Functional Encryption for Turing Machines with Dynamic Bounded Collusion from LWE

Shweta Agrawal[*] Monosij Maitra[†] Narasimha Sai Vempati[‡] Shota Yamada[§]

June 22, 2021

Abstract

The classic work of Gorbunov, Vaikuntanathan and Wee (CRYPTO 2012) and follow-ups provided constructions of bounded collusion Functional Encryption (FE) for circuits from mild assumptions. In this work, we improve the state of affairs for bounded collusion FE in several ways:

1. *New Security Notion.* We introduce the notion of *dynamic* bounded collusion FE, where the declaration of collusion bound is delayed to the time of encryption. This enables the encryptor to dynamically choose the collusion bound for different ciphertexts depending on their individual level of sensitivity. Hence, the ciphertext size grows linearly with its own collusion bound and the public key size is independent of collusion bound. In contrast, all prior constructions have public key and ciphertext size that grow at least linearly with a fixed bound Q.

2. *CPFE for circuits with Dynamic Bounded Collusion.* We provide the first CPFE schemes for circuits enjoying dynamic bounded collusion security. By assuming identity based encryption (IBE), we construct CPFE for circuits of *unbounded* size satisfying *non-adaptive* simulation based security. By strengthening the underlying assumption to IBE with receiver selective opening security, we obtain CPFE for circuits of *bounded* size, output length and depth enjoying *adaptive* simulation based security. Moreover, we show that IBE is a necessary assumption for these primitives.
   Moreover, by relying on the Learning With Errors (LWE) assumption, we obtain the first *succinct* CPFE for circuits, i.e. supporting circuits with unbounded size, but fixed output length and depth. This scheme achieves *adaptive* simulation based security.

3. *KPFE for circuits with dynamic bounded collusion.* We provide the first KPFE for circuits of unbounded size, but bounded depth and output length satisfying dynamic bounded collusion security. Our construction relies on LWE and achieves *adaptive* simulation based security. This improves the security of succinct KPFE by Goldwasser et al. [GTKP+13b].

4. *KP and CP FE for TM/NL with dynamic bounded collusion.* We provide the first KPFE and CPFE constructions of bounded collusion functional encryption for Turing machines in the public key setting from LWE. Our constructions achieve non-adaptive simulation based security. Both the input and the machine in our construction can be of *unbounded* polynomial length but the ciphertext size grows with the upper bound on the running time of the Turing machine on the given input. Given RAM access to the ciphertext, the scheme enjoys input specific decryption time.
   We provide a variant of the above scheme that satisfies *adaptive* security, but at the cost of supporting a smaller class of computation, namely Nondeterministic Logarithmic-space (NL). Since NL contains Nondeterministic Finite Automata (NFA), this result subsumes *all* prior work of bounded collusion FE for uniform models from standard assumptions [AMY19, AS17].

[*]IIT Madras, India. Email: shweta.a@cse.iitm.ac.in
[†]TU Darmstadt, Germany. Email: monosij.maitra@tu-darmstadt.de
[‡]IIT Madras, India. Email: narasimhasai07@gmail.com
[§]National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan. Email: yamada-shota@aist.go.jp

Contents

1 Introduction

Functional encryption [SW05, BSW11] is a generalization of public key encryption which allows fine grained control on disclosure of encrypted data. In functional encryption (FE), a secret key is associated with a function f, a ciphertext is associated with an input $\mathbf{x}$ and decryption reveals $f(\mathbf{x})$ and nothing more. Security requires that any collusion of users cannot learn more about the ciphertext beyond what they are individually authorized to learn – this property is known as *collusion resistance*.

In a classic work, Gorbunov, Vaikuntanathan and Wee [GVW12] provided the first construction of Functional Encryption for circuits in the *bounded collusion* model – namely in a model where the scheme may generate an unbounded number of keys but security holds only against an adversary who obtains at most an a-priori bounded number of keys, say Q. Their construction supports all polynomial sized circuits, and is based on the existence of public key encryption (PKE) and pseudorandom generators (PRG) in NC^1. Since circuits are a powerful model of computation, this work provides a strong feasibility result, and moreover, from weak assumptions. Subsequent work provided other useful improvements: the work of Ananth and Vaikuntanathan [AV19] removed the assumption of PRGs so that the resulting scheme relies on the minimal assumption of PKE, while a series of works [AR17, Agr17, AV19] improved the dependence of the public key and ciphertext on the collusion bound Q. A parallel line of work has studied the construction of functional encryption schemes supporting unbounded collusions [Agr19, AJL+19, JLMS19, JLS20, GJLS20], from standard assumptions, culminating in the recent breakthrough of Jain, Lin and Sahai [JLS20] which achieves this much sought-after goal. However, this intricate construction relies on several assumptions including pairings making it quantum insecure, and has many complex reductions that incur significant loss in efficiency. Hence, it remains meaningful to consider simpler, plausibly post-quantum constructions, even in weaker security models.

Limitations of Prior Work. Despite their success, existing constructions of FE in the bounded collusion model suffer from at least three major drawbacks: i) the collusion bound Q must be declared at setup time and is fixed once and for all, ii) in the public key setting, these constructions support only the circuit model of computation and iii) all constructions that we are aware of are in the *key policy* setting (KPFE), where the function f is embedded in the secret key and the input $\mathbf{x}$ is embedded in the ciphertext of the FE scheme. The dual, ciphertext policy setting (CPFE), where the roles of f and $\mathbf{x}$ are swapped, is more natural in several applications but has received much less attention. We discuss each of these limitations in turn.

The first limitation pervades all prior work to the best of our knowledge and is quite a significant drawback in our opinion. Since the collusion bound must be declared at setup time, all data encrypted under the scheme must necessarily be subject to the same level of collusion-resistance. Thus, the practioner is forced to choose a collusion bound which is strong enough for the *most sensitive* information that may ever be encrypted under the scheme – this implies that Q is an upper bound on collusion resistance. However, not all information has the same level of sensitivity. Consider an organization: messages involving a potential merger with another organization, or a case of harassment which must be investigated are significantly more delicate than routine exchanges. It is desirable that such sensitive messages are protected even if a large number of key holders collude. On the other hand, for routine ciphertexts decrypted via function keys (such as checking whether some encrypted message is spam or not), such a strong level of collusion resistance is unnecessary. Moreover, the collusion bound Q impacts the size of the public key and ciphertext of the scheme. Thus, if Q is large, then the public key as well as every ciphertext generated by the scheme is forced to grow at least linearly in Q leading to a prohibitive impact on efficiency.

Regarding the second limitation, it is well known that being non-uniform, the circuit model is ill-suited for several applications [GTKP+13a, AS16, AS17, AM18]. In particular, circuits force the size of the input to be fixed a-priori which in turn necessitates instantiation of the scheme with an upper

bound on data size. This again leads to loss in efficiency and is ill-suited to datasets of dynamic size. Moreover, circuits incur *worst case* running time over all inputs, which is also clearly undesirable in practice. Finally, all constructions of CPFE for circuits that we are aware of, proceed via the universal circuit route, i.e. consider the universal circuit $U(f, \mathbf{x}) = f(\mathbf{x})$ and construct KPFE with the circuit $U(\cdot, \mathbf{x})$ in the secret key and f represented as a string, in the ciphertext. Aside from the loss of efficiency that results by this transformation (now, both the input and the function grow with the maximum circuit size), this transformation restricts the CPFE scheme to *bounded* size circuits. This is dissatisfying, as much in practice as in theory. The ciphertext policy variant of FE is desirable in many applications. Even in the special case of attribute based encryption (ABE), the ciphertext policy model allows an access control policy f to be embedded against a secret message m in the ciphertext. The secret key contains user attributes $\mathbf{x}$ which represent the various roles of a user, such as institute, department, date of joining and such others. Decryption succeeds to recover m if and only if the user's attributes satisfy the access control policy in the ciphertext. It is arguably more natural to have an access control policy apply to a secret message than to a user. Another application scenario is when the function f is proprietary and must be hidden via encryption, while the data can be made publicly available. For instance, suppose a government wants to enable citizens to run useful algorithms developed by different research labs on some public data, eg. census data. The government publishes a secret key for the data x, the labs publish ciphertexts with their respective algorithms f_i (of arbitrary size) and anyone can compute $f_i(x)$. The research labs want to keep their algorithms secret from competitors (but the government agency is trusted) so the function is encrypted. The data is public but becomes available asynchronously and independently of the function(s), so the labs cannot compute the function outputs each time. For example, new census data may be published every year but the same programs can be used every year. Computing private programs on public data is analogous to the setting of obfuscation, except that here the government agency can be trusted, making it simpler than obfuscation. Given the many natural applications of CPFE, it is undesirable to settle for a limited generic transformation via KPFE.

State of the Art. While FE for uniform models of computation has been studied [GTKP+13a, AS16, AS17, AM18], direct constructions from standard assumptions, supporting unbounded length inputs, are few and far-between. In the public key setting, the FE scheme by Agrawal and Singh [AS17] supports Turing machines and is based on the Learning With Errors assumption. However, this scheme only supports a single key request by the adversary in the security game. In the symmetric key setting, the recent work of Agrawal, Maitra and Yamada [AMY19] provided a construction of FE for non-deterministic finite automata (NFA) which is secure against bounded collusions of arbitrary size. However, generalizing this construction to the public key setting and to stronger models like Turing machines was left open. To the best of our knowledge, all bounded collusion FE schemes suffer from the fixed collusion bound, and the only CPFE scheme that supports unbounded sized circuits is that by Sahai and Seyalioglu [SS10], which is only single key secure.

1.1 Our Results

In this work, we improve the state of affairs in several ways:

1. *New Security Notion.* We introduce the notion of *dynamic* bounded collusion FE, where the declaration of collusion bound is delayed to the time of encryption. This enables the encryptor to dynamically choose the collusion bound for different ciphertexts depending on their individual level of sensitivity. Hence, the ciphertext size grows linearly with its own collusion bound and the public key size is independent of collusion bound. In contrast, all prior constructions have public key and ciphertext size that grow at least linearly with a fixed bound Q. All our constructions satisfy our new security notion – we also refer to our new notion as achieving *delayed* collusion resistance.

2. *CPFE for circuits with Dynamic Bounded Collusion.* We provide the first CPFE schemes for circuits enjoying dynamic bounded collusion security. In more detail:

   - By relying on the assumption of identity based encryption (IBE), we construct CPFE for circuits of *unbounded* size, output length and depth satisfying *non adaptive* simulation based security. Recall that non-adaptive simulation security refers to a game where the attacker must make all it's key requests before obtaining the challenge ciphertext [BSW11].
   - By strengthening the underlying assumption to IBE with receiver selective opening security, we obtain CPFE for circuits of *bounded* size, output length and depth enjoying *adaptive* simulation based security[1]. Moreover, we show that IBE is a necessary assumption for these primitives.
   - By relying on the Learning With Errors (LWE) assumption, we obtain a *succinct* CPFE for circuits – namely, supporting circuits with unbounded size, but fixed output length and depth, which achieves *adaptive* simulation based security.

3. *KPFE for circuits with dynamic bounded collusion.* We provide the first KPFE for circuits of unbounded size, but bounded depth and output length satisfying dynamic bounded collusion. Our construction relies on LWE and achieves adaptive simulation based security. This improves the security of succinct KPFE by Goldwasser et al. [GTKP+13b].

4. *KP and CP FE for TM/NL with dynamic bounded collusion.* We provide the first KPFE and CPFE constructions of bounded collusion functional encryption for Turing machines in the public key setting from LWE. Such a result was not known even with fixed collusion resistance to the best of our knowledge. Our constructions achieve non-adaptive simulation based security. In terms of functionality: a secret key in our KPFE construction encodes an TM M, a ciphertext encodes a message $\mathbf{x}$ and decryption allows recovery of $M(\mathbf{x})$ and nothing else. Both the input $\mathbf{x}$ and the machine M in our construction can be of *unbounded* polynomial length but the ciphertext size grows with the upper bound on the running time of the Turing machine on the given input $\mathbf{x}$. Given RAM access to the ciphertext, the scheme enjoys input specific decryption time.

5. *Adaptive Bounded Collusion FE for NL with dynamic bounded collusion.* The above construction guarantees non-adaptive security while supporting general Turing machines. We also consider a variant of the above scheme that satisfies stronger adaptive security, but at the cost of supporting smaller class of computation Nondeterministic Logarithmic-space (NL). Since NL contains Nondeterministic Finite Automata (NFA), this result subsumes *all* prior work of bounded collusion FE for uniform models from standard assumptions [AMY19, AS17].

1.2 Our Techniques

In this section, we provide an overview of our techniques. At a high level, our work addresses two broad challenges: obtaining stronger security guarantees via dynamic collusion and achieving more powerful functionality via general TM or NL. We examine each of these in turn.

Dynamic Bounded Collusion. Observe that the problem of constructing FE with delayed collusion bounds has not been studied until our work, even for bounded size circuits. A first observation is that in such an FE scheme, it is *necessary* that the efficiency of the setup and key generation algorithms are independent of (or dependent only poly-logarithmically on) the collusion bound Q. To the best of our knowledge, previous bounded FE schemes such as [GVW12, AR17, Agr17] do not satisfy this property.

[1] For the knowledgeable reader, the lower bound from [BSW11] does not apply because there is only one challenge ciphertext, with bounded output length in the security game.

However, the recent construction by Ananth and Vaikuntanathan [AV19] (AV19) does satisfy one half of this requirement – it enjoys a key generation algorithm which is efficient in this sense. Unfortunately, the setup algorithm of their construction runs in time that grows with Q. Our first step will be to remove the dependency on Q from the setup algorithm.

Improving AV19 to remove setup dependence on Q. To begin, we recap some relevant ideas from their construction. Their construction is generic: they construct Q-bounded FE scheme from a single key FE scheme. For concreteness, we instantiate the single key FE scheme with the concrete one by Sahai and Seyalioglu [SS10]. While their construction is key policy, we adapt it to the ciphertext policy setting in what follows. At a very high level, their Q bounded FE scheme runs Q subsystems in parallel, such that each subsystem in turn runs N instances of the SS10 scheme. Since their construction is optimized using an elegant combinatorial argument, they obtain a secret key size of poly rather than $O(Q \cdot \mathrm{poly})$. This optimized construction forms the starting point of our work. The specific details of their final construction are not relevant to this overview: we note only some salient features. Their construction makes use of an "MPC style" secret sharing scheme, where an input circuit C is divided into shares $\widehat{C}_1, \ldots, \widehat{C}_N$. Now, n out of N parties, without any interaction, perform some computation on their shares corresponding to input x, to obtain partial outputs $\widehat{y}_1, \ldots, \widehat{y}_n$. These n partial outputs are then combined to obtain output $y = C(x)$.

Using the above secret sharing scheme, the AV19 construction may be summarized as follows. The setup algorithm generates several (the exact number is not relevant for us) public and secret key pairs of a PKE scheme. To encrypt a circuit C, the encryptor computes many shares of C as described above. These shares $\widehat{C}_i$ are then hardwired into garbled circuits which, given input x, compute $\widehat{y}_i$ using the above method. The labels of these garbled circuits are encrypted using the public keys provided by the setup algorithm. The function key for x is a set of PKE secret keys that depend on x (again, the precise dependence is not required here). The decryptor first uses the PKE secret keys to recover the appropriate labels of the garbled circuit corresponding to x. Then the garbled circuit is evaluated to obtain shares $\widehat{y}_i$ of the decryption result. These shares are then combined to obtain $C(x)$.

The reason why their setup algorithm takes time linear in the collusion bound Q is that the number of public keys required by their scheme is linear in Q. Having unrolled their construction when instantiated with [SS10], our approach to removing this dependency is simple: we replace these public keys of PKE with a single master public key of an identity based encryption (IBE) scheme. Then, encryption with PK_i is replaced by an encryption with $\mathsf{IBE.Enc}(\mathsf{IBE.mpk}, i, \cdot)$, where i is the identity. The intuition for security is the same as the original construction. Thus, we use the power of the IBE to hide the labels of the garbled circuits, use the power of the garbled circuits to hide information other than the decryption shares, and finally use the power of the secret sharing scheme to hide the circuit C.

We show that if we desire adaptive simulation (AD-SIM) security for the resultant construction, we need an IBE satisfying the stronger security notion of *receiver selective opening security* [KT18]. This is for a reason similar to why [GVW12, AV19] required non-committing security for the underlying public key encryption. Since the length of the message that can be encrypted using an IBE with receiver selective opening security is bounded, so is the size of the circuits that can be encrypted in our CPFE scheme. If we relax the security notion and consider non-adaptive (NA-SIM) security, we can use IBE with standard IND-CPA security. This allows us to encrypt a message of unbounded length, which in turn allows us to encrypt circuits with unbounded size. A simple adaptation of the lower bound of Boneh et al. [BSW11] shows that to support unbounded sized circuits, NA-SIM security is optimal[2].

[2]Consider a circuit C^* with unbounded size and unbounded output length. Let us say that this circuit has hardwired with random string s of length ℓ, and upon an input x, the circuit ignores the input and outputs s. Here, ℓ is unbounded. Now if the attacker makes even a single key request for some x^* *after* seeing the challenge ciphertext corresponding to C^*, the simulator is faced with the impossible task of embedding a random string of length ℓ into a fixed sized secret key. Hence, the adversary must not be allowed post-challenge key requests when circuits of unbounded output length are supported.

Supporting Delayed Collusion in Security. So far, we have constructed bounded CPFE schemes whose setup and key generation algorithms run in time independent of Q. However, this is only necessary and not sufficient to construct FE with delayed collusion bound. Once the system is set up with the bound Q, this will still only be secure against a collusion of size Q. Our next step is to remove this restriction so that the encryptor can choose the bound flexibly, and in particular, differently for each ciphertext.

Here, our crucial observation is that the setup and key generation algorithms of the scheme can be run even for super polynomial Q, thanks to their efficiency properties. Hence, we may use the "powers of two trick" [GTKP+13a], where we run the system with different collusion bounds $Q = 2, 2^2, \ldots, 2^\lambda$. The setup and key generation algorithms are run for these λ subsystems in parallel. When we encrypt the message, the encryptor chooses the smallest 2^i that exceeds the bound Q it wants and encrypts the message using the i-th subsystem. Decryption is performed using the secret key for the i-th instance of the subsystem. The resulting scheme inherits the efficiency and security properties of the subsystem. Namely, if the subscheme is NA-SIM (respectively AD-SIM) secure and supports unbounded (respectively bounded) circuits, so does the resulting scheme. Thus, we obtain two schemes with incomparable properties. Please see Section 3 for details.

Observe that the construction of bounded collusion FE in AV19 can be based on the minimal assumption of plain PKE [AV19]. On the other hand, our constructions described above rely on the stronger primitive of IBE. It is natural to ask whether we can base the security of the construction to weaker primitives such as PKE. We answer this question negatively. In Section 7 we argue that the usage of IBE is unavoidable, by showing that FE with dynamic bounded collusion for very small class of functionalities already implies IBE.

Supporting More General Function Classes. Next, we describe our techniques for supporting more flexible models of computation, namely Turing machines or NL. The main difficulty of constructing FE for these function classes is to handle unbounded length inputs and unbounded size machines simultaneously. To address this, we borrow a trick from the work of Agrawal, Maitra and Yamada [AMY19]: instead of trying to handling them at once, we construct intermediate schemes that can handle an unbounded size object on one side, but bounded size object on the other. Towards this, we construct KPFE and CPFE schemes with dynamic bounded collusion that support unbounded size circuits, but with bounded output length and depth (note that input length is always fixed). Later, we will see how to compile these to construct FE for more general function classes. Note that in the previous step we already constructed a CPFE scheme that can handle circuits with unbounded size even without restrictions on depth and output length. However, this construction was only NA-SIM secure. Here, we aim to construct schemes with AD-SIM security.

Succinct KPFE and CPFE. Let us start with the construction of KPFE that can support unbounded size circuits. Note that such FE schemes have already been constructed by the previous works under the name of succinct FE [GTKP+13b, Agr17]. However, they do not satisfy the security requirement that we want. Namely, they are (conventional) bounded collusion schemes and do not satisfy the delayed collusion property. In addition, they only satisfy NA-SIM security. Here, we upgrade the security of existing succinct FE schemes so that they satisfy the delayed collusion property and AD-SIM security with the help of our CPFE scheme for *bounded* circuits that already satisfies the desired security properties. In more detail, we combine succinct single-key KPFE, denoted by 1KPFE with our AD-SIM secure CPFE for bounded circuits, to obtain a new succinct KPFE with the desired security properties.

At a high level, the construction works as follows. The master public key and master secret key of the final KPFE scheme are those of the CPFE. To encrypt a message x for a collusion bound 1^Q, the encryptor first constructs a circuit $\mathsf{1KPFE.Enc}(\cdot, x)$[3], which is an encryption algorithm of the single-key KPFE that takes as input a master public key of the single-key KPFE and outputs an encryption of the

[3]The description here is oversimplified – in fact we need a PRF to derive the randomness for the encryption.

message x under the key. The encryptor then encrypts the circuit using the CPFE scheme with respect to the bound 1^Q. To generate a secret key for a circuit C, we first freshly generate a master key pair of the single-key KPFE $(1\mathsf{KPFE.mpk}, 1\mathsf{KPFE.msk})$. We then generate a CPFE secret key $\mathsf{CPFE.sk}$ corresponding to the string $1\mathsf{KPFE.mpk}$ and then generate secret key $1\mathsf{KPFE.sk}_C$ for the circuit C of the single-key KPFE scheme. The final secret key is $(\mathsf{CPFE.sk}, 1\mathsf{KPFE.sk}_C)$. Decryption is done by first decrypting the CPFE ciphertext using the CPFE secret key to recover $1\mathsf{KPFE.Enc}(1\mathsf{KPFE.mpk}, x)$ and then decrypting it using the secret key $1\mathsf{KPFE.sk}_C$ of the single-key KPFE scheme to recover $C(x)$.

We discuss the efficiency of the above scheme. First we claim that the above scheme is succinct (or equivalently, can deal with unbounded size of circuits). This is the case even though underlying CPFE can only deal with bounded size circuits, since the size of circuits that should be supported by the encryption algorithm of CPFE is $|1\mathsf{KPFE.Enc}(\cdot, x)| = \mathrm{poly}(\lambda, |x|)$, which is independent of the size of circuits by the succinctness of the underlying $1\mathsf{KPFE}$ scheme. Next, we discuss the security of the scheme. Intuitively speaking, by the security of the underlying CPFE, the adversary can obtain no information beyond the decryption result of the CPFE. This means that, the adversary only obtains the information $\{1\mathsf{KPFE.Enc}(1\mathsf{KPFE.mpk}^{(i)}, x), 1\mathsf{KPFE.sk}^{(i)}_{C^{(i)}}\}_{i\in[Q]}$ for Q freshly generated, independent instances. In turn, this implies that the adversary can only obtain the information of $\{C^{(i)}(x)\}_{i\in[Q]}$ by the single-key security of the underlying KPFE, as desired. A formal argument shows that the resulting KPFE scheme inherits AD-SIM security of the CPFE, even if the underlying single-key KPFE is only NA-SIM secure. Please see Section 4 for details.

Next, we discuss the adaptation to the CPFE setting. In this construction, we use a reusable garbled circuit scheme [GTKP+13b] instead of single key succinct KPFE. Recall that a reusable garbled circuit is a symmetric key variant of single key succinct KPFE, where the circuit in the secret key is also hidden. Now, to encrypt a circuit C in our CPFE, we run the garbling algorithm of the reusable garbled circuit scheme on input C to obtain the garbled version of C as well as some secret information for input garbling. We then construct an input encoding circuit that takes as input x and outputs an encoded version of it using the secret information generated above. Then, we encrypt this input encoding circuit using the underlying CPFE. The final ciphertext consists of the garbled circuit and the encrypted circuit. To generate a secret key for x, we use the key generation algorithm of the underlying CPFE scheme to obtain a secret key for x. Decryption requires running the decryption algorithm of the underlying CPFE scheme to obtain the encoded version of input x and then using this result with the garbled version of the circuit to recover $C(x)$. The resulting scheme is AD-SIM secure and supports unbounded size circuits. Please see Section 5 for details.

Handling Unbounded Inputs. So far, we have constructed KPFE and CPFE schemes that can handle unbounded size circuits but with fixed input size. However, for obtaining FE for Turing machines, we have to be able to encrypt unbounded length inputs and this is clearly insufficient. To enable this, we use the "delayed encryption" technique by Goyal, Koppula, and Waters (GKW16) [GKW16]. Intuitively, the technique uses the power of garbled circuits and IBE to transport us to a world where there are infinitely many instances of FE $\{\mathsf{mpk}_i\}_{i\in\mathbb{N}}$ – the encryptor can choose a master public key for some index j and encrypt the message. A secret key is associated with some index k and the decryption is possible iff $j = k$.

In more detail, the GKW16 scheme works as follows. During setup, the master public key and master secret key of an IBE scheme are generated. To encrypt a message x using the j-th instance of the FE scheme as described above without knowing the corresponding mpk_j, the encryptor first constructs a circuit $\mathsf{Enc}(\cdot, x)$ that takes as input the master public key mpk_j for the j-th instance and outputs the ciphertext $\mathsf{Enc}(\mathsf{mpk}_j, x)$. The encryptor then garbles this encryption circuit to obtain the corresponding garbled circuit and set of labels. Then, the encryptor encrypts each pair of labels with the IBE, where the identity for which a label is encrypted encodes the index j, the position of the label and a single bit. Note

that the above step can be done without knowing mpk_j. Correspondingly, the key generation algorithm computes IBE secret keys for the correct bits of mpk_k, which together with the IBE ciphertexts allow the decryptor to recover the labels corresponding to mpk_j in the j^{th} garbled circuit when $j = k$. This lets the decryptor evaluate the garbled circuit to retrieve the ciphertext $\mathsf{Enc}(\mathsf{mpk}_j, x)$ as desired.

With this technique, we make progress towards our goal, because we can encrypt a message of any length by the scheme, rather than only being able to encrypt a message of fixed length. However, this is still not enough, since the decryption is possible only when the index j and k match. For example, let us imagine that we want to construct FE for Turing machine using infinitely many instances of FE for circuits, where the i-th instance of FE supports circuits with input length i. Let t be an upper bound on the runtime of the TM on input x. If we encrypt a message $(x, 1^t)$ as an input to a Turing machine, we may encrypt it using $|(x, 1^t)|$-th instance of the FE. On the other hand, to generate a secret key for a Turing machine M, we convert the machine into a circuit $C_M(\cdot)$ and generate a secret key for it. However, it is unclear how to define the input length of the circuit. If the input length does not match $|(x, 1^t)|$, the decryption is impossible. Meanwhile, the entity who generates the secret key does not know the input length $|(x, 1^t)|$, so is stuck.

