mathsf{e}^{(k)}_{\mathsf{Q}}$ be denoted by the set of labels $\{(Y_{\iota,0}, Y_{\iota,1})\}_{\iota \in [\ell']}$. Let y_ι be the ι^{th} bit of vk_{v_k}. Let $\mathsf{y}^{(j+1)}_{\mathsf{Q}} = \{Y_{\iota,\mathsf{y}_\iota}\}_{\iota \in [\ell']}$ as the labels.

The adversary $\mathcal{B}$ functions as follows, it gets the public parameter pp. It then outputs x^* as the challenge message as $\mathsf{vk}_{v_{k-1}||0}||\mathsf{vk}_{v_{k-1}||1}$. Challenger samples $(\mathsf{vk}_{v_{k-1}}, \mathsf{sk}_{v_{k-1}})$ and sends $(\mathsf{vk}_{v_{k-1}})$

to $\mathcal{B}$. $\mathcal{B}$ obtains the signature $\sigma \leftarrow \mathsf{SSign}(\mathsf{pp}, \mathsf{sk}_{v_{k-1}}, x^*)$. It uses σ in keygen procedure as it does not know $\mathsf{sk}_{v_{k-1}}$. Finally, it outputs $i^* = \{\mathsf{tag}_k \ell' + \iota\}_{\iota \in [\ell']}$ as the challenge locations and sets the multi-message challenge ciphertexts as $M_0^* = \{Y_{\iota,1-\mathsf{y}_\iota}\}_{\iota \in [\ell']}$ and $M_1^* = \{Y_{\iota,\mathsf{y}_\iota}\}_{\iota \in [\ell']}$. The ciphertext set $C_0^* = \{\mathsf{SEnc}(\mathsf{pp}, (\mathsf{vk}_{v_{k-1}}, \mathsf{tag}_k \ell' + \iota, 1-\mathsf{y}_\iota), Y_{\iota,1-\mathsf{y}_\iota})\}_{\iota \in [\ell']}$. Consider the ciphertexts $C_1^* = \{\mathsf{SEnc}(\mathsf{pp}, (\mathsf{vk}_{v_{k-1}}, \mathsf{tag}_k \ell' + \iota, 1-\mathsf{y}_\iota), Y_{\iota,\mathsf{y}_\iota})\}_{\iota \in [\ell']}$. Note that when we switch from using Q to $\mathsf{Sim.Q}$, this is precisely the change, and thus we can rely on selective OTSE security to make our claim. □

Lemma 4.4. If GC is a secure garbling scheme, for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{2z+1}(\lambda) - \mathcal{P}_{\mathcal{A}}^{2z+2}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. This is very similar to the proof of Lemma 4.2. □

Lemma 4.5. If OTSE is a secure one time encryption scheme, for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{2z+2}(\lambda) - \mathcal{P}_{\mathcal{A}}^{2z+3}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. This is very similar to the proof of Lemma 4.3. □

Lemma 4.6. If GC is a secure garbling scheme, for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{2z+3}(\lambda) - \mathcal{P}_{\mathcal{A}}^{2z+4}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. This is very similar to the proof of Lemma 4.2. □

Lemma 4.7. If BFE is a secure bounded-collusion simulation-secure FE scheme scheme, for every adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\mathcal{P}_{\mathcal{A}}^{2z+4}(\lambda) - \mathcal{P}_{\mathcal{A}}^{2z+5}(\lambda)| = \mathsf{negl}(\lambda)$.