To resolve the above problem, we incorporate a trick by Agrawal, Maitra and Yamada [AMY19] used to support unbounded inputs for an NFA machine in the context of ABE. They construct two restricted ABE schemes for NFA: one that supports decryption in the case where the length $|x|$ of the input x is larger than the size $|M|$ of the machine M and one that supports the case where $|x| \leq |M|$. Then, they run the restricted schemes in parallel. In the decryption algorithm, these sub-schemes complement each other. Namely, we use the first sub-scheme to decrypt a ciphertext if $|x| > |M|$ and the second otherwise. Though they introduce the trick in the context of ABE, this perfectly works in the context of FE as well. A hurdle is that their technique works only in the secret key setting, since the encryptor is required to know a master secret in order to generate unbounded instances of FE (proportional to its input length) on the fly. However, we show that in conjunction with the technique from [GKW16] described above, this idea can be made to work in the public key setting as well. In a nutshell, this technique lets us encode x and M in multiple slots of the FE instances so that they always intersect, instead of encoding them on a single slot like in [GKW16].

Onward to FE for TM. Armed with these techniques, let us try constructing FE for TM. As discussed above, our construction handles the cases $|(x, 1^t)| \leq |M|$ and $|(x, 1^t)| > |M|$ separately. Let us begin with the former using our KPFE that supports unbounded size circuits.

By the technique of [GKW16], we can assume that we are in a world where there are infinitely many KPFE instances available and the i-th instance supports circuits with input length i. To encrypt a message x with respect to the time bound 1^t, we use the $|(x, 1^t)|$-th instance of the KPFE. To generate the secret key for a Turing machine M on the other hand, we encode M into a set of circuits $C_{i,M}$ for $i = 1, \ldots, |M|$, where $C_{i,M}$ is a circuit that takes as input a string $(x, 1^t)$ and then run the machine M for t steps and outputs the result. We then generate secret keys for $C_{i,M}$ using the i-th instance of KPFE for all of $i \in [|M|]$. This is possible even for unbounded M, because each KPFE instance supports unbounded size circuits. The decryption is possible when $|(x, 1^t)| \leq |M|$ by using the $|(x, 1^t)|$-th instance. However, it is evident that this is an incomplete scheme, since the decryption is not possible when $|(x, 1^t)| > |M|$.

To complement this, we next construct a scheme that deals with the case of $|(x, 1^t)| > |M|$ using our unbounded CPFE scheme. To encrypt a message $(x, 1^t)$ we convert it into a circuit $\{U_{i,x,t}\}_{i \in [|(x,1^t)|]}$, where $U_{i,x,t}$ is a circuit that takes as input a string M of length i, interprets it as a description of a Turing machine, and then runs it on input x for t steps to obtain the result. We then encrypt the circuit $U_{i,x,t}$ using the i-th instance of the FE for all of $i \in [|(x, 1^t)|]$. This requires the underlying CPFE scheme to support unbounded size of circuits. Since our NA-SIM secure CPFE construction supports such circuits, we can use this here. On the other hand, our CPFE scheme with AD-SIM security cannot be used here, because the scheme can only support circuits with bounded depth, which prohibits us from running

the Turing machine inside the circuit for t steps, where t may be arbitrarily large. To generate a secret key for a Turing machine M, we use the $|x|$-th instance of the FE. It can be seen that the decryption is possible in this scheme when $|(x, 1^t)| > |M|$. Having the construction for the cases of $|(x, 1^t)| > |M|$ and $|(x, 1^t)| \leq |M|$, we can obtain the final construction by running them in parallel. The final scheme is NA-SIM secure, because the underlying CPFE scheme is NA-SIM secure (even though the KPFE scheme satisfies the stronger AD-SIM security). Please see Section 6 for details.

Above, the ciphertext size grows with the t, the upper bound on the runtime of the TM on a given input x. We emphasize that t is not a global bound but can vary with each input x, and is therefore unbounded. While we do not know how to remove this dependence using our current techniques, we remark that decryption time can still be input specific using the "powers of two" trick from Goldwasser et al. [GTKP+13a]. This requires the decryption algorithm to have RAM access to the ciphertext. In more detail, the encryptor may repeat the encryption procedure for $\lceil \log_2 t \rceil$ possible values of TM runtime, with values 2^i for $i \in [\lceil \log_2 t \rceil]$. The decryptor can start with the ciphertext corresponding to the smallest value and proceed to the next ciphertext only if the previous decryption did not result in a valid output[4]. This ensures that the scheme enjoys input specific decryption time.

FE for NL with Adaptive Security. The above construction guarantees NA-SIM security while supporting general Turing machines. We also consider a variant of the above scheme that satisfies stronger AD-SIM security, at the cost of supporting smaller class of computation NL. This is achieved by using AD-SIM secure FE for both the building blocks of KPFE and CPFE. Recall that the reason why we cannot use AD-SIM secure CPFE in the above construction was that it was not possible to run the Turing machine for an unbounded number of steps inside a circuit of fixed depth. This issue arises because the Turing machine is run sequentially. In a nutshell, our next idea is to parallelize computation so that the depth for the corresponding circuit can be bounded by some fixed polynomial. Such a parallelization of the computation is not known to be possible for general Turing machines, but we can do it for NL, which is more restrictive. To do so, we represent the computation of NL as a multiplication of matrices as was done in [LL20]. First, we enumerate all the possible internal configurations of the Turing machine M on input x that may appear during the computation. The number of such internal configurations can be bounded by some polynomial, since the length of the working tape is logarithmic. We then construct the transition matrix $\mathbf{M}$ for the configurations. Then, one can determine whether M accepts the input x within time t by computing $\mathbf{M}^t$ due to the properties of the transition matrix. Since the matrix exponentiation can be done by $O(\log t)$ multiplications of the matrix, this can be performed by fixed polynomial depth even for unbounded t (assuming $t < 2^\lambda$). Please see Section 6.3 for details.

1.3 Concurrent Work

We note that a concurrent work [GGLW21] has independently introduced the notion of dynamic collusion bound for KPFE schemes, same as what we consider in this paper. They obtain simulation secure KPFE schemes for circuits with dynamic collusion resistance. Further, their techniques also significantly overlap with our CPFE constructions in Section 3. In particular, they compile existing bounded collusion KPFE schemes [GVW12, AV19] with IND-CPA secure IBE to obtain KPFE for circuits satisfying dynamic collusion resistance. However, we also extend our results further to obtain succinct CP/KP-FE schemes for circuits with dynamic collusion property, and also to support Turing machines and NL with various security tradeoffs as explained in Section 1.2. Finally, we also argue about the necessity of IBE to achieve dynamic collusion bound property for FE schemes, which [GGLW21] has left as an open problem.

[4]The definition of TM can be easily modified to output "unfinished" if the computation did not conclude in a given number of steps.

2 Preliminaries

In this section, we define some notation and preliminaries that we require. Some additional preliminaries are provided in Appendix A.

Notation. We begin by defining the notation that we will use throughout this work. We use bold letters to denote vectors and the notation $[a, b]$ to denote the set of integers $\{k \in \mathbb{N} \mid a \leq k \leq b\}$. We use $[n]$ to denote the set $[1, n]$. When we consider a string of form $(x, 1^t)$, we assume that x and 1^t can be derived from it and we differentiate $(x, 1^t)$ from $(x1, 1^{t-1})$ for instance. To do so, we consider the alphabet ",," in addition to 0 and 1 and represent each alphabet by 2-bit for example. We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we use $\mathrm{negl}(n)$ to denote a negligible function of n. We say $f(n)$ is *polynomial* if it is $O(n^c)$ for some constant $c > 0$, and we use $\mathrm{poly}(n)$ to denote a polynomial function of n. We use the abbreviation PPT for probabilistic polynomial-time. The function $\log x$ is the base 2 logarithm of x.

2.1 Turing Machines

Here, we recall the definition of a Turing machine (TM) following [LL20]. The definition considers Turing machines with two tapes, namely, input tape and working tape.

Definition 2.1 (Turing Machine)**.** A (deterministic) TM M is represented by the tuple $M = (Q, \delta, F)$ where Q is the number of states (we use $[Q]$ as the set of states and 1 as the initial state), $F \subset [Q]$ is the set of accepting state and

$$\begin{aligned} \delta : [Q] \times \{0,1\} \times \{0,1\} &\to [Q] \times \{0,1\} \times \{0, \pm 1\} \times \{0, \pm 1\} \\ (q, b_1, b_2) &\mapsto (q', b_2', \Delta i, \Delta j) \end{aligned}$$

is the state transition function, which, given the current state q, the symbol b_1 on the input tape under scan, and the symbol b_2 on the work tape under scan, specifies the new state q', the symbol b_2', overwriting b_2, the direction Δi to which the input tape pointers moves, and the direction Δj to which the work tape pointer moves. The machine is required to hang (instead of halting) once it reaches an accepting state, i.e., for all $q \in [Q]$ such that $q \in F$ and $b_1, b_2 \in \{0, 1\}$, it holds that $\delta(q, b_1, b_2) = (q, b_2, 0, 0)$.

For input length $n \geq 1$ and space complexity bound $s \geq 1$, the set of internal configurations of M is

$$\mathcal{Q}_{M,n,s} = [n] \times [s] \times \{0,1\}^s \times [Q]$$

where $(i, j, W, q) \in \mathcal{Q}_{M,n,s}$ specifies the input tape pointer $i \in [n]$, the work tape pointer $j \in [s]$, the content of the work tape $W \in \{0,1\}^s$ and the machine state $q \in [Q]$.

For any bit-string $x \in \{0,1\}^n$ for $n \geq 1$ and time/space complexity bounds t, $s \geq 1$, the machine M accepts x within time t and space s if there exists a sequence of internal configurations (computation path of t steps) $c_0, \ldots, c_t \in \mathcal{Q}_{M,n,s}$ with $c_k = (i_k, j_k, W_k, q_k)$ such that $(i_0, j_0, W_0, q_0) = (1, 1, 0^s, 1)$ (initial configuration),

$$\text{for all } 0 \leq k < t: \quad \begin{cases} \delta(q_k, x[i_k], W_k[j_k]) = (q_{k+1}, W_{k+1}[j_k], i_{k+1} - i_k, j_{k+1} - j_k) \\ W_{k+1}[j] = W_k[j] \quad \text{for all } j \neq j_k \qquad \text{(valid transitions);} \end{cases}$$

where $x[i]$ is the i the bit of the string x and $W_k[j]$ is the j the bit of the string W_k, and $q_t \in F$ (accepting). We also say M accepts x within time t (without the space bound) if M accepts x within time t and space $s = t$.

Next, we define time/space bounded computation with *non-deterministic* Turing machines. The definition is the same as Theorem 2.1, except with the following changes:

- The transition criterion δ can be any relation between (i.e., any subset of the Cartesian product of) $[Q] \times \{0,1\}^2$ and $[Q] \times \{0,1\} \times \{0, \pm q\}^2$, where $((q, b_1, b_2), (b_2', b_2', \Delta i, \Delta j)) \in \delta$ means that if the current state is q, the input tape symbol under scan is b_1 and the work tape symbol under scan is b_2, then it is valid to transit into state q', overwrite b_2 with b_2', and move the input and work tape pointer by offsets Δi and Δj respectively.
- The definition of hanging in accepting states is that for all $q \in [Q]$ such that $q \in F$ and all $b_1, b_2 \in \{0,1\}$,

$$\delta \cap \left(\{(q, b_1, b_2)\} \times ([Q] \times \{0,1\} \times \{0, \pm 1\}^2)\right) = \{(q, b_1, b_2), (q, b_2, 0, 0)\}.$$

- In the definition of acceptance

$$\delta(q_k, x[i_k], W_k[j_k]) = (q_{k+1}, W_{k+1}[j_k], i_{k+1} - i_k, j_{k+1} - j_k)$$

$$\text{is changed to } ((q_k, x[i_k], W_k[j_k]), (q_{k+1}, W_{k+1}[j_k], i_{k+1} - i_k, j_{k+1} - j_k)) \in \delta.$$

The following lemma can be obtained by a simple argument on emulating a Turing machine on Boolean circuits. While we have many other clever methods of simulating Turing machines on circuits (e.g., [PF79]), we use the following simple one because it evaluates the depth of the circuit as a function on the size of Turing machine, which is usually regarded as a constant and ignored.

Lemma 2.2 (Emulating a Turing Machine on Circuit)**.** *Consider a circuit that takes as input a description of a (deterministic) Turing machine $M = (Q, \delta, F)$, input x to M, a configuration $(i, j, W, q) \in \mathcal{Q}_{M,|x|,|W|}$ and outputs the next configuration (i', j', W', q'). We can implement such a circuit with depth* $\mathrm{poly}(\log|x|, \log|W|, \log|M|)$ *and size* $\mathrm{poly}(|x|, |W|, |M|)$.

Proof. The circuit is implemented as follows. We focus on the depth of the circuits, since the bound on the size will be clear from the description. Given the input, it first retrieves the i-th bit $x[i] \in \{0,1\}$ of the input tape. This can be done by a circuit with depth $O(\log|x|)$, which checks whether $i = \nu$ or not for each position ν of the string x in parallel and returns $x[\nu]$ for ν such that $\nu = i$. Similarly, it can retrieve $W[j]$ with depth $O(\log|W|)$. Given $x[i]$ and $W[j]$, it then retrieves $\delta(q, x[i], W[j])$ from δ, which can be done with depth $O(\log|Q|)$ similarly to the above. Given $\delta(q, x[i], W[j]) = (q', b', \Delta i, \Delta j)$, the update of i and j can be done in depth $O(\log|x|)$ and $O(\log|W|)$, respectively. Writing back the new value b' can also be done in depth $O(\log|W|)$ by finding the right place to write in the tape and change the value there. From the above discussion, the total depth of the circuit is $\mathrm{poly}(\log|x|, \log|W|, \log|M|)$ as desired. □

We also need the following lemma, which can be obtained by a simple observation.

Lemma 2.3 (Checking Transition for Non-deterministic Turing Machine)**.** *Consider a circuit that takes as input a description of a non-deterministic Turing machine $M = (Q, \delta, F)$, input x to M, two configurations $(i, j, W, q) \in \mathcal{Q}_{M,|x|,|W|}$ and $(i', j', W', q') \in \mathcal{Q}_{M,|x|,|W|}$ and outputs whether $((i, j, W, q), (i', j', W', q')) \in \delta$ or not. We can implement such a circuit with depth* $\mathrm{poly}(\log|x|, \log|W|, \log|M|)$ *and size* $\mathrm{poly}(|x|, |W|, |M|)$.

Proof. The circuit is implemented as follows. We focus on the depth of the circuits, since the bound on the size will be clear from the description. Given the input, it first checks whether $i' - i \in \{0, \pm 1\}$, $j' - j \in \{0, \pm 1\}$. Clearly, this can be done in depth $\mathrm{poly}(\log|x|, \log|W|)$. It then checks whether $W'[k] = W[k]$ for all $k \in [|W|] \backslash \{k\}$, which can be done in depth $\mathrm{poly}(\log|W|)$. It then checks whether $((q, x[i], W[j]), (q', W'[j], i' - i, j' - j)) \in \delta$ or not. This can be checked in depth $\mathrm{poly}(\log|Q|)$. Therefore, the total depth of the circuit is $\mathrm{poly}(\log|x|, \log|W|, \log|M|)$ as desired. □

2.2 Functional Encryption

Functional encryption (FE) [SW05, BSW11, O'N10] has been traditionally defined in a setting where a trusted key generator holding a master secret key provides authorized users with secret keys corresponding to functions. Such a key, when used to decrypt ciphertexts, reveals only the function of the plaintexts and nothing else. In this subsection, we define the notion of functional encryption (FE) more generally so that it captures the above notion as well as other types of functionalities as special cases.

2.2.1 Syntax and Correctness.

Let $R : \mathcal{X} \times \mathcal{Y} \to \{0,1\}^*$ be a two-input function where $\mathcal{X}$ and $\mathcal{Y}$ denote "message space" and "key attribute space", respectively. Ideally, we would like to have an FE scheme that handles the relation R directly, where we can encrypt any message $x \in \mathcal{X}$ and can generate a secret key for any key attribute $y \in \mathcal{Y}$. However, in many cases, we are only able to construct a scheme that poses restrictions on the message space and key attribute space. To capture such restrictions, we introduce a parameter prm and consider subsets of the domains $\mathcal{X}_{\mathsf{prm}} \subseteq \mathcal{X}$ and $\mathcal{Y}_{\mathsf{prm}} \subseteq \mathcal{Y}$ specified by it and the function R_{prm} defined by restricting the function R on $\mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}}$. An FE (FE) scheme for $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$ is defined by the following PPT algorithms:

$\mathsf{Setup}(1^\lambda, \mathsf{prm}) \to (\mathsf{mpk}, \mathsf{msk})$: The setup algorithm takes as input the unary representation of the security parameter λ and a parameter prm that restricts the domain and range of the function and outputs the master public key mpk and a master secret key msk.

$\mathsf{Encrypt}(\mathsf{mpk}, x) \to \mathsf{ct}$: The encryption algorithm takes as input a master public key mpk and a message $x \in \mathcal{X}_{\mathsf{prm}}$. It outputs a ciphertext ct.

$\mathsf{KeyGen}(\mathsf{msk}, y) \to \mathsf{sk}$: The key generation algorithm takes as input the master secret key msk, and a key attribute $y \in \mathcal{Y}_{\mathsf{prm}}$. It outputs a secret key sk. We assume that y is included in sk.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}) \to m$ or $\perp$: The decryption algorithm takes as input a ciphertext ct and a secret key sk. It outputs the message m or $\perp$ which represents that the ciphertext is not in a valid form.

Remark 2.4 (Bounded collusion variants). In this paper, we mainly focus on FE with bounded collusion security, where the security is guaranteed only when the number of secret keys that the adversary obtains during the security game is below collusion bound Q. To do so, we have to slightly change the syntax above. We consider two different types of syntax for FE depending on when Q is declared.

- The first notion we consider is *bounded collusion FE* [GVW12, AV19], where Q is fixed when the system is setup. In more details, we change the syntax of FE defined above so that all the algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ take 1^Q as additional input.
- The second notion we consider is a new notion that we call *dynamic bounded collusion FE*. In this notion, the bound Q is specified by the encryptor, not by the setup. Namely, we change the syntax of FE above so that only the encryption algorithms Enc takes 1^Q as input, whereas other algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Dec})$ do not.

We note that the requirement that the algorithms run in polynomial time along with the fact that all algorithms in bounded collusion FE take 1^Q as additional input imply that all the algorithms run in polynomial in Q. In the case of FE with dynamic bounded collusion, similar implication holds for the encryption algorithm. However, the running time of Setup and KeyGen should be dependent only on λ and $|\mathsf{prm}|$, whereas the running time of Dec may indirectly depend on Q if the size of the input ciphertext is dependent on Q.

Definition 2.5 (Correctness). An FE scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is correct if for all prm, $x \in \mathcal{X}_{\mathsf{prm}}$, and $y \in \mathcal{Y}_{\mathsf{prm}}$,

$$\Pr\left[\begin{array}{c}(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm}) : \\ \mathsf{Dec}\big(\mathsf{Enc}(\mathsf{mpk}, x), \mathsf{KeyGen}(\mathsf{msk}, y)\big) \neq R(x, y)\end{array}\right] = \mathrm{negl}(\lambda)$$

where probability is taken over the random coins of Setup, KeyGen and Enc.

2.2.2 Security Notions.

As security notions for FE, we define simulation-based notions. We are mainly interested in the bounded collusion settings because the security notion without the collusion bound is shown to be impossible [AGVW13]. We first provide the description of the security game for FE with dynamic bounded collusion and then for bounded collusion FE.

Definition 2.6 ($\mathsf{AD\text{-}SIM}$ and $\mathsf{NA\text{-}SIM}$ Security for FE with Dynamic Bounded Collusion). Let $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a (public key) FE scheme with dynamic bounded collusion for the function family $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$. For every stateful PPT adversary A and a stateful PPT simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$ consider the following experiments:

$\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{A}}(1^\lambda)$:	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}(1^\lambda)$:
1: $\mathsf{prm} \leftarrow \mathsf{A}(1^\lambda)$	1: $\mathsf{prm} \leftarrow \mathsf{A}(1^\lambda)$
2: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm})$	2: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm})$
3: $(x, 1^Q) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$	3: $(x, 1^Q) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$ • Let $(y^{(1)}, \ldots, y^{(Q_1)})$ be A's oracle queries. • Let $\mathsf{sk}^{(q)}$ be the oracle reply to $y^{(q)}$. • Let $\mathcal{V} := \left\{\left(z^{(q)} := R(x, y^{(q)}), y^{(q)}, \mathsf{sk}^{(q)}\right)\right\}_{q \in [Q_1]}$.
4: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, 1^Q)$	4: $(\mathsf{ct}, \mathsf{st}) \leftarrow \mathsf{SimEnc}(\mathsf{mpk}, \mathcal{V}, 1^{\lvert x\rvert}, 1^Q)$
5: $b \leftarrow \mathsf{A}^{\mathcal{O}(\mathsf{msk},\cdot)}(\mathsf{mpk}, \mathsf{ct})$	5: $b \leftarrow \mathsf{A}^{\mathcal{O}'(\mathsf{st},\mathsf{msk},\cdot)}(\mathsf{mpk}, \mathsf{ct})$
6: Output b	6: Output b

We emphasize that the adversary A is stateful, even though we do not explicitly include the internal state of it into the output above for the simplicity of the notation. On the other hand, the above explicitly denotes the internal state of the simulator Sim by st. We distinguish between two cases of the above experiment:

1. The adaptive case, where:

   - The oracle $\mathcal{O}(\mathsf{msk}, \cdot) = \mathsf{KeyGen}(\mathsf{msk}, \cdot)$ with $1 \leq Q_1 < Q$, and
   - The oracle $\mathcal{O}'(\mathsf{st}, \mathsf{msk}, \cdot)$ takes as input the q-th key query $y^{(q)}$ for $q \in [Q_1 + 1, Q_1 + Q_2]$ and returns $\mathsf{SimKG}(\mathsf{st}, \mathsf{msk}, R(x, y^{(q)}), y^{(q)})$, where $Q_1 + Q_2 \leq Q$

   The FE scheme FE is then said to be simulation secure for one message against adaptive adversaries ($\mathsf{AD\text{-}SIM}$-secure, for short) if there is a PPT simulator Sim such that for every PPT adversary A, the following holds:

$$\left|\Pr[\mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{A}}\left(1^\lambda\right) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}\left(1^\lambda\right) = 1]\right| = \mathrm{negl}(\lambda) \tag{2.1}$$

2. The non-adaptive case, where $Q_1 \leq Q$ and the oracles $\mathcal{O}(\mathsf{msk}, \cdot)$ and $\mathcal{O}'(\mathsf{msk}, \cdot)$ are both the "empty" oracles that return nothing: The FE scheme FE is then said to be simulation secure for one message against non-adaptive queries ($\mathsf{NA\text{-}SIM}$-secure, for short) if there is PPT simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \perp)$ such that for every PPT adversaries A, Eq. (2.1) holds. Note that in the non-adaptive case, we can ignore st since SimKG is not present in the above game and it is never used by other algorithm.

Definition 2.7 ($\mathsf{AD\text{-}SIM}$ **and** $\mathsf{NA\text{-}SIM}$ **Security for bounded collusion FE** [GVW12])**.** Let $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a (public key) bounded collusion FE scheme for the function family $\{R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*\}_{\mathsf{prm}}$. We define $\mathsf{AD\text{-}SIM}$ and $\mathsf{NA\text{-}SIM}$ security for FE by considering the same game as Definition 2.6 with the following changes:

- We change A to output 1^Q in addition to prm at the beginning of the game.
- All the algorithms run in the experiment $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{SimEnc}, \mathsf{SimKG})$ take 1^Q as additional input.

We also define weaker notion of the security where the adversary is restricted to always choose $Q = 1$ in the non-adaptive case. If the scheme only satisfies this weaker security notion, we say the scheme is 1-$\mathsf{NA\text{-}SIM}$ secure.

Remark 2.8. The above definition allows to argue security of a single instance of FE. For the security proof of our constructions in Section 6.1, it is convenient to consider multi-instance version of the above notion. In the multi-instance security variant, the adversary declares the number of instances M at the beginning of the game and interact with each instance as above. In the real world, the adversary interacts with real algorithms in all instances, while in the ideal world, it interacts with simulators in all instances. The adversary can make queries in arbitrary order and make them arbitrarily correlated as long as it respects the restriction for each instance. This multi-instance security notion is easily shown to be equivalent to the single-instance security notion above by simple hybrid argument.

2.2.3 Special Classes of FE

We then define various kinds of FE by specifying the relation.

KPFE for circuits. To define KPFE for circuits, we set $\mathcal{X} = \{0,1\}^*$ and $\mathcal{Y}$ as the set of all circuits and define $R(x, C) = C(x)$ if the length of the string x and the input length of C match and otherwise $R(x, C) = \perp$. In this paper, we will consider the circuit class $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ that consists of circuits with input length $\mathsf{inp} := \mathsf{inp}(\lambda)$, depth $\mathsf{dep} := \mathsf{dep}(\lambda)$, and output length $\mathsf{out} := \mathsf{out}(\lambda)$. To do so, we set $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$, $\mathcal{X}_{\mathsf{prm}} = \{0,1\}^{\mathsf{inp}}$, and $\mathcal{Y}_{\mathsf{prm}} = \mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$.

CPFE for circuits. To define CPFE for circuits, we set $\mathcal{X}$ to be the set of all circuits and $\mathcal{Y} = \{0,1\}^*$ and define $R(C, x) = C(x)$ if the length of the string x and the input length of C match and otherwise $R(x, C) = \perp$. In this paper, we will consider the circuit class $\mathcal{C}_{\mathsf{inp}}$ that consists of circuits with input length $\mathsf{inp} := \mathsf{inp}(\lambda)$. To do so, we set $\mathsf{prm} = 1^{\mathsf{inp}}$, $\mathcal{X}_{\mathsf{prm}} = \mathcal{C}_{\mathsf{inp}}$, and $\mathcal{Y}_{\mathsf{prm}} = \{0,1\}^{\mathsf{inp}}$.

Remark 2.9. In the definition of KPFE for circuits, even though the input length, output length, and depth of the circuits in $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ are bounded, the size of the circuits is unbounded. Similar comments hold true for $\mathcal{C}_{\mathsf{inp}}$ in the definition of CPFE.

Remark 2.10. Note that our definition of the KPFE requires that the running time of the encryption algorithm is bounded by $\mathrm{poly}(\lambda, |\mathsf{prm}|) = \mathrm{poly}(\lambda, \mathsf{inp}, \mathsf{dep}, \mathsf{out})$. In particular, the running time should be independent from the size of the circuit being supported by the scheme, which is unbounded. This property is called succinctness in [GTKP+13b].

FE for Turing Machines. To define FE for Turing machines, we set $\mathcal{X} = \{0,1\}^*$, $\mathcal{Y}$ to be set of all Turing machine, and define $R : \mathcal{X} \times \mathcal{Y} \to \{0,1\} \cup \{\perp\}$ as

$$R((x, 1^t), M) = \begin{cases} 1 & \text{if } M \text{ accepts } x \text{ in } t \text{ steps} \\ 0 & \text{otherwise.} \end{cases}.$$

FE for NL. To define FE for NL, we set $\mathcal{X} = \{0,1\}^*$, $\mathcal{Y}$ to be set of all non-deterministic Turing machines with two tapes, one of which encodes the input and can only be read, whereas the other tape can be read as well as written. When we measure the space complexity of the computation, we consider the space being used for the latter tape. We define $R : \mathcal{X} \times \mathcal{Y} \to \{0,1\} \cup \{\perp\}$ as

$$R((x, 1^t, 1^{2^s}), M) = \begin{cases} 1 & \text{if } M \text{ accepts } x \text{ within } t \text{ steps and space } s \\ 0 & \text{otherwise.} \end{cases}.$$

Note that here, s is in the exponent to reflect the idea that the space for the computation is logarithmically bounded.

We recall the following result by Goldwasser et al. [GTKP+13b] that we will use later in Section 4.

Theorem 2.11 ([GTKP+13b])**.** *There exists a KPFE scheme* KPFE *for the circuit class* $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ *with* 1-NA-SIM *security assuming sub-exponential hardness of the LWE problem.*

Remark 2.12. Note that [GTKP+13b] defines their security notion as *full-simulation* based security. However, as stated in [GTKP+13b] only, their *full-simulation* security is equivalent to the *non-adaptive simulation* security definition from [GVW12], where no post-challenge key queries are allowed. The 1-NA-SIM security given in Theorem 2.7 is implied by the same that is adopted for this work. Thus, [GTKP+13b] is 1-NA-SIM secure according to our Theorem 2.7.

3 CPFE with Dynamic Bounded Collusion

In this section, we construct public-key, ciphertext-policy functional encryption (CPFE) schemes with delayed collusion bound. The first scheme supports *unbounded* polynomial-size circuits and achieves NA-SIM security. The second scheme only supports bounded polynomial-size circuits, but achieves stronger AD-SIM security. Both schemes are obtained by first constructing a bounded FE with special efficiency property (Sections 3.2 and 3.3) and then converting it into a scheme with delayed collusion bound (Section 3.4).

3.1 Preparations

Here, we define reusable, dynamic multi-party computation (RDMPC) protocol in the client-server framework, which is introduced by Ananth and Vaikuntanathan [AV19] as a useful notion for the construction of bounded FE. The formal definition of the protocol as a tuple of algorithms with its correctness and security definitions appear in Appendix B.1. Note that we adapt their syntax and the definitions to our setting.

We provide some intuition on how the set of algorithms from Appendix B.1 may be used as a multi-party computation in a client-server framework. Similar to [AV19], here also the setting consists of a single client and N servers. In particular, the client wants to offload an *apriori bounded* number of computations R to these N servers. Each computation is termed as a session. To this end, the protocol consists of two phases as described below:

- An *offline* phase, where the client takes the session count R and a *secret* circuit $C \in \mathcal{C}_{\mathsf{inp}}$ as input, and encodes it (via CktEnc algorithm) as $(\widehat{C}_1, \ldots, \widehat{C}_N)$. For all $k \in [N]$, it then sends the kth encoding $\widehat{C}_k$ to the kth server.

- An *online* phase, that is performed for R sessions. The client wishes to delegate an input $x^{(j)}$ in the jth session for some $j \in [R]$. For this, it encodes $x^{(j)}$ (via InpEnc algorithm) as $\widehat{x}^{(j)}$ and forwards it to all the N servers. Hereon, some n out of N servers (formalized via any subset $S \subseteq [N]$ of size n) may come to do some *local* computation (via the Local algorithm) on their own inputs to get a *partial* output encoding. In particular, the uth server may compute the partial output encoding $\widehat{y}_u^{(j)} = \mathsf{Local}(\widehat{C}_u, \widehat{x}^{(j)})$. The final output $C(x^{(j)})$ is obtained by combining these partial output encodings $\{\widehat{y}_u^{(j)}\}_{u \in S}$ via the public evaluation algorithm Decode.

Crucially, the R input encodings $\{\widehat{x}^{(j)}\}_{j \in [R]}$ are generated with randomness *independent* from that of the offline phase. Further the terms "*reusable*" and "*dynamic*" stems from the respective requirements that the secret circuit encoding must be reusable across all R sessions and the final output may be recovered dynamically by *any* subset S of the N servers in each session. We then recall the result from [AV19] adapted to our setting.

Theorem 3.1 (Adapted from [AV19])**.** *Assuming the existence of one-way functions, there exists an RDMPC protocol with parameter $N = \Theta(R^2\lambda)$, $t = \Theta(R\lambda)$, and $n = \Theta(t)$ for $\mathcal{C}_{\mathsf{inp}}$ with any $\mathsf{inp} = \mathrm{poly}(\lambda)$.*

3.2 Basic Construction

Here, we give a construction of bounded CPFE. The construction supports unbounded size circuits and is secure against bounded collusion. A nice feature of the construction is that the running times of the setup and key generation algorithms are independent from the collusion bound. Looking ahead, we leverage this property to upgrade the construction to be an CPFE scheme with *dynamic bounded collusion property* later in Section 3.4.

In more details, we provide the description of CPFE for the circuit class $\mathcal{C}_\ell$ for arbitrary $\ell = \ell(\lambda)$, where $\mathcal{C}_\ell$ is the set of all polynomial size circuits with input length ℓ. Formally, our FE is for the relation $R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*$ where $\mathsf{prm} = 1^\ell$, $\mathcal{X}_{\mathsf{prm}} := \mathcal{C}_\ell$, and $\mathcal{Y}_{\mathsf{prm}} := \{0,1\}^\ell$ and $R_{\mathsf{prm}}(C, x) = C(x)$, for all $C \in \mathcal{C}_\ell$ and $x \in \{0,1\}^\ell$.

Ingredients. We now describe the underlying building blocks used to obtain our CPFE construction:

1. A reusable, dynamic MPC protocol for $\mathcal{C}_\ell$ denoted by $\mathsf{RDMPC} = (\mathsf{CktEnc}, \mathsf{InpEnc}, \mathsf{Local}, \mathsf{Decode})$ with $N = \Theta(R^2\lambda)$, $t = \Theta(R\lambda)$, and $n = \Theta(t)$. We can instantiate this by the protocol by Ananth and Vaikuntanathan [AV19], which can be based on any one-way function (See Theorem 3.1.). Looking ahead, we will set $R = \lambda$ in our construction and therefore ignore the dependence on R when considering the size of the parameters in the following. We denote the length of an encoding $\widehat{x}$ of $x \in \{0,1\}^\ell$ by $\widehat{\ell} = \mathrm{poly}(\lambda, \ell)$. We also denote the size of the circuit $\mathsf{Local}(\widehat{C}_j, \cdot)$ by $\widehat{s}(\lambda, |C|)$, where $\widehat{C}_j$ is an output of CktEnc on input a circuit C. By the efficiency properties of RDMPC, we have $\widehat{s}(\lambda, |C|) = \mathrm{poly}(\lambda, |\widehat{C}_j|) = \mathrm{poly}(\lambda, |C|)$.

2. A garbled circuit scheme $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$ for circuit class $\mathcal{C}_{\widehat{\ell}}$. We can instantiate this by Yao's scheme [Yao82], which can be based on any one-way function. We assume that a label is represented by a binary string. The length of a label depends on the size of circuits that are garbled. We denote the length of the labels obtained by garbling a circuit of size $\widehat{s}(\lambda, |C|)$ by $L(\lambda, |C|)$. By the efficiency property of the garbled circuit, we have $L(\lambda, |C|) = \mathrm{poly}(\lambda, \widehat{s}(\lambda, |C|)) = \mathrm{poly}(\lambda, |C|)$.

3. An IBE scheme $\mathsf{IBE} = (\mathsf{IBE.Setup}, \mathsf{IBE.Enc}, \mathsf{IBE.KeyGen}, \mathsf{IBE.Dec})$ with IND-CPA security whose identity space and message space are $\{0,1\}^*$. We can instantiate IBE from various standard assumptions including LWE [ABB10, CHKP10], CDH, and Factoring [DG17].

Construction. Let $N := N(\lambda, R)$, $n := n(\lambda, R)$, and $t := t(\lambda, R)$ be the parameters associated with RDMPC. In the following construction, we run the protocol with $R = \lambda$. The basic CPFE scheme $\mathsf{BCPFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ for the circuit class $\mathcal{C}_\ell$ works as follows. Note that the scheme below deviates from the syntax of bounded collusion FE because Setup and KeyGen take the collusion bound Q as binary form, rather than unary form as defined in Theorem 2.4. This change in syntax is to reflect the fact that these algorithms run in polylogarithmic time in Q rather than in polynomial time. Even with this change, we can consider the same correctness requirement and security notions for it.

$\mathsf{Setup}(1^\lambda, 1^\ell, Q)$: On input the security parameter λ, an input length $\ell = \mathrm{poly}(\lambda)$ of the circuit family to be supported, and the upper bound for the collusion $1 \le Q \le 2^\lambda$ in *binary* form and compute $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$. Output the master key pair as $(\mathsf{mpk}, \mathsf{msk}) := (\mathsf{IBE.mpk}, \mathsf{IBE.msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, x, Q)$: On input master secret key $\mathsf{msk} = \mathsf{IBE.msk}$, an input $x \in \{0,1\}^\ell$ and the upper bound for the collusion $1 \le Q \le 2^\lambda$ in *binary* form and do the following:

1. Compute $\widehat{x} \leftarrow \mathsf{InpEnc}(1^\lambda, 1^\lambda, 1^\ell, x)$.
2. Sample $u \leftarrow [Q]$.
3. Sample random set $\Delta \subset [N]$ such that $|\Delta| = n$.
4. For all $j \in \Delta$ and $k \in [\widehat{\ell}]$, generate a secret key as

$$\mathsf{IBE.sk}_{u,j,k,\widehat{x}_k} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{IBE.msk}, (u, j, k, \widehat{x}_k)),$$

   where $\widehat{x}_k$ is the k-th bit of $\widehat{x}$.
5. Output

$$\mathsf{sk} = \left(u, \Delta, \left\{\mathsf{IBE.sk}_{u,j,k,\widehat{x}_k}\right\}_{j\in\Delta, k\in[\widehat{\ell}]}\right). \tag{3.1}$$

$\mathsf{Enc}(\mathsf{mpk}, C, 1^Q)$: On input the master public key $\mathsf{mpk} = \mathsf{IBE.mpk}$, a circuit $C \in \mathcal{C}_\ell$, and the query bound $1 \le Q \le 2^\lambda$ in unary form, do the following:

1. Compute $(\widehat{C}_{i,1}, \ldots, \widehat{C}_{i,N}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^\lambda, 1^\ell, C)$ for $i \in [Q]$.
2. Define the circuit $\mathsf{L}_{i,j}(\cdot) := \mathsf{Local}(\widehat{C}_{i,j}, \cdot)$ with input length $\widehat{\ell} := |\widehat{x}|$.
   For all $i \in [Q]$ and $j \in [N]$, do the following:
   (a) Run the garbling algorithm

$$\{\mathsf{lab}_{i,j,k,b}\}_{k\in[\widehat{\ell}], b\in\{0,1\}} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{L}_{i,j}).$$

   (b) For all $k \in [\widehat{\ell}]$ and $b \in \{0,1\}$, compute

$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,b}\right).$$

3. Output

$$\mathsf{ct} = \{\mathsf{IBE.ct}_{i,j,k,b}\}_{i\in[Q], j\in[N], k\in[\widehat{\ell}], b\in\{0,1\}}. \tag{3.2}$$

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}, 1^Q)$: On input a secret key sk, a ciphertext ct, and the query bound $1 \leq Q \leq 2^\lambda$ in unary form, do the following:

1. Parse the secret key as Eq. (3.1) and the ciphertext as Eq. (3.2).
2. For all $j \in \Delta$, do the following:
   (a) For all $k \in [\widehat{\ell}]$, compute $\mathsf{lab}'_{u,j,k} := \mathsf{IBE.Dec}(\mathsf{IBE.sk}_{u,j,k,\widehat{x}_k}, \mathsf{IBE.ct}_{u,j,k,\widehat{x}_k})$, where $\widehat{x}_k$ is the k-th bit of $\widehat{x}$.
   (b) Compute $\widehat{y}'_{u,j} := \mathsf{GC.Eval}(\{\mathsf{lab}'_{u,j,k}\}_{k\in[\widehat{\ell}]})$
3. Compute and output $z = \mathsf{Decode}(\{\widehat{y}'_{u,j}\}_{j\in\Delta})$.

Remark 3.2 (Intuition for the construction.). The above scheme can be seen as a variant of the FE scheme by Ananth and Vaikuntanathan [AV19], who showed a generic construction of Q-bounded FE scheme from a single-key FE scheme. Here, we instantiate the single-key FE scheme with the construction by Sahai and Seyalioglu [SS10] and then replace the PKE used for encrypting the labels of the garbled circuits inside their construction with IBE. In more details, in our construction above, we run QN instances of the single-key FE scheme. These QN instances are grouped into Q groups each consisting of N instances. In the key generation algorithm of BCPFE, one chooses a group $u \in [Q]$ and then generates n out of N secret keys of the single-key FE instances in the group. To encrypt a message, one generates shares of the message (in our case, a circuit) using RDMPC and then encrypts them using N instances of FE for each group $i \in [Q]$. By the correctness of RDMPC, n secret keys of the single-key FE from a single group can recover the decryption results.

We prove the the correctness and evaluate the efficiency of the above construction in Appendix B.2.

Security. The following theorem asserts the security of our CPFE scheme.

Theorem 3.3. *Assume that* IBE *satisfies* IND-CPA *security,* GC *is a secure garbled circuit scheme, and* RDMPC *is secure. Then,* BCPFE *for the circuit class* $\mathcal{C}_\ell$ *is* NA-SIM*-secure.*

Before the proof, we recall some lemmas from previous works and then provide an overview for the proof.

Lemma 3.4 (Adapted from Claim 1 in [AV19])**.** *Consider random variables* $\gamma_{i,q}$ *for* $i \in [Q]$ *and* $q \in [Q]$ *as follows. For each* $q \in [Q]$, $u^{(q)} \leftarrow [Q]$ *is chosen and we set each* $\gamma_{i,q}$ *as*

$$\gamma_{i,q} \leftarrow \begin{cases} 1, & \text{if } i = u^{(q)} \\ 0 & \text{if } i \neq u^{(q)} \end{cases}$$

Then, we have $\Pr\left[\sum_{q\in[Q]} \gamma_{i,q} \geq \lambda \quad \text{holds for some } i \in [Q] \right] \leq e^{-\Theta(\lambda)}$.

Lemma 3.5 (Adapted from Lemma B.1 in [GVW12])**.** *Let* $R \in \mathbb{N}$. *Set* $t = \Theta(\lambda)$, $N = \Theta(R^2t^2)$ *and* $n = \Theta(t)$. *Choose random subset* $\Delta^{(i)}$ *of* $[N]$ *with size* n *for* $i \in [R]$. *Then, we have*

$$\Pr\left[\left|\bigcup_{i,i'\in[R],i\neq i'} \Delta^{(i)} \cap \Delta^{(i')}\right| \leq t\right] \leq \mathrm{negl}(\lambda).$$

Overview for the proof of Theorem 3.3. We explain the overview for the proof briefly, before describing it in detail. The security proof is similar to that for the construction by Ananth and Vaikuntanathan [AV19]. As we explained in Remark 3.2, our scheme can be seen as parallel QN instances of single-key FE scheme, where the instances are grouped into Q groups each consisting of N instances. Since each user in the system is randomly assigned to a single group, if Q users collude, the number of corrupted users

for each group does not exceed λ with overwhelming probability by Lemma 3.4. Therefore, it suffices to show that each group can tolerate collusion of size up to λ. This is shown via the combination of the security of RDMPC and of the single-key FE. For an instance of FE such that only a single key is generated, we can use the security of the FE. On the other hand, for an instance of FE such that more than one secret key is generated, we cannot use FE security. Our parameter setting allows us to bound the number of the latter instances in each group by t with overwhelming probability using Theorem 3.5. This along with the fact that RDMPC can tolerate up to t corruptions for each group allows us to prove the security of the system.

We then provide the detailed proof of Theorem 3.3 in Appendix B.3.

3.3 A Variant of Basic Construction with AD-SIM Security

Here, we provide a variant of the basic construction in Section 3.2 that satisfies stronger AD-SIM security rather than NA-SIM security at the cost of only supporting circuits with bounded size. Similarly to the construction in Section 3.2, the construction can be upgraded into a construction with delayed collusion bound in Section 3.4. In more details, our construction is for the circuit class $\mathcal{C}_{\ell,s}$, where $\mathcal{C}_{\ell,s}$ is the set of circuits with input length ℓ and size at most s. Formally, our FE is for the relation $R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^*$ where $\mathsf{prm} = (1^\ell, 1^s)$, $\mathcal{X}_{\mathsf{prm}} := \mathcal{C}_{\ell,s}$, and $\mathcal{Y}_{\mathsf{prm}} := \{0,1\}^\ell$ and $R_{\mathsf{prm}}(C,x) = C(x)$, for all $C \in \mathcal{C}_{\ell,s}$ and $x \in \{0,1\}^\ell$.

Ingredients and Parameters. Our construction is the same as that in Section 3, but we require stronger SIM-RSO security for the IBE [KT18]. As shown in [KT18], IBE with SIM-RSO security can be constructed only from IBE with more standard IND-CPA security. Therefore, our construction here can be based on the same set of assumptions as the construction in Section 3. However, since the IBE scheme with SIM-RSO security can only encrypt messages with bounded length, we will have a bound on the length of the labels of GC. This in turn implies that we can only support circuits with bounded size as messages of the CPFE. In the construction, we encrypt circuits with fixed size $s = s(\lambda)$. We denote the size of the circuit $\mathsf{Local}(\widehat{C}_j, \cdot)$ by $\widehat{s}$, where $\widehat{C}_j$ is an output of CktEnc on input a circuit C of size s. We also denote the length of the labels output by GC on input a circuit of size $\widehat{s}$ as L. While $\widehat{s}$ and L are polynomial function in λ and s, we treat them as functions only depend on λ here, since the size of the circuit s is fixed. We assume that the message space of IBE is $\{0,1\}^L$.

We observe that Setup and KeyGen run in time $\mathrm{poly}(\lambda, \ell, s, \log Q)$ by the efficiency properties of IBE and RDMPC. This means that these algorithms can be run even for super polynomial Q. Looking ahead, this property will be used crucially for the full-fledged construction that we show in Section 3.4. We also observe that Enc and Dec run in time $Q \cdot \mathrm{poly}(\lambda, \ell, s)$ by the efficiency properties of IBE, GC, and RDMPC.

The following theorem asserts the security of our CPFE scheme.

Theorem 3.6. *Assume that* IBE *satisfies* IND-CPA *security and* SIM-RSO *security,* GC *is a secure garbled circuit scheme, and* RDMPC *is secure. Then,* BCPFE *for the circuit class* $\mathcal{C}_{\ell,s}$ *is* AD-SIM*-secure.*

Overview for the proof. Before going to the formal proof, we provide its overview. The proof is similar to that for Theorem 3.3. However, here, we have to consider secret key queries after the encryption query. Since we consider simulation-based security definition for BCPFE, this intuitively means that we have to be able to "program" a decryption result into a secret key issued after the encryption query. RDMPC already has this type of capability, since this is designed for constructing FE with AD-SIM security [AV19]. Using this property and the security of the garbled circuits, we can "program" the decryption results into labels of garbled circuits. What remains is to simulate the IBE ciphertexts and secret keys so that the decryption results correspond with the labels generated as above. For this purpose, we use IBE with SIM-RSO security [KT18].

We then provide the detailed proof of Theorem 3.6 in Appendix B.4.

3.4 Full-fledged Construction

In Sections 3.2 and 3.3, we construct bounded collusion CPFE schemes. Here, we show that these schemes can be converted into CPFE schemes with dynamic bounded collusion property. The resulting schemes satisfy the same level of the security and support the same class of circuits as the original schemes. The conversion is the same for both schemes. Let $\mathsf{BCPFE} = (\mathsf{BCPFE.Setup}, \mathsf{BCPFE.KeyGen}, \mathsf{BCPFE.Enc}, \mathsf{BCPFE.Dec})$ be our FE scheme in either Section 3.2 or 3.3 and let $\mathcal{C}_{\mathsf{prm}}$ be the supported circuit class. We have $\mathsf{prm} = 1^\ell$ or $\mathsf{prm} = (1^\ell, 1^s)$. An input to a circuit $C \in \mathcal{C}_{\mathsf{prm}}$ has fixed length $\ell(\lambda)$ in both cases.

Construction. We now construct a CPFE scheme $\mathsf{CPFE} = (\mathsf{CPFE.Setup}, \mathsf{CPFE.KeyGen}, \mathsf{CPFE.Enc}, \mathsf{CPFE.Dec})$ with delayed collusion bound as follows.

$\mathsf{Setup}(1^\lambda, \mathsf{prm})$: On input the security parameter λ, the parameter prm that specifies the circuit family to be supported, do the following:

1. Run $(\mathsf{BCPFE.mpk}_i, \mathsf{BCPFEmsk}_i) \leftarrow \mathsf{BCPFE.Setup}(1^\lambda, \mathsf{prm}, 2^i)$ for $i \in [\lambda]$, where 2^i is represented as a binary number here.
2. Output $(\mathsf{mpk}, \mathsf{msk}) := (\{\mathsf{BCPFE.mpk}_i\}_{i\in[\lambda]}, \{\mathsf{BCPFE.msk}_i\}_{i\in[\lambda]})$.

$\mathsf{KeyGen}(\mathsf{msk}, x)$: On input master secret key $\mathsf{msk} = \{\mathsf{BCPFE.msk}_i\}_{i\in[\lambda]}$ and an input $x \in \{0,1\}^\ell$, do the following:

1. Run $\mathsf{BCPFE.sk}_i \leftarrow \mathsf{BCPFE.KeyGen}(\mathsf{IBE.msk}_i, x, 2^i)$ for $i \in [\lambda]$, where 2^i is represented as a binary number above.
2. Output $\mathsf{sk} := \{\mathsf{BCPFE.sk}_i\}_{i\in[\lambda]}$.

$\mathsf{Enc}(\mathsf{mpk}, C, 1^Q)$: On input the master public key mpk, a circuit $C \in \mathcal{C}_{\mathsf{prm}}$, and the query bound $1 \le Q \le 2^\lambda$ in unary form, do the following:

1. Find u such that $2^{u-1} < Q \le 2^u$.
2. Parse $\mathsf{mpk} \to \{\mathsf{BCPFE.mpk}_i\}_{i\in[\lambda]}$.
3. Compute $\mathsf{BCPFE.ct}_u \leftarrow \mathsf{BCPFE.Enc}\left(\mathsf{BCPFE.mpk}_u, x, 1^{2^u}\right)$ and output $\mathsf{ct} = \mathsf{BCPFE.ct}_u$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}, 1^Q)$: On input a secret key sk, a ciphertext ct, and the query bound $1 \le Q \le 2^\lambda$ in unary form, do the following:

1. Find u such that $2^{u-1} < Q \le 2^u$.
2. Parse $\mathsf{sk} \to \{\mathsf{BCPFE.sk}_i\}_{i\in[\lambda]}$ and $\mathsf{ct} = \mathsf{BCPFE.ct}_u$.
3. Compute and output $\mathsf{BCPFE.Dec}(\mathsf{BCPFE.sk}_u, \mathsf{BCPFE.ct}_u)$.

It is clear that the correctness of the above scheme follows from that of the underlying scheme BCPFE. We analyse the efficiency of the above construction in Appendix B.5.

The following theorem asserts the security of our CPFE scheme.

Theorem 3.7. *If* BCPFE *satisfies* AD-SIM *security as bounded FE scheme, then so does* CPFE *scheme as FE with delayed collusion bound. Similarly, if* BCPFE *satisfies* NA-SIM *security as bounded FE scheme, then so does* CPFE *scheme as FE with delayed collusion bound.*

The proof of the above theorem appears in Appendix B.6.

4 Succinct KPFE with Dynamic Bounded Collusion

In this section, we construct new succinct public key KPFE scheme. The construction substantially improves the security of the previous succinct bounded KPFE schemes [GTKP+13b, Agr17], because it supports delayed collusion bound and satisfies AD-SIM security. Furthermore, our construction can be instantiated only from the LWE assumption, similarly to the previous constructions.

We now describe the parameters for our KPFE scheme in more detail. We construct KPFE scheme for the circuit family $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ containing circuits with input length $\mathsf{inp} = \mathsf{inp}(\lambda)$, depth $\mathsf{dep} = \mathsf{dep}(\lambda)$, output length $\mathsf{out} = \mathsf{out}(\lambda)$ but with arbitrary polynomial size. Formally, we construct an FE scheme for $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$, $\mathcal{X}_{\mathsf{prm}} = \{0,1\}^{\mathsf{inp}}$, $\mathcal{Y}_{\mathsf{prm}} = \mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ and $R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^{\mathsf{out}}$, where $R(x, C) = C(x)$, for all $C \in \mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ and $x \in \{0,1\}^{\mathsf{inp}}$.

Ingredients. The underlying building blocks for our KPFE construction are as follows:

1. A bounded, public key KPFE scheme, denoted by

$$1\mathsf{KPFE} = (1\mathsf{KPFE.Setup}, 1\mathsf{KPFE.KeyGen}, 1\mathsf{KPFE.Enc}, 1\mathsf{KPFE.Dec}),$$

   for the same functionality (i.e., R_{prm} defined above). We require the scheme to satisfy NA-SIM security against a *single* key query. We can instantiate such a scheme with the succinct KPFE scheme from LWE [GTKP+13b] (See Theorem 2.11).

2. A CPFE scheme denoted by

$$\mathsf{CPFE} = (\mathsf{CPFE.Setup}, \mathsf{CPFE.KeyGen}, \mathsf{CPFE.Enc}, \mathsf{CPFE.Dec}),$$

   for bounded polynomial sized circuits that already supports delayed collusion bound property and satisfies AD-SIM security. Namely, CPFE supports a circuit class $\mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$ consisting of circuits with input length inp' and size at most size', where the setting of the parameters is deferred until the description of the construction. Formally, we use FE with $\mathsf{prm}' = 1^{\mathsf{inp}'}$, $\mathcal{X}_{\mathsf{prm}'} = \mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$, $\mathcal{Y}_{\mathsf{prm}'} = \{0,1\}^{\mathsf{inp}'}$ and $R_{\mathsf{prm}'} : \mathcal{X}_{\mathsf{prm}'} \times \mathcal{Y}_{\mathsf{prm}'} \to \{0,1\}^*$, where $R(C, x) = C(x)$, for all $C \in \mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$ and $x \in \{0,1\}^{\mathsf{inp}'}$. We instantiate it from our construction in Section 3.4, which in turn can be based on any IBE.

3. A pseudorandom function $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$. We assume without loss of generality that the input and output space of PRF is the 1KPFE public key space and the randomness space used in 1KPFE.Enc algorithm respectively.

At a high level, our KPFE construction is a compiler that applies the CPFE scheme already satisfying the delayed collusion property and AD-SIM security on top of the single key, succinct KPFE scheme. This makes the resultant KPFE satisfy the delayed collusion property and AD-SIM security as well. We now proceed to the formal construction below.

4.1 Construction

The KPFE scheme $\mathsf{KPFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with delayed collusion bound for the circuit class $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ is as follows.