Proof. We rely on the security for each instance of the BFE scheme where a keygen or ciphertext query is made to the tagged scheme. Let q' be the total number of BFE instances on which we make any kind of query. We can go through a sequence of q' sub-experiments where we change each scheme to it's simulated counterpart. The formal details are routine, and if the advantage in breaking the BFE scheme is $\mathsf{negl}'(\lambda)$, then the advantage between the two experiments is atmost $q'\mathsf{negl}'(\lambda)$. Since $q = \mathsf{poly}(\lambda)$, the winning probability is bounded by a negligible function $\mathsf{negl}(\lambda)$. □

□

5 Main Theorem and New Results

In this section, we give our main theorem that combines our Theorem 4.1 with the black-box compilers from [GGLW22]. First, we recall the theorem about a black-box transformation from tagged FE to dynamic FE from [GGLW22].

Theorem 5.1 ([GGLW22, Paraphrased, Theorems 3.1 and 5.1])**.** If tgfe is a tagged statically λ-bounded collusion simulation-secure FE scheme (as per Definition 3.4), then there exists a dynamic bounded collusion simulation-secure FE scheme (as per Definition 3.2). And, the dynamic FE scheme can be obtained via a black-box transformation from the tagged FE scheme.

Next, by combining the above theorem with Theorem 4.1, we get the following theorem.

Theorem 5.2. If IBE is a secure IBE scheme and BFE is a λ-bounded collusion simulation-secure FE scheme (as per Definition 3.1), then there exists a dynamic bounded collusion simulation-secure FE scheme (as per Definition 3.2). And, the dynamic FE scheme can be obtained via a non-black-box transformation from the static FE scheme.

We want to point out that although Theorem 4.1 was stated in terms of existence of a PRF, garbling scheme, and an OTSE scheme, we can replace all of them with IBE as IBE implies all of them.

Finally, we show that by combining Theorem 5.2 with [GSW21, Wee21], we get the following.

Corollary 5.1. If IBE is a secure IBE scheme, then there exists a dynamic-bounded collusion simulation-secure ABE scheme for Turing Machines.

Corollary 5.2. If IBE is a secure IBE scheme and Learning with Errors assumption is hard, then there exists a dynamic-bounded collusion simulation-secure ABE scheme for DFAs in the secret-key-selective setting.

References

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

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *CRYPTO*, 2013.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, 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. In *CRYPTO*, 2021.

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

[AS17] Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from learning with errors. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP*, 2017.

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In Dennis Hofheinz and Alon Rosen, editors, *TCC*, 2019.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2001.

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

[BLSV18] Zvika Brakerski, Alex Lombardi, Gil Segev, and Vinod Vaikuntanathan. Anonymous ibe, leakage resilience and circular security from new assumptions. In *EUROCRYPT*, 2018.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. In *TCC*, 2011.

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

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

[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.

[DG17a] Nico Döttling and Sanjam Garg. From selective ibe to full ibe and selective hibe. TCC, 2017.

[DG17b] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *CRYPTO*, 2017.

[DGHM18] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Daniel Masny. New constructions of identity-based and key-dependent message secure encryption schemes. In *IACR International Workshop on Public Key Cryptography*, pages 3–31. Springer, 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.

[GGLW22] Rachit Garg, Rishab Goyal, George Lu, and Brent Waters. Dynamic collusion bounded functional encryption from identity-based encryption. In *(To appear) EUROCRYPT*, 2022. https://ia.cr/2021/847.

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *IACR international workshop on public key cryptography*, pages 63–93. Springer, 2019.

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

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *Annual Cryptology Conference*, pages 536–553. Springer, 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.

[GSW21] Rishab Goyal, Ridwan Syed, and Brent Waters. Bounded collusion abe for tms from ibe. In *ASIACRYPT*, 2021.

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

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

[ISV+17] Gene Itkis, Emily Shen, Mayank Varia, David Wilson, and Arkady Yerukhimovich. Bounded-collusion attribute-based encryption from minimal assumptions. In *IACR International Workshop on Public Key Cryptography*, 2017.

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

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

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, 2005.

[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 *CCS*, 2010.

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

[Wee21] Hoeteck Wee. Abe for dfa from lwe against bounded collusions, revisited. In *TCC*, 2021.

[Yao82] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

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