$\mathsf{Setup}(1^\lambda, \mathsf{prm})$: On input the security parameter λ and the parameters $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$ do the following:

1. Compute $\mathsf{prm}' := (1^{\mathsf{inp}'}, 1^{\mathsf{size}'})$, where $\mathsf{inp}' := \mathsf{inp}'(\lambda, \mathsf{prm})$ is the length of the master public key for 1KPFE output by $1\mathsf{KPFE.Setup}(1^\lambda, \mathsf{prm})$ and $\mathsf{size}' = \mathsf{size}'(\lambda, \mathsf{prm})$ is the size of the circuit $\mathsf{E}_{[x,\mathsf{K}]}$ described in Figure 1.

2. Run $(\mathsf{CPFE.mpk}, \mathsf{CPFE.msk}) \leftarrow \mathsf{CPFE.Setup}(1^\lambda, \mathsf{prm}')$.
3. Output $(\mathsf{mpk}, \mathsf{msk}) := (\mathsf{CPFE.mpk}, \mathsf{CPFE.msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, C)$: On input the master secret key $\mathsf{msk} = \mathsf{CPFE.msk}$ and a circuit $C \in \mathcal{C}_{\mathsf{inp,dep,out}}$, do the following:

1. Compute $(\mathsf{1KPFE.mpk}, \mathsf{1KPFE.msk}) \leftarrow \mathsf{1KPFE.Setup}(1^\lambda, \mathsf{prm})$.
2. Generate a secret key under 1KPFE as $\mathsf{1KPFE.sk} \leftarrow \mathsf{1KPFE.KeyGen}(\mathsf{1KPFE.msk}, C)$.
3. Generate another secret key $\mathsf{CPFE.sk} \leftarrow \mathsf{CPFE.KeyGen}(\mathsf{CPFE.msk}, \mathsf{1KPFE.mpk})$, where $\mathsf{1KPFE.mpk}$ is interpreted as a string in $\{0,1\}^{\mathsf{inp}'}$.
4. Output the full secret key $\mathsf{sk} := (\mathsf{CPFE.sk}, \mathsf{1KPFE.sk})$.

$\mathsf{Enc}(\mathsf{mpk}, x, 1^Q)$: On input the master public key $\mathsf{mpk} = \mathsf{CPFE.mpk}$, an input $x \in \{0,1\}^{\mathsf{inp}}$ and the query bound $1 \leq Q < 2^\lambda$, do the following:

1. Sample a PRF key $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
2. Using prm, construct the circuit $\mathsf{E}_{[x,\mathsf{K}]}(\cdot)$ defined as Figure 1.
3. Compute a ciphertext $\mathsf{CPFE.ct} \leftarrow \mathsf{CPFE.Enc}(\mathsf{CPFE.mpk}, \mathsf{E}_{[x,\mathsf{K}]}, 1^Q)$.
4. Output the ciphertext $\mathsf{ct} := \mathsf{CPFE.ct}$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$: On input a secret key sk associated with circuit C and a ciphertext ct, do the following:

1. Parse the ciphertext $\mathsf{ct} = \mathsf{CPFE.ct}$ and the secret key $\mathsf{sk} = (\mathsf{CPFE.sk}, \mathsf{1KPFE.sk})$, where CPFE.sk and 1KPFE.sk are associated with $\mathsf{1KPFE.mpk} \in \{0,1\}^{\mathsf{inp}'}$ and the circuit C respectively.
2. Compute $y = \mathsf{CPFE.Dec}(\mathsf{CPFE.sk}, \mathsf{CPFE.ct})$.
3. Compute and output $z = \mathsf{1KPFE.Dec}(\mathsf{1KPFE.sk}, y)$.

Circuit $\mathsf{E}_{[x,\mathsf{K}]}$

Hardwired constants: $x \in \{0,1\}^{\mathsf{inp}}$, PRF key K.
On input $\mathsf{str} \in \{0,1\}^{\mathsf{inp}'}$, proceed as follows:

1. Compute $\mathsf{R} = \mathsf{PRF.Eval}(\mathsf{K}, \mathsf{str})$.
2. Compute and output $\mathsf{1KPFE.Enc}(\mathsf{str}, x; \mathsf{R})$, where str is interpreted as a master public key for 1KPFE and R is used as a randomness for the encryption.

Figure 1 : Circuit $\mathsf{E}_{[x,\mathsf{K}]}$.

In Appendix C, we show that the scheme is correct, succinct and satisfies AD-SIM security.

5 Succinct CPFE with Dynamic Bounded Collusion

Recall that our CPFE scheme in Section 3.3 only supports bounded size circuits. Here, we construct a CPFE scheme that can support unbounded size of circuits, while still satisfying the AD-SIM security and delayed collusion property. Compared with the CPFE scheme in Section 3.2, our construction here has bounds on the input and output length and depth of the circuits, while the construction in Section 3.2 does not. On the other hand, our scheme here satisfies stronger AD-SIM security. So, these schemes offer a tradeoff. We can regard our scheme in this section as a dual version of the KPFE scheme in Section 4 in the sense that it satisfies similar properties and the idea for the construction is very similar. Further, the scheme can be instantiated only from the LWE assumption, similar to the one from Section 4. The construction here is used as a building block for the FE scheme for NL in Section 6.3.

Our construction in this section makes use of reusable garbled circuits as a building block. Therefore, we first introduce its definition here and then proceed describing all the building blocks formally before showing the full construction.

5.1 Reusable Garbled Circuits

In this section, we define reusable garbled circuits (RGC) adapting from the definition given in [GTKP+13b]. For $\lambda \in \mathbb{N}$, let $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ denote a family of circuits with input length length $\mathsf{inp} = \mathsf{inp}(\lambda)$, depth $\mathsf{dep} = \mathsf{dep}(\lambda)$ and output length $\mathsf{out} = \mathsf{out}(\lambda)$. For brevity, we denote $\mathsf{prm} = \mathsf{prm}(\lambda) = (1^{\mathsf{inp}(\lambda)}, 1^{\mathsf{dep}(\lambda)}, 1^{\mathsf{out}(\lambda)})$. Then, a reusable garbling scheme $\mathsf{RGC} = (\mathsf{RGC.Garble}, \mathsf{RGC.Enc}, \mathsf{RGC.Eval})$ for $\mathcal{C} = \{\mathcal{C}_{\mathsf{prm}(\lambda)}\}_{\lambda\in\mathbb{N}}$ is as follows:

$\mathsf{RGC.Garble}(1^\lambda, \mathsf{prm}, C)$**:** The PPT garbling algorithm takes the security paremeter λ in unary, prm and a circuit $C \in \mathcal{C}_{\mathsf{prm}}$, and outputs the garbled circuit $\widetilde{C}$ and a secret key gsk.

$\mathsf{RGC.Enc}(\mathsf{gsk}, x)$**:** The PPT encoding algorithm takes the secret key gsk and an input $x \in \{0,1\}^{\mathsf{inp}}$ and outputs an encoding c_x.

$\mathsf{RGC.Eval}(\widetilde{C}, c_x)$**:** The evaluation algorithm takes the garbled circuit $\widetilde{C}$ and encoding c_x associated to x[5] as input and outputs $z \in \{0,1\}^{\mathsf{out}}$.

Definition 5.1 (Correctness). An RGC scheme $\mathsf{RGC} = (\mathsf{RGC.Garble}, \mathsf{RGC.Enc}, \mathsf{RGC.Eval})$ is correct if for all $\lambda \in \mathbb{N}$, $C \in \mathcal{C}_{\mathsf{prm}(\lambda)}$ and $x \in \{0,1\}^{\mathsf{inp}(\lambda)}$,

$$\Pr\left[C(x) \neq z \;\middle|\; \begin{array}{c} (\mathsf{gsk}, \widetilde{C}) \leftarrow \mathsf{RGC.Garble}(1^\lambda, \mathsf{prm}, C), \\ c_x \leftarrow \mathsf{RGC.Enc}(\mathsf{gsk}, x), \mathsf{RGC.Eval}(\widetilde{C}, c_x) = z \end{array} \right] = \mathrm{negl}(\lambda).$$

where probability is taken over the random coins of RGC.Garble and RGC.Enc.

Definition 5.2 (Succinctness). We say that RGC is succinct, if the size of gsk and the running time of RGC.Enc does not grow with size of the circuit supported by the scheme. Formally, we have that for all $\lambda \in \mathbb{N}$, $\mathsf{prm} = \mathsf{prm}(\lambda)$, $C \in \mathcal{C}_{\mathsf{prm}}$ such that $(\mathsf{gsk}, \widetilde{C}) \leftarrow \mathsf{RGC.Garble}(1^\lambda, \mathsf{prm}, C)$,

$$|\mathsf{gsk}|, \mathsf{runtime}(\mathsf{RGC.Enc}(\mathsf{gsk}, x)) \leq \mathrm{poly}(\lambda, |\mathsf{prm}|) = \mathrm{poly}(\lambda, \mathsf{inp}, \mathsf{dep}, \mathsf{out}),$$

where poly is some fixed polynomial.

Definition 5.3 (Circuit privacy with reusability). Consider an RGC scheme $\mathsf{RGC} = (\mathsf{RGC.Garble}, \mathsf{RGC.Enc}, \mathsf{RGC.Eval})$ for a circuit family $\mathcal{C} = \{\mathcal{C}_{\mathsf{prm}(\lambda)}\}_{\lambda\in\mathbb{N}}$. For every stateful PPT adversary A and a stateful PPT simulator Sim, consider the following experiments:

[5] We do not require to hide x in our definition. Moving ahead, this will be made clear explictly in the security requirement in Definition 5.3.

$\mathsf{Exp}^{\mathsf{real}}_{\mathsf{RGC},\mathsf{A}}\left(1^\lambda\right)$:	$\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{RGC},\mathsf{A},\mathsf{Sim}}\left(1^\lambda\right)$:
1: $(\mathsf{prm}, C) \leftarrow \mathsf{A}(1^\lambda)$	1: $(\mathsf{prm}, C) \leftarrow \mathsf{A}(1^\lambda)$
2: $(\widetilde{C}, \mathsf{gsk}) \leftarrow \mathsf{RGC.Garble}(1^\lambda, \mathsf{prm}, C)$	2: $(\widetilde{C}, \mathsf{st}) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{prm}, 1^{\lvert C\rvert})$
3: $b \leftarrow \mathsf{A}^{\mathsf{RGC.Enc}(\mathsf{gsk},\cdot)}(\widetilde{C})$	3: $b \leftarrow \mathsf{A}^{\mathcal{O}[C,\mathsf{st}](\cdot)}(\widetilde{C})$
4: Output b.	4: Output b.

Above, $\mathcal{O}[C, \mathsf{st}](\cdot)$ is an oracle that on input x from A, runs $\mathsf{Sim}\left(1^\lambda, x, C(x), \mathsf{st}\right)$ to return its output.

We then say that RGC satisfies circuit privacy with reusability if there is a PPT simulator Sim such that for every PPT adversary A, the following holds:

$$\left|\Pr[\mathsf{Exp}^{\mathsf{real}}_{\mathsf{RGC},\mathsf{A}}\left(1^\lambda\right) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{RGC},\mathsf{A},\mathsf{Sim}}\left(1^\lambda\right) = 1]\right| = \mathrm{negl}(\lambda)$$

Remark 5.4. Similarly to Definition 2.6, we do not denote adversary A's internal state in the above definition, even though it is stateful for the sake of the simplicity of the notation. On the other hand, the above explicitly denotes the internal state of the simulator Sim by st. We use slightly stronger definition than [GTKP+13b] in that we do not allow the simulator to update the internal state. We adopt this more stronger definition since the construction in [GTKP+13b] satisfies this and it is simpler.

Remark 5.5. Note that in the above definition the input x is explicitly fed to the Sim. Therefore, the definition does not require input privacy and is thus weaker than the one in [GTKP+13b]. However, without loss of generality, we can always instantiate such a circuit-private RGC scheme by including x to be part of the encoding c_x. We then recall the following result by Goldwasser et al. [GTKP+13b].

Theorem 5.6 ([GTKP+13b]). *There exists a circuit-private, succinct RGC scheme* RGC *for the circuit class* $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ *assuming sub-exponential hardness of the LWE problem.*

We now describe the building blocks for our CPFE scheme CPFE. To this end, we start by first specifying its parameters. In particular, CPFE would support the circuit family $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ containing circuits with input length $\mathsf{inp} = \mathsf{inp}(\lambda)$, depth $\mathsf{dep} = \mathsf{dep}(\lambda)$, output length $\mathsf{out} = \mathsf{out}(\lambda)$ but with arbitrary polynomial size. Formally, we construct an FE scheme for $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$, $\mathcal{X}_{\mathsf{prm}} = \mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$, $\mathcal{Y}_{\mathsf{prm}} = \{0,1\}^{\mathsf{inp}}$ and $R_{\mathsf{prm}} : \mathcal{X}_{\mathsf{prm}} \times \mathcal{Y}_{\mathsf{prm}} \to \{0,1\}^{\mathsf{out}}$, where $R(C, x) = C(x)$, for all $C \in \mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ and $x \in \{0,1\}^{\mathsf{inp}}$.

Ingredients. The building blocks for our CPFE construction then are as follows:

1. A succinct RGC scheme denoted by

   $$\mathsf{RGC} = (\mathsf{RGC.Garble}, \mathsf{RGC.Enc}, \mathsf{RGC.Eval}),$$

   for $\mathcal{C}_{\mathsf{prm}}$. We can instantiate such a scheme with the succinct RGC scheme from LWE [GTKP+13b]. However, our construction does not require the RGC to satisfy input privacy. Hence, we use the relaxed Definition 5.3 from Section 5.1, which suffices for our purpose (see Definition 5.2 and Theorem 5.6).

2. A CPFE scheme denoted by

   $$\mathsf{CPFE}' = (\mathsf{CPFE}'.\mathsf{Setup}, \mathsf{CPFE}'.\mathsf{KeyGen}, \mathsf{CPFE}'.\mathsf{Enc}, \mathsf{CPFE}'.\mathsf{Dec}),$$

   for bounded polynomial-sized circuits that already supports delayed collusion bound property and satisfies AD-SIM security. Namely, CPFE supports a circuit class $\mathcal{C}_{\mathsf{inp},\mathsf{size}'}$ consisting of circuits

with input length inp and size at most size', where the setting of the parameters is deferred until the description of the construction. Formally, we use FE with $\mathsf{prm}' = (1^{\mathsf{inp}}, 1^{\mathsf{size}'}), \mathcal{X}_{\mathsf{prm}'} = \mathcal{C}_{\mathsf{inp},\mathsf{size}'}, \mathcal{Y}_{\mathsf{prm}'} = \{0,1\}^{\mathsf{inp}}$ and $R_{\mathsf{prm}'} : \mathcal{X}_{\mathsf{prm}'} \times \mathcal{Y}_{\mathsf{prm}'} \to \{0,1\}^*$, where $R(C, x) = C(x)$, for all $C \in \mathcal{C}_{\mathsf{inp},\mathsf{size}'}$ and $x \in \{0,1\}^{\mathsf{inp}}$. We instantiate it from our construction in Section 3.4, which in turn can be based on any IBE.

3. A pseudorandom function $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$. We assume without loss of generality that the input and output space of PRF is the $\{0,1\}^{\mathsf{inp}}$ and the randomness space used in $\mathsf{RGC.Enc}$ algorithm respectively.

At a high level, our CPFE construction applies the CPFE' scheme satisfying the delayed collusion property and AD-SIM security on top of the succinct, reusable garbled circuit scheme supporting the underlying class of circuits. The resultant CPFE thus satisfies succinctness with the delayed collusion property and AD-SIM security as well. We now proceed to the formal construction below.

5.2 Construction

The CPFE scheme $\mathsf{CPFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ with delayed collusion bound for the circuit class $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ is as follows.

$\mathsf{Setup}(1^\lambda, \mathsf{prm})$: On input the security parameter λ and the parameters $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$ do the following:

1. Compute $\mathsf{prm}' := (1^{\mathsf{inp}}, 1^{\mathsf{size}'})$, where $\mathsf{size}' = \mathsf{size}'(\lambda, \mathsf{prm})$ is the size of the circuit $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}$ described in Figure 2.
2. Run $(\mathsf{CPFE}'.\mathsf{mpk}, \mathsf{CPFE}'.\mathsf{msk}) \leftarrow \mathsf{CPFE}'.\mathsf{Setup}(1^\lambda, \mathsf{prm}')$.
3. Output $(\mathsf{mpk}, \mathsf{msk}) := (\mathsf{CPFE}'.\mathsf{mpk}, \mathsf{CPFE}'.\mathsf{msk})$.

$\mathsf{KeyGen}(\mathsf{msk}, x)$: On input the master secret key $\mathsf{msk} = \mathsf{CPFE}'.\mathsf{msk}$ and an input $x \in \{0,1\}^{\mathsf{inp}}$, compute $\mathsf{CPFE}'.\mathsf{sk} \leftarrow \mathsf{CPFE}'.\mathsf{KeyGen}(\mathsf{CPFE}'.\mathsf{msk}, x)$ and output $\mathsf{sk} := \mathsf{CPFE}'.\mathsf{sk}$.

$\mathsf{Enc}(\mathsf{mpk}, C, 1^Q)$: On input the encryption key $\mathsf{mpk} = \mathsf{CPFE}'.\mathsf{mpk}$, a circuit $C \in \mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ and the query bound $1 \leq Q < 2^\lambda$, do the following:

1. Sample a PRF key $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
2. Using prm, generate $(\widetilde{C}, \mathsf{gsk}) \leftarrow \mathsf{RGC.Garble}(1^\lambda, \mathsf{prm}, C)$ and construct the circuit $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}(\cdot)$ defined as Figure 2.
3. Compute a ciphertext $\mathsf{CPFE}'.\mathsf{ct} \leftarrow \mathsf{CPFE}'.\mathsf{Enc}(\mathsf{CPFE}'.\mathsf{mpk}, \mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}, 1^Q)$.
4. Output the ciphertext $\mathsf{ct} := (\widetilde{C}, \mathsf{CPFE}'.\mathsf{ct})$.

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$: On input a secret key sk associated with input x and a ciphertext ct, do the following:

1. Parse the ciphertext $\mathsf{ct} = (\widetilde{C}, \mathsf{CPFE}'.\mathsf{ct})$ and the secret key $\mathsf{sk} = \mathsf{CPFE}'.\mathsf{sk}$.
2. Compute $y = \mathsf{CPFE}'.\mathsf{Dec}(\mathsf{CPFE}'.\mathsf{sk}, \mathsf{CPFE}'.\mathsf{ct})$.
3. Compute and output $z = \mathsf{RGC.Eval}(\widetilde{C}, y)$.

Circuit $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}$

Hardwired constants: RGC key gsk, PRF key K.
On input $x \in \{0,1\}^{\mathsf{inp}}$, proceed as follows:

1. Compute $\mathsf{R} = \mathsf{PRF.Eval}(\mathsf{K}, x)$.
2. Output $c_x = \mathsf{RGC.Enc}(\mathsf{gsk}, x; \mathsf{R})$, where R is used as a randomness for encoding input x.

Figure 2 : Circuit $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}$.

In Appendix D, we show that the scheme is correct, succinct and satisfies AD-SIM security.

6 FE for Turing Machines with Dynamic Bounded Collusion

In this section, we construct FE for Turing machines from KPFE and CPFE schemes that we have constructed so far. For conciseness, we first provide a generic construction of FE that combines many instance of FE together in Section 6.1. We then construct FE for Turing machines with NA-SIM security in Section 6.2 and FE for NL with AD-SIM security in Section 6.3 by instantiating the generic construction.

6.1 Generalized Bundling of Functionality

Consider an FE scheme $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ for a parameter $\mathsf{prm} = 1^i$ and a relation $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\} \cup \{\bot\}$ for all $i \in \mathbb{N}$. Using such FE, we will construct a new FE with message space $\mathcal{A}$ and key space $\mathcal{B}$. We assume that there exist efficiently computable maps $\mathcal{S} : \mathbb{N} \to 2^{\mathbb{N}}$ and $\mathcal{T} : \mathbb{N} \to 2^{\mathbb{N}}$ such that $\max \mathcal{S}(n)$ and $\max \mathcal{T}(n)$ can be bounded by some fixed polynomial in n. We also assume that there exist maps f with domain $\mathcal{A}$ and g with domain $\mathcal{B}$ such that

$$f(x) \in \prod_{i \in \mathcal{S}(|x|)} \mathcal{X}_i \quad \text{and} \quad g(y) \in \prod_{i \in \mathcal{T}(|y|)} \mathcal{Y}_i,$$

where $|x|$ and $|y|$ are the lengths of x and y as binary strings. Namely, f and g are maps such that

$$f : \mathcal{A} \ni x \mapsto \{f(x)_i \in \mathcal{X}_i\}_{i \in \mathcal{S}(|x|)}, \quad g : \mathcal{B} \ni y \mapsto \{g(y)_i \in \mathcal{Y}_i\}_{i \in \mathcal{T}(|y|)}.$$

Here, we require that the length of $|f(x)|_i$ and $|g(x)|_i$ can be computed from the length of $|x|$ alone and they do not depend on the actual value of x. In this setting, we can construct an FE scheme $\mathsf{BFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ for a two input function $R^{\mathsf{bndl}} : \mathcal{A} \times \mathcal{B} \to \{0,1\}^*$ defined in the following

$$R^{\mathsf{bndl}}(x, y) = \{R_i(f(x)_i, g(y)_i)\}_{i \in \mathcal{S}(|x|) \cap \mathcal{T}(|y|)}, \tag{6.1}$$

where $f(x)_i \in \mathcal{X}_i$ and $g(y)_i \in \mathcal{Y}_i$ are the i-th entries of $f(x)$ and $g(x)$, respectively.

Ingredients. We now describe the underlying building blocks used to obtain our FE construction:

1. A pseudorandom function $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$. We assume that $\mathsf{PRF.Eval}(\mathsf{K}, \cdot)$ has domain $\{0,1\}^\lambda$ and range $\{0,1\}^\lambda$ for any key K output by $\mathsf{PRF.Setup}(1^\lambda)$. The input space $\{0,1\}^\lambda$ can be regarded as $[2^\lambda]$ by the natural bijection between the two sets. We can instantiate PRF from any one-way function [GGM84].

2. An FE scheme $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.KeyGen}, \mathsf{FE.Enc}, \mathsf{FE.Dec})$ for a parameter $\mathsf{prm} = 1^i$ and a relation $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\} \cup \{\bot\}$ for $i \in \mathbb{N}$ with delayed collusion property. We require the scheme to have either NA-SIM or AD-SIM security. We also require that the randomness used by $\mathsf{FE.Setup}(1^\lambda, 1^i)$ is of fixed length λ. This is without loss of generality, since we can generate unbounded length of pseudorandom bits from a binary string R of length λ by regarding R as a PRF key.

3. A garbled circuit scheme $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$. We assume that a label is represented by a binary string and denote its length by $L(\lambda, |C|)$, where C is the circuit being garbled. We can instantiate it by Yao's garbled circuit [Yao82], which can be based on any one-way function.

4. An IBE scheme $\mathsf{IBE} = (\mathsf{IBE.Setup}, \mathsf{IBE.Enc}, \mathsf{IBE.KeyGen}, \mathsf{IBE.Dec})$ with IND-CPA security whose identity space and message space are $\{0,1\}^*$. We assume that the key generation algorithm is deterministic. This is without loss of generality, since we can use PRF to derandomize the key generation algorithm. We can instantiate IBE from various standard assumptions including LWE [ABB10, CHKP10], CDH, and Factoring [DG17].

Construction. We then provide the description of the construction of $\mathsf{BFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ for R^{bndl} above.

$\mathsf{Setup}(1^\lambda)$: On input the security parameter λ, do the following:

1. Run $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$.
2. Sample a PRF key $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$.
3. Output the master key pair as $(\mathsf{mpk}, \mathsf{msk}) := (\mathsf{IBE.mpk}, (\mathsf{IBE.msk}, \mathsf{K}))$.

$\mathsf{KeyGen}(\mathsf{msk}, y)$: On input master secret key $\mathsf{msk} = \mathsf{IBE.msk}$, a key attribute $y \in \mathcal{B}$, do the following:

1. Compute $\mathcal{T}(|y|) \subseteq \mathbb{N}$, where $|y|$ is the length of y as a binary string.
2. Compute $\mathsf{R}_i = \mathsf{PRF.Eval}(\mathsf{K}, i)$ for $i \in \mathcal{T}(|y|)$, where $i \in \mathbb{N}$ is interpreted as a binary string in $\{0,1\}^\lambda$.
3. Run $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i; \mathsf{R}_i)$ for $i \in \mathcal{T}(|y|)$.
4. Compute $g(y) = \{g(y)_i \in \mathcal{Y}_i\}_{i \in \mathcal{T}(|y|)}$.
5. For $i \in \mathcal{T}(|y|)$, compute
$$\mathsf{FE.sk}_i \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, g(y)_i).$$
6. Let $\ell_i := |\mathsf{FE.mpk}_i|$. For all $i \in \mathcal{T}(|y|)$ and $j \in \ell_i$, generate a secret key as
$$\mathsf{IBE.sk}_{i,j} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{IBE.msk}, (i, j, \mathsf{FE.mpk}_{i,j}))$$
where $\mathsf{FE.mpk}_{i,j}$ is the j-th bit of $\mathsf{FE.mpk}_i \in \{0,1\}$ as a binary string.
7. Output
$$\mathsf{sk} = \left(\mathcal{T}(|y|),\ \left\{\mathsf{FE.sk}_i, \{\mathsf{IBE.sk}_{i,j}\}_{j \in [\ell_i]}\right\}_{i \in \mathcal{T}(|y|)}\right). \tag{6.2}$$

$\mathsf{Enc}(\mathsf{mpk}, x, 1^Q)$: On input the encryption key $\mathsf{mpk} = \mathsf{IBE.mpk}$, a message $x \in \mathcal{A}$, and the query bound $1 \leq Q \leq 2^\lambda$ in unary form, do the following:

1. Compute $\mathcal{S}(|x|) \subset \mathbb{N}$, where $|x|$ is the length of x as a binary string.

2. Compute $f(x) = \{f(x)_i\}_{i\in\mathcal{S}(|x|)}$.
3. Do the following for $i \in \mathcal{S}(|x|)$.
   (a) Compute the length ℓ_i of $\mathsf{FE.mpk}_i$. This is done without knowing $\mathsf{FE.mpk}_i$.
   (b) Sample a randomness r_i for the encryption algorithm $\mathsf{FE.Enc}(\mathsf{FE.mpk}_i, f(x)_i, 1^Q; r_i)$. This is done without knowing $\mathsf{FE.mpk}_i$.
   (c) Define a circuit
   $$\mathsf{E}_i(\cdot) := \mathsf{FE.Enc}(\cdot, f(x)_i, 1^Q; r_i)$$
   that takes as input a string $\mathsf{str} \in \{0,1\}^{\ell_i}$ and outputs $\mathsf{FE.Enc}(\mathsf{str}, f(x)_i, 1^Q; r_i)$, where str is interpreted as a master public key of the FE.
   (d) Generate a garbled circuit
   $$\{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_i],b\in\{0,1\}} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{E}_i).$$
   (e) For all $j \in [\ell_i]$ and $b \in \{0,1\}$, compute
   $$\mathsf{IBE.ct}_{i,j,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,b), \mathsf{lab}_{i,j,b}\right).$$
4. Output
$$\mathsf{ct} = \left(\mathcal{S}(|x|),\ \{\mathsf{IBE.ct}_{i,j,b}\}_{i\in\mathcal{S}(|x|),j\in[\ell_i],b\in\{0,1\}}\right). \tag{6.3}$$

$\mathsf{Dec}(\mathsf{ct}, \mathsf{sk})$: On input a secret key sk, a ciphertext ct, and the query bound $1 \le Q \le 2^\lambda$ in unary form, do the following:

1. Parse the secret key as Eq. (6.2) and the ciphertext as Eq. (6.3).
2. For all $i \in \mathcal{S}(|x|) \cap \mathcal{T}(|y|)$ do the following.
   (a) Retrieve $\mathsf{FE.mpk}_i$ from the $\mathsf{FE.sk}_i$.
   (b) For all $j \in [\ell_i]$ compute $\mathsf{lab}'_{i,j} := \mathsf{IBE.Dec}(\mathsf{IBE.sk}_{i,j,\mathsf{FE.mpk}_{i,j}}, \mathsf{IBE.ct}_{i,j,\mathsf{FE.mpk}_{i,j}})$.
   (c) Compute $c_i := \mathsf{GC.Eval}(\{\mathsf{lab}'_{i,j}\}_{j\in[\ell_i]})$.
   (d) Compute $z_i := \mathsf{FE.Dec}(\mathsf{FE.sk}_i, c_i)$
3. Output $\{z_i\}_{i\in\mathcal{S}(|x|)\cap\mathcal{T}(|y|)}$.

In Appendix E, we show that the scheme is correct, satisfies efficiency in that all the algorithms run in time polynomial in their respective input lengths and satisfies AD-SIM (resp., NA-SIM) security, if the same holds for the FE.

6.2 FE for Turing Machines with NA-SIM Security

Here, we provide the construction of FE for Turing machines defined as in Section 2.2.3, which satisfies NA-SIM security. Recall that in FE for Turing machines, a ciphertext is associated with $(x, 1^t)$ and a secret key is for a Turing machine M, and the decryption results to 1 if the machine accepts the input within t steps and 0 otherwise. To construct such a scheme, we start with constructing two schemes with partial functionality and then combine them. The one scheme takes care of the case where $|(x, 1^t)| > |M|$, while the other takes care of the case where $|(x, 1^t)| \le |M|$. Both schemes are obtained by applying the generic conversion in Section 6.1 to the schemes we constructed so far.

6.2.1 The Case of $|(x, 1^t)| > |M|$

We first show that by applying the conversion in Section 6.1 to the CPFE scheme in Section 3.2, we can obtain an FE scheme for Turing machines for the case where $|(x, 1^t)| > |M|$. Formally, we construct an FE for $R^{>} : \mathcal{A} \times \mathcal{B} \to \{0,1\}$, where $\mathcal{A} = \{0,1\}^*$, $\mathcal{B}$ is the set of all Turing machines, and

$$R^{>}((x, 1^t), M) = \begin{cases} 1 & (\text{if } M \text{ accepts } x \text{ in } t \text{ steps}) \wedge \left(|(x, 1^t)| > |M|\right) \\ 0 & \text{otherwise.} \end{cases}.$$

To apply the conversion, we recall that the scheme in Section 3.2 is an FE for $\mathsf{prm} = 1^i$, $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\}$ where $\mathcal{X}_i$ is the set of circuits with input length i, $\mathcal{Y}_i = \{0,1\}^i$, and $R_i(C, x) = C(x)$. We then set $\mathcal{S}$, $\mathcal{T}$, f, and g as

$$\mathcal{S}(i) = \{1, 2, \ldots, i-1\}, \quad \mathcal{T}(i) = \{i\}, \quad f(x, 1^t) = \{U_{i,x,t}(\cdot)\}_{i \in [|(x,1^t)|-1]}, \quad g(M) = M$$

where $U_{i,x,t}(\cdot)$ is defined as Figure 3. The circuit (in particular, Step 2 of the computation) is padded so that the circuit size only depends on $|(x, 1^t)|$. Note that $U_{i,x,t}$ is in $\mathcal{X}_i$ even for x and t with unbounded length, since $\mathcal{X}_i$ contains circuits of unbounded size.

Circuit $U_{i,x,t}$

Hardwired constants: x, t.
On input $M = (Q, \delta, F) \in \{0,1\}^i$, proceed as follows:

1. Parse the input $M \in \{0,1\}^i$ as a description of a Turing machine.
2. Run M on input x for t steps.
3. Output 1 if the state is in F (accept) and 0 otherwise.

Figure 3 : Circuit $U_{i,x,t}$.

Then, by inspection, we can observe that for $\mathcal{X}_i, \mathcal{Y}_i, \mathcal{S}, \mathcal{T}, f, g, \mathcal{A}, \mathcal{B}$ defined as above, R^{bndl} defined as Equation (6.1) is equivalent to $R^{>}$ except for the case of $|(x, 1^t)| \leq |M|$. In this case, the decryption result in FE for R^{bndl} is an empty set $\emptyset$, whereas it should be 0 in FE for $R^{>}$. However, the former can be converted into the latter very easily. To do so, we add the extra step to the decryption algorithm of the former where it outputs 0 if the decryption result is $\emptyset$. Since the original scheme is NA-SIM secure, so is the resulting scheme by Theorem E.2.

6.2.2 The Case of $|(x, 1^t)| \leq |M|$

We next show that by applying the conversion in Section 6.1 to the KPFE scheme in Section 4, we can obtain an FE scheme for Turing machines for the case where $|(x, 1^t)| \leq |M|$. Formally, we construct an FE for $R^{\leq} : \mathcal{A} \times \mathcal{B} \to \{0,1\}$, where $\mathcal{A} = \{0,1\}^*$, $\mathcal{B}$ is the set of all Turing machines, and

$$R^{\leq}((x, 1^t), M) = \begin{cases} 1 & (\text{if } M \text{ accepts } x \text{ in } t \text{ steps}) \wedge \left(|(x, 1^t)| \leq |M|\right) \\ 0 & \text{otherwise.} \end{cases}.$$

To apply the conversion, we observe that the scheme in Section 4 can be used as an FE for $\mathsf{prm} = 1^i$, $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\}$ where $\mathcal{X}_i$ is the set of circuits with input length i, depth $i \cdot \lambda$, and output length 1 and $\mathcal{Y}_i = \{0,1\}^i$, and $R_i(C, x) = C(x)$. We then set $\mathcal{S}$, $\mathcal{T}$, f, and g as

$$\mathcal{S}(i) = i, \quad \mathcal{T}(i) = \{1, 2, \ldots, i\}, \quad f(x, 1^t) = (x, 1^t), \quad g(M) = \{U_{i,M}(\cdot)\}_{i \in [M]},$$

where $U_{i,M}(\cdot)$ is defined as Figure 4. Here, we check that $U_{i,M}(\cdot)$ is in $\mathcal{Y}_i$. Recall that even though $\mathcal{Y}_i$ supports circuits with unbounded size, it has a bound on the depth of the circuit. We therefore argue that the depth of $U_{i,M}(\cdot)$ does not exceed $i\lambda$, even for unbounded size $|M|$. We evaluate the depth of Step 2 of the circuit, since this is the only non-trivial step. By Lemma 2.2, this step can be implemented by a circuit with depth

$$t \cdot \mathrm{poly}(\log |x|, \log t, \log |M|) \leq i \cdot \mathrm{poly}(\log \lambda) \leq i \cdot \lambda$$

Circuit $U_{i,M}$

Hardwired constants: Description of a Turing Machine M.
On input $y \in \{0,1\}^i$, proceed as follows:

1. Parse the input $y \in \{0,1\}^i$ as $(x, 1^t)$.
2. Otherwise, run M on input x for t steps.
3. Output 1 if the state is in F (accept) and 0 otherwise.

Figure 4 : Circuit $U_{i,M}$.

Then, by inspection, we can observe that for $\mathcal{X}_i, \mathcal{Y}_i, \mathcal{S}, \mathcal{T}, f, g, \mathcal{A}, \mathcal{B}$ defined as above, R^{bndl} defined as Equation (6.1) is equivalent to $R^{\leq}$ except for the case of $|(x, 1^t)| > |M|$. In this case, the decryption result in FE for R^{bndl} is an empty set $\emptyset$, whereas it should be 0 in FE for $R^{>}$. However, the former can be converted into the latter by adding the extra step to the decryption algorithm where it outputs 0 if the decryption result is $\emptyset$. This gives us the construction of FE for $R^{\leq}$. Since the original scheme is AD-SIM secure, so is the resulting scheme by Theorem E.2.

6.2.3 Putting the Pieces Together

Here, we combine the two schemes we considered so far to obtain the full-fledged scheme. We set $\mathcal{A} = \{0,1\}^*$ and $\mathcal{B}$ to be the set of all Turing machines. We also set $R_1 = R^{>}$, $R_2 = R^{\leq}$, $\mathcal{X}_i = \mathcal{A}$, $\mathcal{Y}_i = \mathcal{B}$ for $i = 1, 2$. We have already constructed schemes for R_1 and R_2 and now combine them. To do so, we set $\mathcal{S}, \mathcal{T}, f$, and g as

$$\mathcal{S}(i) = \{1,2\}, \quad \mathcal{T}(i) = \{1,2\}, \quad f(x, 1^t) = \{(x, 1^t), (x, 1^t)\}, \quad g(M) = \{M, M\}$$

We observe that for $\mathcal{X}_i, \mathcal{Y}_i, \mathcal{S}, \mathcal{T}, f, g, \mathcal{A}, \mathcal{B}$ defined as above, R^{bndl} defined as Equation (6.1) is

$$R^{\mathsf{bndl}}((x, 1^t), M) = \begin{cases} (1,0) & (\text{if } M \text{ accepts } x \text{ in } t \text{ steps}) \wedge \left(|(x, 1^t)| > |M|\right) \\ (0,1) & (\text{if } M \text{ accepts } x \text{ in } t \text{ steps}) \wedge \left(|(x, 1^t)| \leq |M|\right) \\ (0,0) & \text{otherwise.} \end{cases}.$$

An FE for the above relation is not exactly the same as the FE for Turing machines we defined in Section 2.2.3. However, the former readily implies the latter. To do so, we add the extra step to the decryption algorithm of the former where it checks whether there is 1 in the left or right slot of the decryption result and outputs 1 if there is. Otherwise, it outputs 0.

It is obvious that the obtained scheme satisfies correctness and efficiency requirements if so does the underlying schemes. As for the security, we can see by Theorem E.2 that the resulting scheme is NA-SIM secure if we combine an AD-SIM secure scheme for $R^{\leq}$ with an NA-SIM secure scheme for $R^{>}$.

Remark 6.1. One might think that FE for R^{bndl} constructed above leaks more information on $(x, 1^t)$ than FE for Turing machines and so is our final scheme, because the decryption result can be $\emptyset$, in addition to 0 or 1 and the decryptor can know whether $|(x, 1^t)| > |M|$ or not. However, it is not the case since whether the decryption result is $\emptyset$ or not can be checked only from the length of the string $|(x, 1^t)|$, which is not meant to be hidden in our definition (and other standard definitions) of FE.

To sum up, we have the following theorem:

Theorem 6.2. *We have FE for Turing machines with delayed collusion property that satisfies* NA-SIM *security from the sub-exponential LWE assumption.*

6.3 FE for NL with AD-SIM Security

Here, we provide the construction of FE for NL. Namely, a ciphertext encrypts a string $(x, 1^t, 1^{2^s})$, where x is a string, t is the time bound, s is the space bound for the computation and a secret key is associated with a non-deterministic Turing machine M. The decryption is possible if M accepts x within t steps and the space used for the computation does not exceed s. The construction is less expressive than FE scheme for Turing machines in Section 6.2, but it satisfies the stronger AD-SIM security. The idea for the construction is very similar to Section 6.2. We consider two schemes that complement each other and then combine them.

Transition Matrix. Before describing the schemes, we need some preparations. Similarly to [LL20], we represent the computation of $M(x)$ as a multiplication of matrices. To do so, let us enumerate all the possible internal configurations that may appear when we run $M = (Q, \delta, F)$ on input x with the space for the computation being bounded by s. As an internal configuration, we have $|x|$ and s choices for the input and work tape pointers, respectively. We also have $|Q|$ choices for the possible state, and 2^s possible choices for the contents of the work tape. Therefore, we have

$$N := s2^s \cdot |x| \cdot |Q|$$

possible internal configurations. We associate each $i \in [N]$ with such configuration and represent the configuration by a vector $\mathbf{e}_i$, which is a unit vector whose entries are all 0 except for the i-th entry that is set to be 1. We also define the matrix $\mathsf{Mat}(M, x, s)$ as

$$\mathsf{Mat}(M, x, s)_{i,j} := \begin{cases} 1 & \text{if the configuration } i \text{ can reach } j \text{ in one step by } \delta \\ 0 & \text{otherwise} \end{cases},$$

where $\mathsf{Mat}(M, x, s)_{i,j}$ is the (i, j)-th entry of the matrix. Furthermore, we consider a special matrix multiplication over $\{0, 1\}$, where the multiplication $\mathbf{A} \cdot \mathbf{B} \in \{0, 1\}^{N_1 \times N_3}$ of two matrices $\mathbf{A} \in \{0, 1\}^{N_1 \times N_2}$ and $\mathbf{B} \in \{0, 1\}^{N_2 \times N_3}$ is defined as

$$(\mathbf{A} \cdot \mathbf{B})_{i,j} = \bigvee_{k \in [N_2]} (\mathbf{B}_{i,k} \wedge \mathbf{C}_{k,j}),$$

where we denote the (i, j)-th entry of a matrix $\mathbf{D}$ by $\mathbf{D}_{i,j}$ above. The multiplication can be defined for any size of matrices and in particular, also defined for computations involving vectors. We then define $\mathbf{e}_{stt}$ as the vector corresponding to the initial state of the computation and $\mathbf{u}_{acc}$ as

$$\mathbf{u}_{acc} = \sum_{i \in [N] : i \text{ encodes accepting state}} \mathbf{e}_i.$$

Then, we observe that

$$\mathbf{e}_{stt}^\top \cdot (\mathsf{Mat}(M, x, s)_{i,j})^t \cdot \mathbf{u}_{acc} = \begin{cases} 1 & \text{if } M \text{ accepts } x \text{ within } t \text{ steps and space } s \\ 0 & \text{otherwise} \end{cases}$$

holds from the property of the transition matrix, where we use the special multiplication above.

6.3.1 The Case of $|(x, 1^t, 1^{2^s})| > |M|$

We first show that by applying the conversion in Section 6.1 to the CPFE scheme in Section 5, we can obtain an FE scheme for NL for the case where $|(x, 1^t, 1^{2^s})| > |M|$. Formally, we construct an FE for $R^{>} : \mathcal{A} \times \mathcal{B} \to \{0,1\}$, where $\mathcal{A} = \{0,1\}^*$, $\mathcal{B}$ is the set of all non-deterministic Turing machines, and

$$R^{>}((x, 1^t, 1^{2^s}), M) = \begin{cases} 1 & (\text{if } M \text{ accepts } x \text{ within } t \text{ steps and space } s) \wedge \left(|(x, 1^t, 1^{2^s})| > |M|\right) \\ 0 & \text{otherwise.} \end{cases}.$$

To apply the conversion, we observe that the scheme in Section 5 can be used to construct an FE for $\mathsf{prm} = 1^i$, $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\} \cup \{\bot\}$ where $\mathcal{X}_i$ is the set of circuits with input length i, depth λ, and output length 1, $\mathcal{Y}_i = \{0,1\}^i$, and $R_i(C, x) = C(x)$. We then set $\mathcal{S}$, $\mathcal{T}$, f, and g as

$$\mathcal{S}(i) = \{1, 2, \ldots, i-1\}, \quad \mathcal{T}(i) = \{i\}, \quad f(x, 1^t, 1^{2^s}) = \{U_{i,x,t,s}(\cdot)\}_{i \in [|(x,1^t,1^{2^s})|-1]}, \quad g(M) = M$$

where $U_{i,x,t,s}(\cdot)$ is defined as Figure 5. The circuit may be padded so that it does not leak more information about the hardwired constants (x, t, s) beyond $|(x, 1^t, 1^{2^s})|$.

We now show that f is a valid map. Namely, $U_{i,x,t,s}$ is in $\mathcal{X}_i$ even for x and t with unbounded length. Recall that $\mathcal{X}_i$ has a bound on the depth of the circuits it supports, even though it does not have such a bound on the size. We argue that the depth of $U_{i,x,t,s}$ is bounded by λ. To do so, we evaluate the depth of each computation step of $U_{i,x,t,s}$. First, Step 1 of the computation is trivial. Step 2 can be implemented by a circuit of depth $\mathrm{poly}(\log |x|, \log s, \log |M|)$ by Lemma 2.3. Step 3 can be executed by $O(\log t)$ multiplication of matrices of size $N \times N$, which can be implemented with depth $O(\log N)$. Therefore, this step can be computed with depth $O(\log N \log t)$. Step 4 can also be implemented with depth $O(\log N)$. Therefore, the entire computation can be implemented with depth

$$\mathrm{poly}(\log s, \log |x|, \log t, \log N, \log |M|) \leq \mathrm{poly}(\log \lambda) \leq \lambda$$

as desired.

Circuit $U_{i,x,t,s}$

Hardwired constants: x, t, s.
On input $M \in \{0,1\}^i$, proceed as follows:

1. Parse the input $M = (Q, \delta, F)$ as a description of a Turing machine.
2. Compute $\mathsf{Mat}(M, x, s)_{j,k}$ for all $j, k \in [N]$ in parallel, where $N = s2^s \cdot |x| \cdot |Q|$.
3. Compute $\mathbf{A} := \mathsf{Mat}(M, x, s)^t$.
4. Compute and output $\mathbf{e}_{stt}^{\top} \cdot \mathbf{A} \cdot \mathbf{u}_{acc}$.

Figure 5 : Circuit $U_{i,x,t,s}$.

Then, by inspection, we can observe that for $\mathcal{X}_i, \mathcal{Y}_i, \mathcal{S}, \mathcal{T}, f, g, \mathcal{A}, \mathcal{B}$ defined as above, R^{bndl} defined as Equation (6.1) is equivalent to $R^{>}$ except for the case of $|(x, 1^t, 1^{2^s})| \leq |M|$. However, this case can be handled by replacing the $\emptyset$ by 0, similarly to Section 6.2. This gives us the construction of FE for $R^{>}$. Since the original scheme is AD-SIM secure, so is the resulting scheme by Theorem E.2.

6.3.2 The Case of $|(x, 1^t, 1^{2^s})| \leq |M|$

We next show that by applying the conversion in Section 6.1 to the KPFE scheme in Section 4, we can obtain an FE scheme for Turing machine for the case where $|(x, 1^t, 1^{2^s})| \leq |M|$. Formally, we construct an FE for $R^{\leq} : \mathcal{A} \times \mathcal{B} \to \{0,1\} \cup \{\bot\}$, where $\mathcal{A} = \{0,1\}^*$, $\mathcal{B}$ is the set of all non-deterministic Turing machines, and

$$R^{\leq}((x, 1^t, 1^{2^s}), M) = \begin{cases} 1 & (\text{if } M \text{ accepts } x \text{ within } t \text{ steps and space } s) \wedge \left(|(x, 1^t, 1^{2^s})| \leq |M|\right) \\ 0 & \text{otherwise.} \end{cases}.$$

To apply the conversion, we observe that the scheme in Section 4 can be used as an FE for $\mathsf{prm} = 1^i$, $R_i : \mathcal{X}_i \times \mathcal{Y}_i \to \{0,1\}$ where $\mathcal{X}_i$ is the set of circuits with input length i, depth $i \cdot \lambda$, and output length 1 and $\mathcal{Y}_i = \{0,1\}^i$, and $R_i(C, x) = C(x)$. We then set $\mathcal{S}$, $\mathcal{T}$, f, and g as

$$\mathcal{S}(i) = i, \quad \mathcal{T}(i) = \{1, 2, \ldots, i\}, \quad f(x, 1^t, 1^{2^s}) = (x, 1^t, 1^{2^s}), \quad g(M) = \{U_{i,M}(\cdot)\}_{i \in [M]},$$

where $U_{i,M}(\cdot)$ is defined as Figure 6.

We now show that g is a valid map. Namely, $U_{i,M}$ is in $\mathcal{Y}_i$ even for M with unbounded size. In particular, we have to show that the depth of the circuit is bounded by λ. This can be shown by the same argument as for $U_{i,x,t,s}$, since both circuits compute $\mathbf{e}_{stt}^{\top} \cdot \mathsf{Mat}(M, x, s)^t \cdot \mathbf{u}_{acc}$ from (x, t, s, M) in exactly the same way.

Circuit $U_{i,M}$

Hardwired constants: Description of a Turing Machine M.
On input $y \in \{0,1\}^i$, proceed as follows:

1. Parse the input $y \in \{0,1\}^i$ as $(x, 1^t, 1^{2^s})$.
2. Compute $\mathsf{Mat}(M, x, s)_{j,k}$ for all $j, k \in [N]$ in parallel, where $N = s2^s \cdot |x| \cdot |Q|$.
3. Compute $\mathbf{A} := \mathsf{Mat}(M, x, s)^t$.
4. Compute and output $\mathbf{e}_{stt}^{\top} \cdot \mathbf{A} \cdot \mathbf{u}_{acc}$.

Figure 6 : Circuit $U_{i,M}$.

Then, by inspection, we can observe that for $\mathcal{X}_i, \mathcal{Y}_i, \mathcal{S}, \mathcal{T}, f, g, \mathcal{A}, \mathcal{B}$ defined as above, R^{bndl} defined as Equation (6.1) is equivalent to $R^{\leq}$ except for the case of $|(x, 1^t, 1^{2^s})| > |M|$. However, this case can be handled by replacing the $\emptyset$ by 0, similarly to Section 6.2. This gives us the construction of FE for $R^{\leq}$. Since the original scheme is AD-SIM secure, so is the resulting scheme by Theorem E.2.

6.3.3 Putting the Pieces Together

Here, we combine the two schemes we considered so far to obtain the full-fledged scheme. We set $\mathcal{A} = \{0,1\}^*$ and $\mathcal{B}$ to be the set of all Turing machines. We also set $R_1 = R^{>}$, $R_2 = R^{\leq}$. $\mathcal{X}_i = \mathcal{A}$, $\mathcal{Y}_i = \mathcal{B}$ for $i = 1, 2$. We have already constructed schemes for R_1 and R_2 and now combine them. To do so, we set $\mathcal{S}$, $\mathcal{T}$, f, and g as

$$\mathcal{S}(i) = \{1, 2\}, \quad \mathcal{T}(i) = \{1, 2\}, \quad f(x, 1^t) = \{(x, 1^t, 1^{2^s}), (x, 1^t, 1^{2^s})\}, \quad g(M) = \{M, M\}$$

We observe that for $\mathcal{X}_i, \mathcal{Y}_i, \mathcal{S}, \mathcal{T}, f, g, \mathcal{A}, \mathcal{B}$ defined as above, R^{bndl} defined as Equation (6.1) is

$$R^{\mathsf{bndl}}((x, 1^t, 1^{2^s}), M) = \begin{cases} (1,0) & (\text{if } M \text{ accepts } x \text{ within } t \text{ steps and space } s) \wedge \left(|(x, 1^t, 1^{2^s})| > |M|\right) \\ (0,1) & (\text{if } M \text{ accepts } x \text{ within } t \text{ steps and space } s) \wedge \left(|(x, 1^t, 1^{2^s})| \leq |M|\right) \\ (0,0) & \text{otherwise.} \end{cases}.$$

To obtain the FE scheme for NL, we add an extra step for the decryption algorithm of the FE scheme for R^{bndl} obtained above, where we output 1 if the decryption result for either left or right slot is 1 and 0 otherwise. To sum up, we have the following theorem:

Theorem 6.3. *We have FE for NL with delayed collusion property that satisfies* AD-SIM *security from the sub-exponential LWE assumption.*

7 Necessity of IBE

In this paper, we mainly focus on the construction of FE with delayed collusion bound property, which is stronger than traditional bounded collusion security. Whereas the construction of the latter can be based on the minimal assumption of plain PKE [AV19], we need stronger primitive of IBE for the former in our constructions. A natural question is whether we can base the construction of the former on weaker primitives such as PKE. In this section, we observe that the usage of IBE is unavoidable by showing that FE with delayed collusion property even for very small class of functionality already implies IBE. This means that our constructions of CPFE in Section 3 are optimal, in the sense that we cannot base the scheme on weaker primitives.

To show the above implication, we use the result by Brakerski et al. [BLSV18], which showed that the primitive called weakly compact IBE can be converted into the standard IBE. Weakly compact IBE is a special kind of IBE such that the identity space is only polynomially large, but the master public key is sublinear in the size of the identity space. Namely, for identity space $[N]$ with $N = \mathrm{poly}(\lambda)$, the size of the master public key is $N^{1-\epsilon}\,\mathrm{poly}(\lambda)$ for some constant $1 \geq \epsilon > 0$. As for the security, we require IND-CPA security as per Definition A.8.

Given the aforementioned result, it suffices to show that weakly compact IBE is implied by an FE scheme with delayed collusion property for some simple function class. In particular, let us consider an FE scheme for the function family $\{R_{\mathsf{prm}}\}_{\mathsf{prm}}$, where $\mathsf{prm} = 1^\lambda$, $\mathcal{X}_{\mathsf{prm}} = \{0,1\}^\lambda \times \{0,1\}$, $\mathcal{Y}_{\mathsf{prm}} = \{0,1\}^\lambda$, and $R_{\mathsf{prm}}((x,b),y) = (x,b)$ if $x = y$ and $R_{\mathsf{prm}}((x,b),y) = x$ otherwise. We require the FE scheme to satisfy NA-SIM with delayed collusion bound. It is easy to see that this FE already satisfies the functionality of IBE by interpreting b as a message to be encrypted and x and y as identities associated with the ciphertext and secret key, respectively. However, this FE scheme does not satisfy the standard collusion resistance as IBE if we fix the collusion bound Q for the encryption to be some polynomial, since the delayed collusion bound property does not guarantee the security for an adversary who makes more than Q secret key queries.

Nevertheless, we observe that the above scheme can be used as weakly compact IBE if we restrict the identity space. To show this, let us denote the length of the master public key by $L(\lambda)$ and restrict the identity space to be $[N] \subseteq \{0,1\}^\lambda$, where $N = \lceil L^{1/\epsilon} \rceil$ for some constant $0 < \epsilon < 1$ and $[N]$ is embedded into $\{0,1\}^\lambda$ by the natural bijection between $\{0,1\}^\lambda$ and $[2^\lambda]$. We then have the encryptor always choose $Q = N - 1$.

We argue that the above construction is secure weakly compact IBE. To see this, we first observe that the size of the master public key is $L(\lambda) \leq N^\epsilon$, which is sublinear in the number of all users in the system. Next, we argue that the scheme satisfies IND-CPA security as weakly compact IBE. To see this, we first observe that NA-SIM security of the FE scheme implies corresponding indistinguishability-based security notion that we call NA-IND, where we require the indistinguishability of two ciphertexts

encrypting two different messages m_0 and m_1 instead of requiring that there is a simulator that generates a simulated ciphertext indistinguishable from the real one. This implication is standard and was observed by [O'N10, GVW12] for general FE. Note that there is no bound on the number of key queries any more, since we allow the adversary to collude with at most $Q = N - 1$ users (i.e., all but the target identity). The only difference between NA-IND security and the standard IND-CPA security is that in the latter, the adversary is allowed to make secret key queries after the encryption query. However, since the number of users in the system is polynomially bounded, both security notions are in fact equivalent. In particular, these security notions are equivalent to the selective variant of the security notion where the adversary chooses its target identity at the beginning of the game. This can be seen by a simple reduction that guesses the target identity for which the ciphertext is generated and queries all the other identities.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.

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

[Agr19] Shweta Agrawal. Indistinguishability obfuscation minus multilinear maps: New methods for bootstrapping and instantiation. In *Eurocrypt*, 2019.

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

[AJL+19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: io from lwe, bilinear maps, and weak pseudorandomness. In *Crypto*, 2019.

[AM18] Shweta Agrawal and Monosij Maitra. Fe and io for turing machines from minimal assumptions. In *TCC*, 2018.

[AMY19] Shweta Agrawal, Monosij Maitra, and Shota Yamada. Attribute based encryption (and more) for nondeterministic finite automata from lwe. In *CRYPTO*, 2019.

[AR17] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In *TCC*, 2017.

[AS16] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *TCC*, 2016.

[AS17] Shweta Agrawal and Ishaan Preet Singh. Reusable garbled deterministic finite automata from learning with errors. In *ICALP*, 2017.

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

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

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *ASIACRYPT*, 2012.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, 2012.

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

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

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, 2010.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, 2001.

[DG17] Nico Döttling and Sanjam Garg. Identity-based encryption from the diffie-hellman assumption. In *CRYPTO*, 2017.

[GGLW21] Rachit Garg, Rishab Goyal, George Lu, and Brent Waters. Dynamic collusion bounded functional encryption from identity-based encryption. Personal Communication, 2021.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, 1984.

[GJLS20] Romain Gay, Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. *IACR Cryptology ePrint Archive, Report 2020/764*, 2020.

[GKW16] Rishab Goyal, Venkata Koppula, and Brent Waters. Semi-adaptive security and bundling functionalities made generic and easy. In *TCC*, 2016.

[GTKP+13a] S. Goldwasser, Y. Tauman Kalai, R. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run turing machines on encrypted data. In *CRYPTO*, 2013.

[GTKP+13b] S. Goldwasser, Y. Tauman Kalai, R. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions from multiparty computation. In *CRYPTO*, 2012.

[JLMS19] Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials over r to build io. In *EUROCRYPT*, 2019.

[JLS20] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. *Cryptology ePrint Archive, Report 2020/1003*, 2020.

[KT18] Fuyuki Kitagawa and Keisuke Tanaka. Key dependent message security and receiver selective opening security for identity-based encryption. In *PKC*, 2018.

[LL20] Huijia Lin and Ji Luo. Compact adaptively secure abe from k-lin: Beyond nc1 and towards nl. In *EUROCRYPT*, 2020.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptol.*, 22(2), 2009.

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

[PF79] Nicholas Pippenger and Michael J Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2), 1979.

[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*, 2005.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, 1982.

APPENDICES

A Additional Preliminaries

A.1 Pseudorandom Functions

Let $\{\mathsf{K}_\lambda, \mathcal{X}_\lambda, \mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ordered family of sets and let $\mathsf{PRF} = (\mathsf{PRF.Setup}, \mathsf{PRF.Eval})$ describe a function family $\mathcal{F} := \{\mathcal{F}_\lambda : \mathsf{K}_\lambda \times \mathcal{X}_\lambda \to \mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ with the following syntax:

- $\mathsf{PRF.Setup}(1^\lambda)$: The PPT algorithm takes as input the unary description of the security parameter and outputs a key $\mathsf{K} \in \mathsf{K}_\lambda$.
- $\mathsf{PRF.Eval}(\mathsf{K}, x)$: The deterministic, polynomial time algorithm takes a key $\mathsf{K} \in \mathsf{K}_\lambda$ and $x \in \mathcal{X}_\lambda$ as inputs and outputs $\mathcal{F}_\lambda(\mathsf{K}, x) = \mathsf{R} \in \mathcal{Y}_\lambda$.

The sets $\mathsf{K}_\lambda, \mathcal{X}_\lambda$ and $\mathcal{Y}_\lambda$ define the *key space*, *domain*, and *range* of the set of function family $\mathcal{F}$ respectively. We say $\mathcal{F}$ is a pseudorandom function family if for every PPT adversary A, there exists a negligible function $\mathrm{negl}(\cdot)$ such that

$$\mathsf{Adv}_{\mathsf{PRF},\mathsf{A}}(\lambda) := \left| \Pr_{\mathsf{K}\leftarrow\mathsf{PRF.Setup}(1^\lambda)} \left[\mathsf{A}^{\mathsf{PRF.Eval}(\mathsf{K},\cdot)}(1^\lambda) = 1\right] - \Pr_{f_\lambda\leftarrow\mathcal{U}_\lambda} \left[\mathsf{A}^{f(\cdot)}(1^\lambda) = 1\right] \right| \leq \mathrm{negl}(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, where $\mathcal{U}_\lambda := \{f_\lambda : \mathcal{X}_\lambda \times \mathcal{Y}_\lambda\}$ is the set of all functions from $\mathcal{X}_\lambda$ to $\mathcal{Y}_\lambda$.

A.2 Garbled Circuits

Our definition of garbled circuits is based upon the one considered in [GKW16]. For $\lambda \in \mathbb{N}$, let $\mathcal{C}_{\mathsf{inp}}$ denote a family of circuits with inp bit inputs and $\mathcal{C} = \{\mathcal{C}_{\mathsf{inp}(\lambda)}\}_{\lambda\in\mathbb{N}}$. A garbling scheme $\mathsf{GC} = (\mathsf{GC.Garble}, \mathsf{GC.Eval})$ for $\mathcal{C}$ consists of two algorithms:

- $\mathsf{GC.Garble}(C, 1^\lambda)$: The PPT garbling algorithm takes as input the security parameter λ and a circuit $C \in \mathcal{C}_{\mathsf{inp}(\lambda)}$. It outputs $2\cdot\mathsf{inp}$ labels $\{\mathsf{lab}_{i,b}\}_{i\in[\mathsf{inp}],b\in\{0,1\}}$.
- $\mathsf{GC.Eval}(\{\mathsf{lab}_i\}_{i\in[\mathsf{inp}]})$: The evaluation algorithm takes as input an inp labels $\{\mathsf{lab}_i\}_{i\in[\mathsf{inp}]}$ and outputs y.

Definition A.1 (Correctness). A garbling scheme GC for circuit family $\{\mathcal{C}_{\mathsf{inp}(\lambda)}\}_{\lambda\in\mathbb{N}}$ is correct if for all $C \in \mathcal{C}_{\mathsf{inp}(\lambda)}$ and all $x \in \{0,1\}^{\mathsf{inp}(\lambda)}$, we have

$$\Pr\left[\mathsf{GC.Eval}\left(\{\mathsf{lab}_{i,x_i}\}_{i\in[\mathsf{inp}]}\right) \neq C(x) : \left(\{\mathsf{lab}_{i,b}\}_{i\in[\mathsf{inp}],b\in\{0,1\}}\right) \leftarrow \mathsf{GC.Garble}(1^\lambda, C)\right] = \mathrm{negl}(\lambda),$$

where the probability is taken over the random coins of $\mathsf{GC.Garble}$.

Definition A.2 (Security). A garbling scheme $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$ for $\mathcal{C} = \{\mathcal{C}_{\mathsf{inp}(\lambda)}\}_{\lambda\in\mathbb{N}}$ is said to be a secure garbling scheme if there exists a PPT simulator $\mathsf{GC.Sim}$ such that for all $\lambda \in \mathbb{N}, \mathsf{inp}, C \in \mathcal{C}_{\mathsf{inp}(\lambda)}$ and $x \in \{0,1\}^{\mathsf{inp}(\lambda)}$, we have

$$\begin{aligned} &\left\{ \{\mathsf{lab}_i\}_{i\in[\mathsf{inp}]} \ : \ \{\mathsf{lab}_i\}_{i\in[\mathsf{inp}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\mathsf{inp}}, 1^{|C|}, C(x)\right) \right\} \\ \overset{c}{\approx} \ &\left\{ \{\mathsf{lab}_{i,x_i}\}_{i\in[\mathsf{inp}]} \ : \ \{\mathsf{lab}_{i,b}\}_{i\in[\mathsf{inp}],b\in\{0,1\}} \leftarrow \mathsf{GC.Garble}\left(C, 1^\lambda\right) \right\} \end{aligned} \tag{A.1}$$

Garbled circuits are known to exist from one-way functions [Yao82, BHR12b]. Though the above definition is not as general as one in [BHR12b], it suffices for our constructions in this paper.

Remark A.3. Note that in the standard syntax of garbling scheme [LP09, BHR12a], the output of the garbling algorithm consists of labels along with a garbled circuit. On the other hand, the output only consists of labels in our syntax. However, the former can easily be converted into the latter by including the garbled circuit into a label for example. This change in syntax is made for simplifying our constructions and notations. We also note that the same primitive is called decomposable randomized encoding in [GVW12].

Remark A.4 (Multi-instance security definition). The above definition allows to argue security of a single instance of GC. For the security proof of our constructions in Section 3, it is convenient to consider multi-instance version of the above notion. In the multi-instance variant, the adversary can adaptively make polynomial number of garbling queries to the challenger. All queries are answered by honestly generated labels in the real world whereas they are all simulated in the ideal world. If both worlds are computationally indistinguishable, we say that GC satisfies multi-instance security. Note that such *multi-instance* security follows from the standard single instance security (from Definition A.2 above) by a set of simple hybrid arguments.

A.3 Identity-based Encryption

We define identity-based encryption (IBE) [KT18, BF01, Coc01] in this section. An IBE scheme IBE with identity space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$ consists of four algorithms as follows.

- $\mathsf{Setup}(1^\lambda)$ is a PPT algorithm that takes as input the unary representation of security parameter and outputs the master public and secret keys mpk and msk.
- $\mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ is a PPT algorithm that takes as input the master secret key msk and an identity $\mathsf{id} \in \mathcal{I}_\lambda$. It outputs a corresponding secret key $\mathsf{sk}_{\mathsf{id}}$.
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)$ is a PPT algorithm that takes as input the master public parameter mpk, an identity $\mathsf{id} \in \mathcal{I}_\lambda$ and a message $m \in \{0,1\}^*$. It outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct})$ is a deterministic polynomial time algorithm that takes as input the secret key $\mathsf{sk}_{\mathsf{id}}$ and a ciphertext ct, and outputs either a message m' or $\bot$.

Definition A.5 (Correctness). For correctness, we require that for all $\lambda \in \mathbb{N}, m \in \{0,1\}^*, \mathsf{id} \in \mathcal{I}_\lambda$,

$$\Pr\left[\begin{array}{c}\mathsf{Dec}(\mathsf{sk}_{\mathsf{id}}, \mathsf{ct}_{\mathsf{id}}) = m \mid (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^L),\\ \mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id}), \mathsf{ct}_{\mathsf{id}} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m)\end{array}\right] = 1,$$

where the probability is taken over the random coins of Setup, KeyGen and Enc.

Remark A.6 (About the identity space.). In this paper, we set the identity space $\mathcal{I}$ to be $\{0,1\}^*$ by default. This is without loss of generality assuming collision resistant hash function and an IBE scheme with sufficiently large identity space (e.g., $\mathcal{I}_\lambda = \{0,1\}^{2\lambda}$), since we can hash any string into a string of fixed length using collision resistant hash function.

Remark A.7 (About the message length.). In the above definition, we assume that the message space is $\{0,1\}^*$. This is without loss of generality if we consider IND-CPA security for IBE, since it is straightforward to extend the message space by encrypting each bit (or chunk) of the message in parallel. However, when we consider SIM-RSO security, the above does not work. When we consider such a scheme, we augment the above syntax so that the setup algorithm takes an additional input 1^L that specifies the message space $\{0,1\}^L$. The efficiency of the algorithms then should depend on L polynomially. Our constructions in Section 3.2 requires IBE schemes with unbounded message space and IND-CPA security and our construction in Section 3.3 requires a scheme with bounded message space and SIM-RSO security.

We consider two different definitions of security in this paper. Namely, we consider IND-CPA security and SIM-RSO (*simulation based receiver selective opening*) security. Below, we describe them in order.

Definition A.8 (IND-CPA **Security).** An IBE scheme IBE for an identity space $\mathcal{I}_\lambda$ and message space $\{0,1\}^*$ is said to satisfy indistinguishability based security if for any stateful PPT adversary A, there exists a negligible function $\mathrm{negl}(.)$ such that

$$\mathsf{Adv}_{\mathsf{IBE},\mathsf{A}}(1^\lambda) = \left|\Pr[\mathsf{Exp}^{(0)}_{\mathsf{IBE},\mathsf{A}}(1^\lambda) = 1] - \Pr[\mathsf{Exp}^{(1)}_{\mathsf{IBE},\mathsf{A}}(1^\lambda) = 1]\right| \leq \mathrm{negl}(\lambda),$$

where for each $b \in \{0,1\}$ and $\lambda \in \mathbb{N}$, the experiment $\mathsf{Exp}^{(b)}_{\mathsf{IBE},\mathsf{A}}$, modelled as a game between adversary and challenger, is defined as follows:

1. **Setup Phase:** On input 1^λ from the adversary A, the challenger samples $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and replies to A with mpk.
2. **Query phase:** During the game, A adaptively makes the following queries in any arbitrary order and unbounded many times.
   (a) **Key Queries:** A chooses an identity $\mathsf{id} \in \mathcal{I}_\lambda$ and sends it to the challenger. For each such query, the challenger replies with $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$.
   (b) **Encryption Queries:** A submits to the challenger, an identity $\mathsf{id}^* \in \mathcal{I}_\lambda$ and a pair of equal length messages (m_0, m_1). The challenger replies with $\mathsf{ct}_{\mathsf{id}^*} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{id}^*, m_b)$.
3. **Output phase:** A outputs it's guess b' as output of the experiment.

We say an adversary A is legitimate if during the challenge phase, it is restricted to query for ciphertexts corresponding to identities $\mathsf{id}^* \in \mathcal{I}_\lambda$ whose secret keys are not queried during any key query.

Remark A.9. Since we are in the public key setting, we can simplify the above experiment so that the encryption query is allowed for only once. This simplified definition is shown to equivalent to the definition above by a simple hybrid argument. We adopt the above multi-challenge security definition since it is convenient for our purpose in this paper.

Definition A.10 (SIM-RSO **Security).** Consider an IBE scheme IBE for an identity space $\mathcal{I}_\lambda$. For every stateful PPT adversary A and a stateful PPT simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$, consider the following experiments:

$\underline{\mathsf{Exp}^{\mathsf{real}}_{\mathsf{IBE},\mathsf{A}}(1^\lambda)}$**:**

1: $1^L \leftarrow \mathsf{A}(1^\lambda)$
2: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$
3: $(T, \mathsf{Msg}) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$, where
   - $T \subset \mathcal{I}_\lambda \setminus \mathsf{Ext}$, where $\mathsf{Ext} = \{\mathsf{id}_q\}_{q\in[Q_1]}$ is the set of identities for which A made a query to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$.
   - $\mathsf{Msg} := \{m_{\mathsf{id}} \in \{0,1\}^L\}_{\mathsf{id}\in T}$.
4: $\mathsf{CT} := \{\mathsf{Enc}(\mathsf{mpk}, \mathsf{id}, m_{\mathsf{id}})\}_{\mathsf{id}\in T}$, where $\mathsf{CT} := \{\mathsf{ct}_{\mathsf{id}}\}_{\mathsf{id}\in T}$.
5: $b \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}(\mathsf{msk},\cdot)}(\mathsf{CT})$, where
   - A cannot query $\mathsf{id} \in T$ to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$.
   - A cannot query $\mathsf{id} \notin T$ to $\mathcal{O}(\mathsf{msk}, \cdot)$.
6: Output b.

$\underline{\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{IBE},\mathsf{Sim}}(1^\lambda)}$**:**

1: $1^L \leftarrow \mathsf{A}(1^\lambda)$
2: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$
3: $(T, \mathsf{Msg}) \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$, where
   - $T \subset \mathcal{I}_\lambda \setminus \mathsf{Ext}$, where $\mathsf{Ext} = \{\mathsf{id}_q\}_{q\in[Q_1]}$ is the set of identities for which A made a query to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$.
   - $\mathsf{Msg} := \{m_{\mathsf{id}} \in \{0,1\}^L\}_{\mathsf{id}\in T}$.
4: $(\mathsf{CT}, \mathsf{st}) \leftarrow \mathsf{SimEnc}(\mathsf{mpk}, T, 1^L)$, where $\mathsf{CT} := \{\mathsf{ct}_{\mathsf{id}}\}_{\mathsf{id}\in T}$.
5: $b \leftarrow \mathsf{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot),\mathcal{O}'(\mathsf{msk},\mathsf{st},\cdot)}(\mathsf{CT})$, where
   - A cannot query $\mathsf{id} \in T$ to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$.
   - A cannot query $\mathsf{id} \notin T$ to $\mathcal{O}'(\mathsf{msk}, \mathsf{st}, \cdot)$.
6: Output b.

We note that though the adversary A is stateful, we do not explicitly include its internal state (which may include the list $\mathsf{Ext}, T, \mathsf{Msg}$) into the output above to keep the notation simple. On the other hand, in the ideal experiment, st explicitly denotes the internal state of the simulator Sim. The oracles from both the experiments above are defined below. They are called arbitrarily many times, but with the restrictions on the identities that can be queried.

- The oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ can be invoked for all $\mathsf{id} \in \mathcal{I}_\lambda$ before A specifies its target T. After that, it can only be invoked for $\mathsf{id} \notin T$. Each time it is invoked on some $\mathsf{id} \in \mathcal{I}_\lambda$, it runs $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ and returns $\mathsf{sk}_{\mathsf{id}}$ to A.
- The oracle $\mathcal{O}(\mathsf{msk}, \cdot) = \mathsf{KeyGen}(\mathsf{msk}, \cdot)$ can only be invoked for an identity id in T. In particular, each time it is invoked on some $\mathsf{id} \in \mathcal{I}_\lambda$, it runs $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, \mathsf{id})$ and returns $\mathsf{sk}_{\mathsf{id}}$ to A.
- The oracle $\mathcal{O}'(\mathsf{msk}, \mathsf{st}, \cdot)$ can only be invoked for $\mathsf{id} \in T$. When it is invoked for $\mathsf{id} \in T$, it computes
$$\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{SimKG}(\mathsf{st}, \mathsf{msk}, \mathsf{id}, m_{\mathsf{id}})$$
and returns $\mathsf{sk}_{\mathsf{id}}$ to A.

Note that A can choose arbitrary size T in Step 3 in the experiments. The IBE scheme IBE is then said to satisfy simulation based receiver selective opening security against adaptive adversaries ($\mathsf{SIM\text{-}RSO}$-secure, for short) if there is a PPT simulator Sim such that for every PPT adversary A, the following holds:

$$\left|\Pr[\mathsf{Exp}^{\mathsf{real}}_{\mathsf{IBE},\mathsf{A}}\left(1^\lambda\right) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{IBE},\mathsf{Sim}}\left(1^\lambda\right) = 1]\right| = \mathrm{negl}(\lambda). \tag{A.2}$$

We call a query to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ a secret key query, and a query to $\mathcal{O}(\mathsf{msk}, \cdot)$ and $\mathcal{O}'(\mathsf{msk}, \mathsf{st}, \cdot)$ a corruption query.

Remark A.11. We would like to highlight a few points in Definition A.10 above.

1. First of all, the definition here is adapted from [KT18], but according to the notation and formulation that we require for our construction in Section 3.3.

2. In [KT18], the adversary chooses a finite distribution on the message space in the challenge phase. In contrast, here the adversary chooses a *constant* message distribution, namely a *fixed list* of messages $\mathsf{Msg} = \{m_{\mathsf{id}_1}, m_{\mathsf{id}_2}, \ldots, m_{\mathsf{id}_{|T|}}\}$, where $T = \{\mathsf{id}_1, \ldots, \mathsf{id}_{|T|}\}$ and $|T|$ is some arbitrary polynomial in λ. Thus our definition corresponds to the constant distribution in their setting and hence is implied by their definition.

3. In [KT18], they do not explicitly require the existence of simulator $(\mathsf{SimEnc}, \mathsf{SimKG})$ in their security definition. However, they implicitly construct such algorithms that satisfy the above properties in their security proof.

4. Our definition models *adaptive* (i.e., *multi-shot*) corruptions by the adversary in Step 5, this being a requirement for our construction. The definition in [KT18] considers the slightly simplified non-adaptive (i.e., single-shot) corruption setting, though they note that their $\mathsf{SIM\text{-}RSO}$ secure IBE scheme satisfies adaptive corruptions as well.

5. Finally, the $\mathsf{SIM\text{-}RSO}$ secure IBE scheme in [KT18] encrypts single-bit messages, whereas our construction in Section 3.3 requires to support multi-bit messages. However, we note that the

message space in their construction can be extended simply by encrypting a message $m = (m_1, \ldots, m_L) \in \{0,1\}^L$ as

$$\mathsf{Enc}(\mathsf{mpk}, (\mathsf{id}, 1), m_1), \ldots, \mathsf{Enc}(\mathsf{mpk}, (\mathsf{id}, L), m_L),$$

with (id, i) set as the extended identity for encrypting each bit $i \in [L]$. Therefore, we may use the scheme from [KT18] to instantiate the IBE for our construction.

B Missing Details from Section 3

B.1 Reusable, Dynamic MPC Protocol

We described the RDMPC protocol informally in Section 3.1. Here, we provide its formal definition adapted from [AV19]. In particular, we consider the following algorithms. Recall that $C \in \mathcal{C}_{\mathsf{inp}}$ is the circuit class consisting of circuits with input length inp. The protocol is further associated with polynomial functions $N = N(\lambda, R)$, $n = n(\lambda, R)$, and $t = t(\lambda, R)$.

$\mathsf{CktEnc}(1^\lambda, 1^R, 1^{\mathsf{inp}}, C) \to (\widehat{C}_1, \ldots, \widehat{C}_N)$: The circuit encoding algorithm takes as input the security parameter λ, the number of sessions R, input length of circuit inp, and a circuit $C \in \mathcal{C}_{\mathsf{inp}}$. It then outputs an encoding $(\widehat{C}_1, \ldots, \widehat{C}_N)$ of the circuit C.

$\mathsf{InpEnc}(1^\lambda, 1^R, 1^{\mathsf{inp}}, x) \to \widehat{x}$: The input encoding algorithm takes as input the security parameter λ, the number of sessions R, input length of circuit inp, and an input $x \in \{0,1\}^{\mathsf{inp}}$. It then outputs an encoding $\widehat{x}$ of the input x.

$\mathsf{Local}(\widehat{C}_u, \widehat{x}) \to \widehat{y}_u$: The local computation algorithm takes as input the u-th encoding $\widehat{C}_u$ of C and an encoding $\widehat{x}$ of x and outputs $\widehat{y}_u$. We assume that this algorithm is deterministic.

$\mathsf{Decode}(\{\widehat{y}_u\}_{u \in S}, S) \to z$: The decoding algorithm takes as input a set of encodings $\{\widehat{y}_u\}_{u \in S}$ and a set $S \subseteq [N]$ and outputs z.

Remark B.1 (Comparison with the original definition [AV19]). The protocol RDMPC we consider here is syntactically different from the one in [AV19]. Firstly, we exchange the roles of circuits and inputs. In particular, we are concerned about hiding the circuit and not the input, while they do the opposite. This difference stems from the fact that we consider the construction of CPFE which is the dual notion of KPFE they consider. Secondly, our input encoding algorithm outputs single encoding $\widehat{x}$, whereas their circuit encoding algorithm outputs multiple encodings $(\widehat{C}_1, \ldots, \widehat{C}_N)$.[6] However, their construction satisfies $\widehat{C}_1 = \cdots = \widehat{C}_N$ and thus our slightly simplified syntax can still capture this. Finally, they input the size of the circuits to CktEnc and InpEnc (in unary form), whereas we input the input length of the circuits to the algorithms. This means that our syntax allows the encoding algorithm to encode the input without knowing the size of the circuit. Looking ahead, this property will be useful for constructing CPFE that supports circuits with *unbounded size*. We note that their construction can be adapted to our setting naturally.

Definition B.2 (Correctness). An RDMPC protocol $\mathsf{RDMPC} = (\mathsf{CktEnc}, \mathsf{InpEnc}, \mathsf{Local}, \mathsf{Decode})$ with parameter (N, n, t) is correct if for all $\mathsf{inp} \in \mathbb{N}$, $x \in \{0,1\}^{\mathsf{inp}}$, $C \in \mathcal{C}_{\mathsf{inp}}$, and set $S \subset [N]$ of size n, we have

$$\Pr\left[\begin{array}{c} (\widehat{C}_1, \ldots, \widehat{C}_N) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^R, 1^{\mathsf{inp}}, C), \\ \widehat{x} \leftarrow \mathsf{InpEnc}(1^\lambda, 1^R, 1^{\mathsf{inp}}, x), \\ \mathsf{Decode}\left(\left\{\mathsf{Local}(\widehat{C}_u, \widehat{x})\right\}_{u \in S}, S\right) = C(x) \end{array}\right] = 1$$

[6] Recall that we swap the roles of circuits and inputs. Thus, we compare our input encoding algorithm with their circuit encoding algorithm.

where probability is taken over the random coins of CktEnc, InpEnc and Decode (Recall that Local is assumed to be deterministic.).

Definition B.3 (Security)**.** Let $\mathsf{RDMPC} = (\mathsf{CktEnc}, \mathsf{InpEnc}, \mathsf{Local}, \mathsf{Decode})$ be an RDMPC protocol for the circuit family $\mathcal{C}_{\mathsf{inp}}$ with parameter (N, n, t). For a stateful PPT adversary A, a simulator $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$, and a coin $\beta \in \{0, 1\}$ consider the following experiment $\mathsf{Exp}^{(\beta)}_{\mathsf{RDMPC},\mathsf{A}}$:

1. **Setup phase:** On input 1^λ, A submits the query bound 1^R and input length 1^{inp} of the circuits to the challenger. Note that this defines the total number of parties $N = N(\lambda, R)$, number of parties $n = n(\lambda, R)$ participating in any session, threshold $t = t(\lambda, R)$. The adversary A also chooses $S_{\mathsf{crr}} \subset [N]$ of size at most t and sets $\Delta^{(1)}, \ldots, \Delta^{(R)} \subseteq [N]$ such that $|\Delta^{(i)}| = n$ and submits it to the challenger.

2. **Query phase:** During the game, A is allowed to make a total of R input encoding queries. First, it makes $R_1 \leq R$ adaptive input encoding queries. Namely, when A makes the q-th input encoding query $\widehat{x}^{(q)}$ with $q \leq R$, the challenger runs $\widehat{x}^{(q)} \leftarrow \mathsf{InpEnc}(x^{(q)})$ and returns $\widehat{x}^{(q)}$ to A.

3. **Challenge phase:** During the game, A is allowed to make single circuit encoding query. When A submits a circuit $C \in \mathcal{C}_{\mathsf{inp}}$, the challenger proceeds as follows.

   - **Real World.** If $\beta = 0$, the challenger runs $(\widehat{C}_1, \ldots, \widehat{C}_N) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^R, 1^{\mathsf{inp}}, C)$ and returns $\left(\left\{\widehat{C}_j\right\}_{j \in S_{\mathsf{crr}}}, \left\{\mathsf{Local}(\widehat{C}_j, \widehat{x}^{(q)})\right\}_{q \in [R_1], j \in \Delta^{(q)}}\right)$ to A .
   - **Ideal World.** If $\beta = 1$, we define $\mathcal{V}$ as $\mathcal{V} = \{C(x^{(q)}), x^{(q)}\}_{q \in [R_1]}$. Then, the simulator is run as $\left(\mathsf{st}, \left\{\widehat{C}_j\right\}_{j \in S_{\mathsf{crr}}}, \left\{\widehat{y}_j^{(q)}\right\}_{q \in [R_1], j \in \Delta^{(q)}}\right) \leftarrow \mathsf{Sim}_0(1^{|C|}, S_{\mathsf{crr}}, \mathcal{V})$ and the output is returned to A. Here, st is the internal state of the simulator.

4. **Query phase:** A then makes $R_2 \leq R - R_1$ input encoding queries. When A submits an input $x^{(q)} \in \{0, 1\}^{\mathsf{inp}}$, the challenger proceeds as follows.

   - **Real World.** If $\beta = 0$, the challenger runs $\widehat{x}^{(q)} \leftarrow \mathsf{InpEnc}(x^{(q)})$ and returns $\left(\widehat{x}^{(q)}, \left\{\widehat{C}_j(\widehat{x}^{(q)})\right\}_{j \in \Delta^{(q)}}\right)$ to A.
   - **Ideal World.** If $\beta = 1$, we run the simulator as $\left(\widehat{x}^{(q)}, \left\{\widehat{y}_j^{(q)}\right\}_{j \in \Delta^{(q)}}\right) \leftarrow \mathsf{Sim}_1(\mathsf{st}, \Delta^{(q)}, C(x^{(q)}), x^{(q)})$ and returns the output to A.

5. **Output phase:** A outputs a guess bit β' as the output of the experiment.

We say that an RDMPC protocol RDMPC is secure if for every adversary A, there exists a PPT simulator Sim such that

$$\mathsf{Adv}_{\mathsf{RDMPC},\mathsf{A}}(1^\lambda) = \left|\Pr\left[\mathsf{Exp}^{(0)}_{\mathsf{RDMPC},\mathsf{A}} = 1\right] - \Pr\left[\mathsf{Exp}^{(1)}_{\mathsf{RDMPC},\mathsf{A}} = 1\right]\right| \leq \mathrm{negl}(\lambda).$$

Remark B.4 (Comparison with the original definition). The above definition is slightly stronger than the original definition in that the input encoding queries before the challenge phase are honestly answered regardless of $\beta = 0$ or $\beta = 1$. Original definition by Ananth and Vaikuntanathan [AV19] allows the simulator to simulate the encodings. Another difference is that we do not allow Sim_1 to update its internal state st while they do in their definition. Since their construction of RDMPC in fact satisfies this stronger and simpler version of the definition, we use the above version of the definition.

Remark B.5 (Multi-instance definition). The above definition allows to argue security of a single instance of RDMPC. For the security proof of our constructions in Section 3, it is convenient to consider multi-instance version of the above notion. In the multi-instance security variant, the adversary declares the number of instances M at the beginning of the game and interact with each instance as above. In the real world, the adversary interacts with real algorithms in all instances, while in the ideal world, it interacts with simulators in all instances. The adversary can make queries in arbitrary order and make them arbitrarily correlated as long as it respects the restriction for each instance. This multi-instance security notion is easily shown to be equivalent to the single-instance security notion above by simple hybrid argument.

B.2 Correctness proof of BCPFE

Theorem B.6. BCPFE *defined in Section 3.2 is correct.*

Proof. We observe that $\mathsf{lab}'_{u,j,k} = \mathsf{lab}_{u,j,k,\widehat{x}_k}$ holds for all $\mathsf{lab}'_{u,j,k}$ recovered in Step 2a of the decryption algorithm by the correctness of IBE, since the ciphertext and secret key are both generated with respect to the identity $(u, j, k, \widehat{x}_k)$. Then, by the correctness of GC, we have $\widehat{y}'_{u,j} = \mathsf{L}_{u,j}(\widehat{x}) = \mathsf{Local}(\widehat{C}_{u,j}, \widehat{x})$ for all $\widehat{y}'_{u,j}$ recovered in Step 2b of the decryption algorithm. Finally, by the correctness of RDMPC, we have $z = C(x)$ for z recovered in Step 3 of the decryption algorithm, since we have $|\Delta| = n$. □

Efficiency of Basic CPFE scheme BCPFE**.** We observe that Setup and KeyGen run in time $\mathrm{poly}(\lambda, \ell, \log Q)$ by the efficiency properties of IBE and RDMPC. This means that these algorithms can be run even for super polynomial Q. Looking ahead, this property will be used crucially for the full-fledged construction in Section 3.4. We also observe that Enc and Dec run in time $Q \cdot \mathrm{poly}(\lambda, \ell, |C|)$ by the efficiency properties of IBE, GC, and RDMPC.

B.3 Proof of Theorem 3.3

Theorem B.7 (Restate of Theorem 3.3)**.** *Assume that* IBE *satisfies* IND-CPA *security,* GC *is a secure garbled circuit scheme, and* RDMPC *is secure. Then,* BCPFE *for the circuit class $\mathcal{C}_\ell$ is* NA-SIM*-secure.*

Proof. To prove the security, we construct the simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \perp)$ for NA-SIM security of BCPFE. The ideal world with simulator Sim for BCPFE and adversary A proceeds as follows.

Ideal World. On input 1^λ, A outputs $\mathsf{prm} = 1^\ell$ and 1^Q, upon which the game runs $\mathsf{Setup}(1^\lambda, \mathsf{prm}, Q)$, gets $\mathsf{mpk} = \mathsf{IBE.mpk}$, and sends it to A. For each $q \in [Q_1]$, A queries secret key for $x^{(q)}$ to receive $\mathsf{sk}^{(q)}$. A then outputs the encryption query C.

Simulating the ciphertext. Here, the simulator SimEnc is given

$$\mathsf{mpk} = \mathsf{IBE.mpk}, \quad \mathcal{V}, \quad 1^{|C|}, \quad 1^Q$$

where

$$\mathcal{V} := \left\{ C(x^{(q)}),\ x^{(q)},\ \mathsf{sk}^{(q)} = \left(u^{(q)}, \Delta^{(q)}, \left\{ \mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k} \right\}_{j \in \Delta^{(q)}, k \in [\widehat{\ell}]} \right) \right\}_{q \in [Q_1]}$$

and simulates the ciphertext for the adversary. Here, $x^{(q)}$ and $\mathsf{sk}^{(q)}$ are the q-th key query made by the adversary and the secret key returned to it, respectively. Furthermore, C is the circuit chosen by the adversary for the encryption. Before giving the description of the simulator, we prepare some notations. We define $\Gamma_i \subseteq [Q_1]$ as

$$\Gamma_i := \{q \in [Q_1] : u^{(q)} = i\}.$$

We also define $S_{\mathsf{crr},i} \subseteq [N]$ and $S_{\mathsf{qrd},i} \subseteq [N]$ for $i \in [Q]$ as

$$S_{\mathsf{crr},i} := \bigcup_{q,q' \in \Gamma_i, q \neq q'} \left(\Delta^{(q)} \cap \Delta^{(q')} \right), \quad S_{\mathsf{qrd},i} := \left(\bigcup_{q \in \Gamma_i} \Delta^{(q)} \right) \setminus S_{\mathsf{crr},i}$$

Recall that our construction can be seen as QN parallel instances of single key FE scheme. Each single-key FE scheme is associated with index (i, j), where $i \in [Q]$ denotes the group index and $j \in [N]$ denotes index of the instance in the group. Intuitively, Γ_i corresponds to the set of queries made for the i-th group of the system. Furthermore, $S_{\mathsf{crr},i}$ corresponds to the set of instances j in group i such that the secret key for the instance (i, j) is issued at least twice. For such instance, we cannot hope to use the security of the single-key FE (or equivalently, security of the garbled circuit). On the other hand, $S_{\mathsf{qrd},i}$ corresponds to the set of instances j in group i such that the secret key for the instance (i, j) is issued exactly once. For such an instance, we can use the security of the single-key FE.

We then provide the description of the simulator. The simulator proceeds as follows.

1. If there exists $i \in [Q]$ such that $|\Gamma_i| > \lambda$ or $|S_{\mathsf{crr},i}| > t$, abort.
2. For each of $i \in [Q]$, run

$$\left(\left\{ \widehat{C}_{i,j} \right\}_{j \in S_{\mathsf{crr},i}}, \left\{ \widehat{y}_{i,j}^{(q)} \right\}_{q \in \Gamma_i, j \in \Delta^{(q)}}, \mathsf{st}_i \right) \leftarrow \mathsf{RDMPC.Sim}_0 \left(1^{|C|}, S_{\mathsf{crr},i}, \{ C(x^{(q)}), x^{(q)} \}_{q \in [\Gamma_i]} \right).$$

3. For all $i \in [Q]$ and $j \in [N]$, do the following. There are three cases to consider:
   - If $j \in S_{\mathsf{crr},i}$, set $\mathsf{L}_{i,j}(\cdot) := \mathsf{Local}(\widehat{C}_{i,j}, \cdot)$ and run

$$\{ \mathsf{lab}_{i,j,k,b} \}_{k \in [\widehat{\ell}], b \in \{0,1\}} \leftarrow \mathsf{GC.Garble} \left(1^{\lambda}, \mathsf{L}_{i,j} \right).$$

     Then, for all $k \in [\widehat{\ell}]$ and $b \in \{0, 1\}$ compute

$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc} \left(\mathsf{IBE.mpk}, (i, j, k, b), \mathsf{lab}_{i,j,k,b} \right).$$

   - If $j \in S_{\mathsf{qrd},i}$, retrieve $q \in \Gamma_i$ such that $j \in \Delta^{(q)}$. By the definition of $S_{\mathsf{qrd},i}$, there exists unique such q. Then run

$$\{ \mathsf{lab}_{i,j,k} \}_{k \in [\widehat{\ell}]} \leftarrow \mathsf{GC.Sim} \left(1^{\lambda}, 1^{\widehat{\ell}}, 1^{\widehat{s}(\lambda, |C|)}, \widehat{y}_{i,j}^{(q)} \right),$$

     where $\widehat{s}(\lambda, |C|)$ is the size of the circuit $\mathsf{L}_{i,j}$. Then, for all $k \in [\widehat{\ell}]$ and $b \in \{0, 1\}$ compute

$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc} \left(\mathsf{IBE.mpk}, (i, j, k, b), \mathsf{lab}_{i,j,k} \right).$$

     Note that here, we encrypt the same label $\mathsf{lab}_{i,j,k}$ for both $b \in \{0, 1\}$.
   - If $j \notin S_{\mathsf{crr},i} \cup S_{\mathsf{qrd},i}$, for all $k \in [\widehat{\ell}]$ and $b \in \{0, 1\}$ compute

$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc} \left(\mathsf{IBE.mpk}, (i, j, k, b), 0^{L(\lambda, |C|)} \right),$$

     where $L(\lambda, |C|)$ is the length of the labels.
4. Output $\mathsf{ct} = \{ \mathsf{IBE.ct}_{i,j,k,b} \}_{i \in [Q], j \in [N], k \in [\widehat{\ell}], b \in \{0,1\}}$ as the ciphertext.

The Games. To prove the security, we consider the following sequence of games, where $\mathbf{Game}_0$ corresponds to the real game and $\mathbf{Game}_5$ corresponds to the ideal world. For i, let $\mathcal{E}_i$ denote the event that A outputs 1 in $\mathbf{Game}_i$. The theorem can be proven by showing $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_5]| \leq \mathrm{negl}(\lambda)$.

$\mathbf{Game}_0$: This is the real game $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{BCPFE},\mathsf{A}}\left(1^\lambda\right)$ for NA-SIM security defined as per Definition 2.6.

$\mathbf{Game}_1$: In this game, the game checks whether there exists $i \in [Q]$ such that $|\Gamma_i| > \lambda$ or $|S_{\mathsf{crr},i}| > t$ before generating a ciphertext. If there is, it aborts and otherwise continues the game. By Lemma 3.4, we have $|\Gamma_i| \leq \lambda$ for all $i \in [Q]$ with overwhelming probability. Conditioned on this, by applying Lemma 3.5, we have $|S_{\mathsf{crr},i}| \leq t$ for all $i \in [Q]$ with overwhelming probability. Therefore, the probability of the above occurring is negligible. Thus, we have

$$|\Pr[\mathcal{E}_0] = \Pr[\mathcal{E}_1]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_2$: In this game, we change the way how the ciphertext is generated. In particular, we change how $\mathsf{IBE.ct}_{i,j,k,b}$ is generated in the following cases.

- If $j \in S_{\mathsf{qrd},i}$, the game computes

$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,\widehat{x}^{(q)}_k}\right),$$

  where q is the unique index such that $u^{(q)} = i$ and $j \in \Delta^{(q)}$. Such an index exists and is unique by the definition of $S_{\mathsf{qrd},i}$. Note that here, we encrypt the same label $\mathsf{lab}_{i,j,k,\widehat{x}^{(k)}}$ regardless of the value of $b \in \{0,1\}$.
- If $j \notin S_{\mathsf{crr},i} \cup S_{\mathsf{qrd},i}$, run

$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), 0^{L(\lambda,|C|)}\right).$$

As we will show in Lemma B.8, using IND-CPA security of IBE, we can prove

$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_3$: In this game, we further change the way the ciphertext is generated. In particular, we change the way $\mathsf{lab}_{i,j,k,b}$ is generated in the following cases.

- If $j \in S_{\mathsf{qrd},i}$, the game retrieves unique q such that $u^{(q)} = i$ and $j \in \Delta^{(q)}$. Then it runs

$$\{\mathsf{lab}_{i,j,k}\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}(\lambda,|C|)}, \widehat{y}^{(q)}_{i,j}\right),$$

  where $\widehat{y}^{(q)}_{i,j} = \mathsf{Local}(\widehat{C}_{i,j}, \widehat{x}^{(q)})$. Then the game sets

$$\mathsf{lab}_{i,j,k,\widehat{x}^{(q)}_k} := \mathsf{lab}_{i,j,k}$$

  for $k \in [\widehat{\ell}]$. The label will be used when it computes $\mathsf{IBE.ct}_{i,j,k,b}$ for $b \in \{0,1\}$ as specified in the previous game.

As we will show in Lemma B.9, using the security of GC, we can prove

$$|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_4$: In this game, we further change the way the ciphertext is generated. In particular, we change the way $\widehat{C}_{i,j}$ and $\widehat{y}^{(q)}_{i,j}$ are generated as follows. For each of $i \in [Q]$, the game runs

$$\left(\left\{\widehat{C}_{i,j}\right\}_{j\in S_{\mathsf{crr},i}}, \left\{\widehat{y}^{(q)}_{i,j}\right\}_{q\in\Gamma_i, j\in\Delta^{(q)}}, \mathsf{st}_i\right) \leftarrow \mathsf{RDMPC.Sim}_0\left(1^{|C|}, S_{\mathsf{crr},i}, \{C(x^{(q)}), x^{(q)}\}_{q\in[\Gamma_i]}\right).$$

We only have $\widehat{C}_{i,j}$ and $\widehat{y}_{i,j}^{(q)}$ for the indices i, j, q that are defined by the output of the above algorithm and leave the values for other indices undefined. Even if it is so, the game is still well-defined since only $\widehat{C}_{i,j}$ for i, j such that $j \in S_{\mathsf{crr},i}$ and $\widehat{y}_{i,j}^{(q)}$ for i, j, q such that $q \in \Gamma_i$, $j \in \Delta^{(q)}$ are needed for the generation of the ciphertext thanks to the changes introduced in $\mathbf{Game}_2$ and $\mathbf{Game}_3$. As we will show in Lemma B.10, we have

$$|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_5$: This is the ideal game $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{BCPFE},\mathsf{Sim}}(1^\lambda)$ for NA-SIM security defined as per Definition 2.6. By inspection, it can be seen that this is the same as the previous game. Therefore, we have

$$\Pr[\mathcal{E}_4] = \Pr[\mathcal{E}_5].$$

To finish the proof of the theorem, it suffices to prove lemmas B.8 to B.10, since we have $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_5]| \leq \sum_{i\in[0,4]} |\Pr[\mathcal{E}_i] - \Pr[\mathcal{E}_{i+1}]|$ by the triangle inequality. □

Lemma B.8. *We have* $|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda)$ *by the* IND-CPA *security of* IBE.

Proof. Assume $|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]|$ is non-negligle. We then describe an adversary B that breaks multi-challenge IND-CPA security of IBE as follows. Note that as we remarked in Theorem A.9, the multi-challenge security notion is equivalent to more standard single-challenge security notion. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = 1^\ell$. Then, B is given IBE.mpk from its challenger and gives $\mathsf{mpk} = \mathsf{IBE.mpk}$ to A as master public key.

Answering the secret key queries before the encryption query. When A makes the q-th secret key for $x^{(q)} \in \{0,1\}^\ell$, B does the following.

1. It chooses $\widehat{x}^{(q)}$, $u^{(q)}$, and $\Delta^{(q)}$ as in the honest key generation algorithm.
2. For all $j \in \Delta^{(q)}$ and $k \in [\widehat{\ell}]$, it makes a secret key query for the identity $(u^{(q)}, j, k, \widehat{x}_k^{(q)})$ and receives $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$.
3. It returns $\mathsf{sk} = \left(u^{(q)}, \Delta^{(q)}, \left\{\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right\}_{j\in\Delta^{(q)},k\in[\widehat{\ell}]}\right)$ to A as a secret key.

Answering the encryption query. When A submits an encryption query C, B does the following:

1. It computes $\{\mathsf{lab}_{i,j,k,b}\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$ as honest encryption algorithm.
2. It computes $\{\mathsf{IBE.ct}_{i,j,k,b}\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$ as follows. There are two cases:
   - If $j \in S_{\mathsf{qrd},i}$ and $b = 1 - \widehat{x}_k^{(q)}$ where q is the index such that $u^{(q)} = i$ and $j \in S_{\mathsf{qrd},i}$, it proceeds as follows. Note that there exists unique q satisfying this by the definition of $S_{\mathsf{qrd},i}$. It submits the identity (i, j, k, b) and two messages $(\mathsf{lab}_{i,j,k,1-\widehat{x}_k^{(q)}}, \mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}})$ to the challenger. Then, the challenger encrypts

$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \begin{cases} \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,1-\widehat{x}_k^{(q)}}\right), & \text{if } \beta = 0 \\ \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}}\right) & \text{if } \beta = 1 \end{cases}$$

   and returns $\mathsf{IBE.ct}_{i,j,k,b}$ to B, where $\beta \in \{0,1\}$ is the coin for the game.

– If $j \notin S_{\mathsf{crr},i} \cup S_{\mathsf{qrd},i}$, it proceeds as follows. It submits the identity (i,j,k,b) and two messages $(\mathsf{lab}_{i,j,k,b}, 0^{L(\lambda,|C|)})$ to the challenger. Then, the challenger encrypts

$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \begin{cases} \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,b}\right), & \text{if } \beta = 0 \\ \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), 0^{L(\lambda,|C|)}\right) & \text{if } \beta = 1 \end{cases}$$

and returns $\mathsf{IBE.ct}_{i,j,k,b}$ to B.

– Otherwise, it computes $\mathsf{IBE.ct}_{i,j,k,b}$ as in the honest encryption algorithm.

3. It then returns the ciphertext $\mathsf{ct} = \left\{\mathsf{IBE.ct}_{i,j,k,b}\right\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_1$ for A if $\beta = 0$ and $\mathbf{Game}_2$ if $\beta = 1$.

Furthermore, B does not make a secret key query for an identity that is also submitted to the challenger for an encryption query. To see this, let us assume that B submits an identity (i,j,k,b) as a secret key query and (i',j',k',b') as an encryption query to the challenger. We then show that we cannot have $(i,j,k,b) = (i',j',k',b')$. To see this, observe that B only makes secret key queries for (i,j,k,b) such that $i = u^{(q)}$, $j \in \Delta^{(q)}$, $b = \widehat{x}_k^{(q)}$ for some $q \in [Q_1]$. On the other hand, for the identity (i',j',k',b') to be used for the encryption query, we need to have $j' \in S_{\mathsf{qrd},i'}$ or $j' \notin S_{\mathsf{qrd},i'} \cup S_{\mathsf{crr},i'}$. Assuming $(i',j') = (i,j)$, the latter cannot happen because $j' = j \in \Delta^{(q)}$. We therefore assume the former is the case. Since $j' \in S_{\mathsf{qrd},i'}$ and (i',j',k',b') is used for the encryption, there exists $q' \in [Q_1]$ satisfying $u^{(q')} = i'$, $j' \in \Delta^{(q')}$, and $b' = 1 - \widehat{x}_{k'}^{(q')}$. However, $j' \in S_{\mathsf{qrd},i'}$ implies that q' satisfying $u^{(q')} = i'$ and $j' \in \Delta^{(q')}$ is unique and thus $(i',j') = (i,j)$ would imply $q = q'$. Therefore, the two identities (i,j,k,b) and (i',j',k',b') cannot be equal, because $(i,j,k) = (i',j',k')$ implies

$$b = \widehat{x}_k^{(q)} = \widehat{x}_{k'}^{(q')} \neq 1 - \widehat{x}_{k'}^{(q')} = b'.$$

This means that B breaks the IND-CPA security of IBE without violating the query restriction of the game. □

Lemma B.9. *We have* $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda)$ *by the security of* GC.

Proof. Assume $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]|$ is non-negligle. We then describe an adversary B that breaks the multi-instance security of GC as follows. Note that since the multi-instance security of GC is implied by the single instance security as we observed in Remark A.4, this is a contradiction. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = 1^\ell$. B then runs $\mathsf{Setup}(1^\lambda, \mathsf{prm}, Q)$ to obtain mpk and msk and then sends mpk to A as the master public key.

Answering the secret key queries before the encryption query. B answers secret key queries using msk.

Answering the encryption query. When A submits an encryption query C, B proceeds as follows.

1. It computes $(\widehat{C}_{i,1}, \ldots, \widehat{C}_{i,N}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^\lambda, 1^\ell, C)$ for $i \in [Q]$.

2. For all $i \in [Q]$ and $j \in [N]$, B does the following. There are two cases to consider:

   – If $j \in S_{\mathsf{qrd},i}$, it retrieves unique $q \in \Gamma_i$ such that $j \in \Delta^{(q)}$. Then, it submits $\mathsf{L}_{i,j} = \mathsf{Local}(\widehat{C}_{i,j}, \cdot)$ and $\widehat{x}^{(q)}$ to its challenger. Then, the challenger runs

   $$\left\{\mathsf{lab}_{i,j,k,b}\right\}_{k\in[\widehat{\ell}],b\in\{0,1\}} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathsf{L}_{i,j}\right)$$

   or

   $$\left\{\mathsf{lab}_{i,j,k}\right\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}}, \mathsf{Local}(\widehat{C}_{i,j}, \widehat{x}^{(q)})\right)$$

depending on whether B is in the real world or simulated world. Then, B is given $\left\{\mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}}\right\}_{k\in[\widehat{\ell}]}$, where $\mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}} = \mathsf{lab}_{i,j,k}$ if it is in the ideal world.

– If $j \notin S_{\mathsf{qrd},i}$, it honestly garbles $\mathsf{L}_{i,j} = \mathsf{Local}(\widehat{C}_{i,j}, \cdot)$ to obtain $\{\mathsf{lab}_{i,j,k,b}\}_{k\in[\widehat{\ell}],b\in\{0,1\}}$.

3. For $i \in [Q]$ and $j \in [N]$, B does the following. There are three cases.

   – If $j \in S_{\mathsf{qrd},i}$, it computes $\mathsf{IBE.ct}_{i,j,k,b}$ for all $k \in [\widehat{\ell}]$, $b \in \{0,1\}$ as

   $$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}}\right),$$

   where the label $\mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}}$ is generated in Step 2 above.

   – If $j \in S_{\mathsf{crr},i}$, it computes $\mathsf{IBE.ct}_{i,j,k,b}$ for all $k \in [\widehat{\ell}]$, $b \in \{0,1\}$ as in the honest encryption algorithm.

   – If $j \notin S_{\mathsf{crr},i} \cup S_{\mathsf{qrd},i}$, it computes $\mathsf{IBE.ct}_{i,j,k,b}$ as

   $$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), 0^{L(\lambda,|C|)}\right).$$

4. It then returns the ciphertext $\mathsf{ct} = \{\mathsf{IBE.ct}_{i,j,k,b}\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_2$ for A if it is in the real world and $\mathbf{Game}_3$ if it is in the ideal world. Thus, B breaks the multi-instance security of GC if A distinguishes between the two games. □

Lemma B.10. *We have* $|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \leq \mathrm{negl}(\lambda)$ *by the security of* RDMPC.

Proof. Assume $|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]|$ is non-negligle. We then describe an adversary B that breaks the multi-instance security of RDMPC as follows. Note that since the multi-instance security of RDMPC is implied by the single instance security as we observed in Remark B.5, this is a contradiction.

Simulating the Setup. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = 1^\ell$. B then runs $\mathsf{Setup}(1^\lambda, \mathsf{prm})$ to obtain mpk and msk and then sends mpk to A as the master public key. Furthermore, B does the following.

1. It chooses $u^{(q)}$ and $\Delta^{(q)}$ for $q \in [Q]$ in advance. It then aborts and outputs random bit if there exists $i \in [Q]$ such that $|\Gamma_i| > \lambda$ or $|S_{\mathsf{crr},i}| > t$.

2. It then declares the number of instances $M = Q$, the query bound $R = \lambda$, the input length of circuits ℓ to its challenger. The security parameter λ and R specify the total number of parties N, number of parties n participating in any session, and threshold t. Furthermore, for $i \in [Q]$, B does the following.

   (a) It submits $S_{\mathsf{crr},i}$ to its challenger as the corruption set in the i-th instance.

   (b) It submits the sets $\{\Delta^{(q)}\}_{q\in\Gamma_i}$ to its challenger where $\Delta^{(q)}$ is the set of parties that will participate in the session in the i-th instance.

Answering the secret key queries. When A makes a secret key query $x^{(q)}$, B does the following.

1. It submits $x^{(q)}$ to the $u^{(q)}$-th instance. Then, the challenger runs

$$\widehat{x}^{(q)} \leftarrow \mathsf{InpEnc}(1^\lambda, 1^\lambda, 1^\ell, x^{(q)})$$

and returns $\widehat{x}^{(q)}$ to B.

2. It computes $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ for all $j \in \Delta^{(q)}$ and $k \in [\widehat{\ell}]$ using $\mathsf{IBE.msk}$.

3. It then returns $\mathsf{sk}^{(q)} = \left(u^{(q)}, \Delta^{(q)}, \left\{\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right\}_{j\in\Delta^{(q)},k\in[\widehat{\ell}]}\right)$ to A.

Answering the encryption query. When A submits an encryption query C, B proceeds as follows.

1. It declares the circuit C as its target for all instances of RDMPC.

2. Then, for each instance $i \in [Q]$, the challenger either computes

$$(\widehat{C}_{i,1}, \ldots, \widehat{C}_{i,N}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^\lambda, 1^\ell, C), \quad \widehat{y}_{i,j}^{(q)} := \mathsf{Local}(\widehat{C}_{i,j}, \widehat{x}^{(q)}) \quad \text{for } q \in \Gamma_i,\ j \in \Delta^{(q)}$$

or

$$\left(\left\{\widehat{C}_{i,j}\right\}_{j\in S_{\mathsf{crr},i}}, \left\{\widehat{y}_{i,j}^{(q)}\right\}_{q\in\Gamma_i, j\in\Delta^{(q)}}, \mathsf{RDMPC.st}_i\right) \leftarrow \mathsf{RDMPC.Sim}_0\left(1^{|C|}, S_{\mathsf{crr},i}, \{C(x^{(q)}), x^{(q)}\}_{q\in\Gamma_i}\right)$$

depending on whether B is in the real game or ideal game. Then, the challenger returns $\left(\left\{\widehat{C}_{i,j}\right\}_{j\in S_{\mathsf{crr},i}}, \left\{\widehat{y}_{i,j}^{(q)}\right\}_{q\in\Gamma_i, j\in\Delta^{(q)}}\right)$ to B for all $i \in [Q]$.

3. For all $i \in [Q]$ and $j \in [N]$, B does the following. There are three cases to consider.
   - If $j \in S_{\mathsf{crr},i}$, it sets $\mathsf{L}_{i,j}(\cdot) := \mathsf{Local}(\widehat{C}_{i,j}, \cdot)$ and honestly computes $\{\mathsf{IBE.ct}_{i,j,k,b}\}_{k\in[\widehat{\ell}],b\in\{0,1\}}$.
   - If $j \in S_{\mathsf{qrd},i}$, it retrieves unique $q \in \Gamma_i$ such that $j \in \Delta^{(q)}$. Then it runs

$$\{\mathsf{lab}_{i,j,k}\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}(\lambda,|C|)}, \widehat{y}_{i,j}^{(q)}\right).$$

     Then, for all $k \in [\widehat{\ell}]$ and $b \in \{0,1\}$ it computes $\mathsf{IBE.ct}_{i,j,k,b}$ as an encryption of $\mathsf{lab}_{i,j,k}$ under the identity (i,j,k,b).
   - If $j \notin S_{\mathsf{crr},i} \cup S_{\mathsf{qrd},i}$, for all $k \in [\widehat{\ell}]$ and $b \in \{0,1\}$ it computes $\mathsf{IBE.ct}_{i,j,k,b}$ by encrypting $0^{L(\lambda,|C|)}$ under the identity (i,j,k,b), where $L(\lambda,|C|)$ is the length of the labels.

4. It returns $\mathsf{ct} = \{\mathsf{IBE.ct}_{i,j,k,b}\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$ to A as the ciphertext.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_3$ for A if it is in the real world and $\mathbf{Game}_4$ if it is in the ideal world. Furthermore, for each instance $i \in [Q]$, the size of the corruption set $S_{\mathsf{crr},i}$ and the number of the sessions $|\Gamma_i|$ are at most t and λ, respectively. Thus, B breaks the multi-instance security of RDMPC if A distinguishes between the two games. □

B.4 Proof of Theorem 3.6

Theorem B.11 (Restate of Theorem 3.6)**.** *Assume that* IBE *satisfies* IND-CPA *security and* SIM-RSO *security,* GC *is a secure garbled circuit scheme, and* RDMPC *is secure. Then,* BCPFE *for the circuit class* $\mathcal{C}_{\ell,s}$ *is* AD-SIM*-secure.*

Proof. To prove the security, we construct the simulator $\mathsf{BCPFE.Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$ for AD-SIM security of BCPFE. The ideal world with simulator Sim for BCPFE and adversary A proceeds as follows.

Ideal World. On input 1^λ, A outputs $\mathsf{prm} = (1^\ell, 1^s)$ and 1^Q, upon which the game runs $\mathsf{Setup}(1^\lambda, \mathsf{prm}, Q)$, gets $\mathsf{mpk} = \mathsf{IBE.mpk}$, and sends it to A. For each $q \in [Q_1]$, A queries secret key for $x^{(q)}$ to receive $\mathsf{sk}^{(q)}$. A then outputs the encryption query C.

Simulating the ciphertext. Here, the simulator SimEnc is given

$$\mathsf{mpk} = \mathsf{IBE.mpk}, \quad \mathcal{V}, \quad 1^{|C|}, \quad 1^Q$$

where

$$\mathcal{V} := \left\{ C(x^{(q)}),\ x^{(q)},\ \mathsf{sk}^{(q)} = \left(u^{(q)}, \Delta^{(q)}, \left\{ \mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} \right\}_{j\in\Delta^{(q)},k\in[\widehat{\ell}]} \right) \right\}_{q\in[Q_1]}$$

and simulates the ciphertext for the adversary. Here, $x^{(q)}$ and $\mathsf{sk}^{(q)}$ are the q-th key query made by the adversary and the secret key returned to it, respectively. Furthermore, C is the circuit chosen by the adversary for the encryption. We then provide the description of the simulator. The simulator proceeds as follows.

1. Choose $u^{(q)} \leftarrow [Q]$ and a random subset $\Delta^{(q)} \subset [N]$ of size n for $q \in [Q_1 + 1, Q]$.

Before continuing the description of the simulator, we prepare some notations. We define Γ_i^{pre} and $\Gamma_i \subseteq [Q_1]$ for $i \in [Q]$ as

$$\Gamma_i^{\mathsf{pre}} := \{q \in [Q_1] : u^{(q)} = i\}, \qquad \Gamma_i := \{q \in [Q] : u^{(q)} = i\}.$$

We also define $S_{\mathsf{crr},i} \subseteq [N]$ and $S_{\mathsf{qrd},i} \subseteq [N]$ for $i \in [Q]$ as

$$S_{\mathsf{qrd},i} := \left(\bigcup_{q\in\Gamma_i^{\mathsf{pre}}} \Delta^{(q)} \right) \setminus S_{\mathsf{crr},i}, \quad S_{\mathsf{crr},i} := \bigcup_{q,q'\in\Gamma_i, q\neq q'} \left(\Delta^{(q)} \cap \Delta^{(q')} \right),$$

Here, the notation is similar to that for the proof of Theorem 3.3. However, we take the secret key queries after the encryption query into account. Intuitively, Γ_i^{pre} (resp., Γ_i) corresponds to the set of queries for the i-th group of the system before the encryption query (resp., throughout the game) . Furthermore, $S_{\mathsf{crr},i}$ corresponds to the set of instances j in group i such that the secret key for the instance (i, j) is issued at least twice throughout the entire game.

We now continue to describe the simulator SimEnc below.

2. If there exists $i \in [Q]$ such that $|\Gamma_i| > \lambda$ or $|S_{\mathsf{crr},i}| > t$, abort.
3. For each of $i \in [Q]$, run

$$\left(\left\{ \widehat{C}_{i,j} \right\}_{j\in S_{\mathsf{crr},i}}, \left\{ \widehat{y}_{i,j}^{(q)} \right\}_{q\in\Gamma_i^{\mathsf{pre}}, j\in\Delta^{(q)}}, \mathsf{RDMPC.st}_i \right) \leftarrow \mathsf{RDMPC.Sim}_0 \left(1^{|C|}, S_{\mathsf{crr},i}, \{C(x^{(q)}), x^{(q)}\}_{q\in\Gamma_i^{\mathsf{pre}}} \right).$$

4. For all $i \in [Q]$ and $j \in [N]$, do the following. There are three cases to consider.
   - If $j \in S_{\mathsf{crr},i}$, set $\mathsf{L}_{i,j}(\cdot) := \mathsf{Local}(\widehat{C}_{i,j}, \cdot)$ and run
   $$\{\mathsf{lab}_{i,j,k,b}\}_{k\in[\widehat{\ell}],b\in\{0,1\}} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathsf{L}_{i,j}\right).$$
   Then, for all $k \in [\widehat{\ell}]$ and $b \in \{0,1\}$ compute
   $$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,b}\right).$$
   - If $j \in S_{\mathsf{qrd},i}$, retrieve $q \in \Gamma_i^{\mathsf{pre}}$ such that $j \in \Delta^{(q)}$. By the definition of $S_{\mathsf{qrd},i}$, there exists unique such q. Then run
   $$\{\mathsf{lab}_{i,j,k}\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}}, \widehat{y}_{i,j}^{(q)}\right).$$
   Then, for all $k \in [\widehat{\ell}]$ and $b \in \{0,1\}$ compute
   $$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k}\right).$$
   Note that here, we encrypt the same label $\mathsf{lab}_{i,j,k}$ for both $b \in \{0,1\}$.
   - If $j \notin S_{\mathsf{crr},i} \cup S_{\mathsf{qrd},i}$, $\mathsf{IBE.ct}_{i,j,k,b}$ is computed by the following. First, set
   $$T := \{(i,j,k,b) : i \in [Q], j \in [N] \backslash (S_{\mathsf{crr},i} \cup S_{\mathsf{qrd},i}), k \in [\widehat{\ell}], b \in \{0,1\}\}. \tag{B.1}$$
   Then, run
   $$\left(\{\mathsf{IBE.ct}_{i,j,k,b}\}_{(i,j,k,b)\in T}, \mathsf{IBE.st}\right) \leftarrow \mathsf{IBE.SimEnc}\left(\mathsf{IBE.mpk}, T, 1^L\right).$$
5. Output the ciphertext
   $$\mathsf{ct} = \{\mathsf{IBE.ct}_{i,j,k,b}\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$$
   and keep its internal state
   $$\mathsf{st} := \left(\{\mathsf{RDMPC.st}_i,\}_{i\in[Q]}, \mathsf{IBE.st}, \left\{u^{(q)}, \Delta^{(q)}\right\}_{q\in[Q]}\right). \tag{B.2}$$

Simulating the secret key after the encryption query. Here, we describe SimKG. It is given
$$\mathsf{st}, \quad \mathsf{msk} = \mathsf{IBE.msk}, \quad \left(C(x^{(q)}), x^{(q)}\right), \quad 1^Q$$
where $x^{(q)}$ is the q-th query with $q \in [Q_1+1, Q_1+Q_2]$ that the adversary makes. Then, it proceeds as follows.

1. Parse its internal state st as Eq. (B.2).
2. Run
$$\left(\widehat{x}^{(q)}, \left\{\widehat{y}_{u^{(q)},j}^{(q)}\right\}_{j\in\Delta^{(q)}}\right) \leftarrow \mathsf{RDMPC.Sim}_1\left(\mathsf{RDMPC.st}_{u^{(q)}}, \Delta^{(q)}, C(x^{(q)}), x^{(q)}\right).$$
3. For all $j \in \Delta^{(q)} \backslash S_{\mathsf{crr},u^{(q)}}$, run
$$\left\{\mathsf{lab}_{u^{(q)},j,k}\right\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}}, \widehat{y}_{u^{(q)},j}^{(q)}\right).$$

4. For all $j \in \Delta^{(q)}$, do the following. There are two cases to consider.
   - If $j \in S_{\mathsf{crr},u^{(q)}}$, for all $k \in [\widehat{\ell}]$, run
   $$\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} \leftarrow \mathsf{IBE.KeyGen}\left(\mathsf{IBE.msk}, (u^{(q)}, j, k, \widehat{x}_k^{(q)})\right).$$
   - If $j \notin S_{\mathsf{crr},u^{(q)}}$, for all $k \in [\widehat{\ell}]$, run
   $$\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} \leftarrow \mathsf{IBE.SimKG}\left(\mathsf{IBE.st}, \mathsf{IBE.msk}, (u^{(q)}, j, k, \widehat{x}_k^{(q)}), \mathsf{lab}_{u^{(q)},j,k}\right).$$
5. Finally, return the secret key $\mathsf{sk}^{(q)} = \left(u^{(q)}, \Delta^{(q)}, \left\{\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right\}_{j\in\Delta^{(q)},k\in[\widehat{\ell}]}\right)$.

The games. To prove the security, we consider the following sequence of games, where $\mathbf{Game}_0$ corresponds to the real game and $\mathbf{Game}_8$ corresponds to the ideal world. We consider similar sequence of games to that of Theorem 3.3, but we have to consider secret key queries *after* the encryption query. To deal with such queries, we introduce additional games. For i, let $\mathcal{E}_i$ denote the event that A outputs 1 in $\mathbf{Game}_i$. The theorem can be proven by showing $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_8]| \leq \mathrm{negl}(\lambda)$.

$\mathbf{Game}_0$: This is the real game $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{BCPFE},\mathsf{A}}(1^\lambda)$ for AD-SIM security defined as per Definition 2.6.

$\mathbf{Game}_1$: In this game, the game chooses $u^{(q)} \leftarrow [Q]$ and random subset $\Delta^{(q)} \subset [N]$ of size n for $q \in [Q_1+1, Q]$ before it answers the encryption query, instead of generating them when it answers the q-th key generation query. This is a conceptual change and we have
$$\Pr[\mathcal{E}_0] = \Pr[\mathcal{E}_1]$$

$\mathbf{Game}_2$: In this game, the game checks whether there exists $i \in [Q]$ such that $|\Gamma_i| > \lambda$ or $|S_{\mathsf{crr},i}| > t$ before generating a ciphertext. If there is, it aborts and otherwise continues the game. By Lemma 3.4, we have $|\Gamma_i| \leq \lambda$ for all $i \in [Q]$ with overwhelming probability. Conditioned on this, by Lemma 3.5, we have $|S_{\mathsf{crr},i}| \leq t$ for all $i \in [Q]$ with overwhelming probability. Therefore, the probability of the above occurring is negligible. Thus, we have
$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_3$: In this game, we change the way how the ciphertext is generated. In particular, we change how $\mathsf{IBE.ct}_{i,j,k,b}$ is generated in the following cases.

- If $j \in S_{\mathsf{qrd},i}$, the game computes
$$\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}}\right),$$
where $q \in [Q_1]$ is the unique index such that $u^{(q)} = i$ and $j \in \Delta^{(q)}$. Such an index exists and is unique by the definition of $S_{\mathsf{qrd},i}$. Note that here, we encrypt the same label $\mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}}$ regardless of the value of $b \in \{0,1\}$.

As we show in Lemma B.12, using IND-CPA security of IBE, we can prove
$$|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_4$: In this game, we change the way how the ciphertext and secret key are generated. In particular, we change how $\mathsf{IBE.ct}_{i,j,k,b}$ is generated in the following cases.

– If $j \notin S_{\mathsf{crr},i} \cup S_{\mathsf{qrd},i}$, $\mathsf{IBE.ct}_{i,j,k,b}$ is computed by the following:

$$\left(\{\mathsf{IBE.ct}_{i,j,k,b}\}_{(i,j,k,b)\in T}, \mathsf{IBE.st}\right) \leftarrow \mathsf{IBE.SimEnc}\left(\mathsf{IBE.mpk}, T, 1^L\right),$$

where T is defined as Equation (B.1).

Furthermore, for the q-th secret key with $q \in [Q_1+1, Q_1+Q_2]$ (i.e., for a secret key query after the encryption query), we change the way how $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ is computed in the following case.

– If $j \notin S_{\mathsf{crr},u^{(q)}}$, for all $k \in [\widehat{\ell}]$, the game computes

$$\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} \leftarrow \mathsf{IBE.SimKG}\left(\mathsf{IBE.st}, \mathsf{IBE.msk}, (u^{(q)}, j, k, \widehat{x}_k^{(q)}), \mathsf{lab}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right),$$

where $\widehat{x}^{(q)}$ and $\mathsf{lab}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ are honestly generated. Note that while the generation of the label was necessary for answering the encryption query in the previous game, it can be deferred until the q-th secret key query in this game.

As we will show in Lemma B.13, using the SIM-RSO security of IBE, we can prove

$$|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_5$: In this game, we further change the way the ciphertext is generated. In particular, we change the way $\mathsf{lab}_{i,j,k,b}$ is generated in the following cases.

– If $j \in S_{\mathsf{qrd},i}$, the game retrieves unique $q \in \Gamma_i$ such that $j \in \Delta^{(q)}$. Then it runs

$$\{\mathsf{lab}_{i,j,k}\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}}, \widehat{y}_{i,j}^{(q)}\right),$$

where $\widehat{y}_{i,j}^{(q)} = \mathsf{Local}(\widehat{C}_{i,j}, \widehat{x}^{(q)})$. Then the game sets

$$\mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}} := \mathsf{lab}_{i,j,k}$$

for all $k \in [\widehat{\ell}]$. The label will be used when it computes $\mathsf{IBE.ct}_{i,j,k,b}$ for $b \in \{0,1\}$ as specified in $\mathbf{Game}_3$.

As we will prove in Lemma B.14, by the security of GC, we can prove

$$|\Pr[\mathcal{E}_4] - \Pr[\mathcal{E}_5]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_6$: In this game, we further change the way the secret key queries after the encryption query are answered. Namely, we change the way how $\mathsf{sk}^{(q)}$ is computed for $q \in [Q_1+1, Q_1+Q_2]$. Recall that in $\mathbf{Game}_4$, $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ was computed by IBE.SimKG algorithm using the label $\mathsf{lab}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$. We change the label to be simulated one in the following case.

– If $j \notin S_{\mathsf{crr},u^{(q)}}$, the game runs

$$\left\{\mathsf{lab}_{u^{(q)},j,k}\right\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}}, \widehat{y}_{u^{(q)},j}^{(q)}\right),$$

where $\widehat{y}_{u^{(q)},j}^{(q)} = \mathsf{Local}(\widehat{C}_{u^{(q)},j}, \widehat{x}^{(q)})$ and sets

$$\mathsf{lab}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} := \mathsf{lab}_{u^{(q)},j,k}$$

for $k \in [\widehat{\ell}]$. As we will show in Lemma B.15, using the security of GC, we can prove

$$|\Pr[\mathcal{E}_5] - \Pr[\mathcal{E}_6]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_7$: In this game, we further change the way the ciphertext and the secret key are generated. In particular, we change the way $\widehat{C}_{i,j}$ and $\widehat{y}^{(q)}_{i,j}$ are generated as follows. When the ciphertext is generated, for each of $i \in [Q]$, the game runs

$$\left(\left\{\widehat{C}_{i,j}\right\}_{j\in S_{\mathsf{crr},i}}, \left\{\widehat{y}^{(q)}_{i,j}\right\}_{q\in\Gamma^{\mathsf{pre}}_i, j\in\Delta^{(q)}}, \mathsf{RDMPC.st}_i\right) \leftarrow \mathsf{RDMPC}_0.\mathsf{Sim}\left(1^{|C|}, S_{\mathsf{crr},i}, \{C(x^{(q)}), x^{(q)}\}_{q\in\Gamma^{\mathsf{pre}}_i}\right).$$

We have $\widehat{C}_{i,j}$ and $\widehat{y}^{(q)}_{i,j}$ only for indices i, j, q that are defined by the output of the above algorithm and leave the values for other indices undefined. Even if it is so, the game is still well-defined since only $\widehat{C}_{i,j}$ for i, j such that $j \in S_{\mathsf{crr},i}$ and $\widehat{y}^{(q)}_{i,j}$ for i, j, q such that $q \in \Gamma^{\mathsf{pre}}_i$, $j \in \Delta^{(q)}$ are needed for the generation of the ciphertext thanks to the changes introduced from $\mathbf{Game}_3$ to $\mathbf{Game}_6$.

We also change how the secret keys are generated. Recall that $\widehat{y}^{(q)}_{u^{(q)},j} = \mathsf{Local}(\widehat{C}_{u^{(q)},j}, \widehat{x}^{(q)})$ for $q \in [Q_1+1, Q_1+Q_2]$ and $j \in \Delta^{(q)} \backslash S_{\mathsf{crr},u^{(q)}}$ are required for the generation of the q-th secret key in the previous game. These values are not defined by the above. In this game, we generate them as

$$\left(\widehat{x}^{(q)}, \left\{\widehat{y}^{(q)}_{u^{(q)},j}\right\}_{j\in\Delta^{(q)}}\right) \leftarrow \mathsf{RDMPC}_1.\mathsf{Sim}\left(\mathsf{RDMPC.st}_{u^{(q)}}, \Delta^{(q)}, C(x^{(q)}), x^{(q)}\right).$$

As we will show in Lemma B.16, we have

$$|\Pr[\mathcal{E}_6] - \Pr[\mathcal{E}_7]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_8$: This is the ideal game $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{BCPFE},\mathsf{Sim}}(1^\lambda)$ defined as per Definition 2.6. By inspection, it can be seen that this is the same as the previous game. Therefore, we have

$$\Pr[\mathcal{E}_7] = \Pr[\mathcal{E}_8].$$

To finish the proof of the theorem, it suffices to prove Lemmas B.12 to B.16. since we have $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_8]| \leq \sum_{i\in[0,7]} |\Pr[\mathcal{E}_i] - \Pr[\mathcal{E}_{i+1}]|$ by the triangle inequality. □

Lemma B.12. *We have* $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda)$ *by the* IND-CPA *security of* IBE.

Proof. The proof is almost the same as Lemma B.8. We only highlight the difference.

- Here, B makes an encryption query for an identity (i, j, k, b) only for the case of $j \in S_{\mathsf{qrd},i}$ and not for the case of $j \notin S_{\mathsf{qrd},i} \cup S_{\mathsf{crr},i}$. In particular, an encryption query is made for (i, j, k, b) such that $j \in S_{\mathsf{qrd},i}$ and $b = 1 - \widehat{x}^{(q)}_k$, where $q \in [Q_1]$ is the unique index such that $u^{(q)} = i$ and $j \in \Delta^{(q)}$. Note that the case of $j \notin S_{\mathsf{qrd},i} \cup S_{\mathsf{crr},i}$ is handled by the next game hop.
- B handles secret key queries made by A similarly to Lemma B.8, even after the encryption query. Namely, B makes a secret key query for (i, j, k, b) such that $i = u^{(q)}$, $j \in \Delta^{(q)}$, $k \in [\widehat{\ell}]$, and $b = \widehat{x}^{(q)}_k$ for $q \in [Q]$.
- Discussion about the disjointness of the encryption queries and secret key queries remains the same, since the definition of the set $S_{\mathsf{qrd},i}$ here takes into account the secret key queries made after the encryption query.

□

Lemma B.13. *We have* $|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \leq \mathrm{negl}(\lambda)$ *by the* SIM-RSO *security of* IBE.

Proof. Assume $|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]|$ is non-negligle. We then describe an adversary B that breaks SIM-RSO security of IBE as follows. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = (1^\ell, 1^s)$. B then computes L as a function of λ and s and declares the length L of the message space in unary form. Then, B is given IBE.mpk and gives $\mathsf{mpk} = \mathsf{IBE.mpk}$ to A as master public key.

Answering the secret key queries before the encryption query. When A makes the q-th secret key for $x^{(q)} \in \{0,1\}^\ell$, B does the following.

1. It chooses $\widehat{x}^{(q)}$, $u^{(q)}$, and $\Delta^{(q)}$ as in the honest key generation algorithm.
2. For all $j \in \Delta^{(q)}$ and $k \in [\widehat{\ell}]$, it makes a secret key query for the identity $(u^{(q)}, j, k, \widehat{x}_k^{(q)})$ and receives $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$.
3. It returns $\mathsf{sk} = \left(u^{(q)}, \Delta^{(q)}, \left\{\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right\}_{j\in\Delta^{(q)},k\in[\widehat{\ell}]}\right)$ to A as a secret key.

Answering the encryption query. When A submits an encryption query C, B does the following:

1. It choose $u^{(q)}$ and $\Delta^{(q)}$ for $q \in [Q_1+1, Q]$ as in key generation algorithm. It then aborts and outputs a random bit if there exists $i \in [Q]$ such that $|\Gamma_i| > \lambda$ or $|S_{\mathsf{crr},i}| > t$.
2. It computes $\{\mathsf{lab}_{i,j,k,b}\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$ as honest encryption algorithm.
3. It computes $\{\mathsf{IBE.ct}_{i,j,k,b}\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$ as follows. There are three cases:
   - If $j \in S_{\mathsf{qrd},i}$, it computes $\mathsf{IBE.ct}_{i,j,k,b}$ for all $k \in [\widehat{\ell}]$, $b \in \{0,1\}$ as $\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k,\widehat{x}_k^{(q)}}\right)$.
   - If $j \in S_{\mathsf{crr},i}$, it computes $\mathsf{IBE.ct}_{i,j,k,b}$ for all $k \in [\widehat{\ell}]$, $b \in \{0,1\}$ as in the honest encryption algorithm.
   - For $(i,j,k,b) \in T$ where T is defined as Equation (B.1), B computes $\mathsf{IBE.ct}_{i,j,k,b}$ using its challenger as follows. It submits the target set T and messages $\{\mathsf{lab}_{i,j,k,b}\}_{(i,j,k,b)\in T}$ to its challenger. The IBE challenger replies B with $\{\mathsf{IBE.ct}_{i,j,k,b}\}_{(i,j,k,b)\in T}$, where it is generated as either
   $$\{\mathsf{IBE.ct}_{i,j,k,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,k,b), \mathsf{lab}_{i,j,k}\right)\}_{(i,j,k,b)\in T}$$
   or
   $$\left(\{\mathsf{IBE.ct}_{i,j,k,b}\}_{(i,j,k,b)\in T}, \mathsf{IBE.st}\right) \leftarrow \mathsf{IBE.SimEnc}\left(\mathsf{IBE.mpk}, T, 1^L\right),$$
   depending on whether it is in the real world or simulated world. Note that by the definition of T, T does not contain any identity for which B has made a secret key query.
   - It then returns the ciphertext $\mathsf{ct} = \{\mathsf{IBE.ct}_{i,j,k,b}\}_{i\in[Q],j\in[N],k\in[\widehat{\ell}],b\in\{0,1\}}$ to A.

Answering the secret key queries after the encryption query. A then continues to make key queries. When A makes a q-th key query for $x^{(q)}$, B proceeds as follows.

1. It computes $\widehat{x}^{(q)} \leftarrow \mathsf{InpEnc}(1^\lambda, 1^\lambda, 1^\ell, x^{(q)})$.
2. It computes $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ for all $j \in \Delta^{(q)}$ and $k \in [\widehat{\ell}]$ as follows. There are two cases:

– If $j \in S_{\mathsf{crr},u^{(q)}}$, B makes a secret key query for the identity $(u^{(q)}, j, k, \widehat{x}_k^{(q)})$ to its challenger. Honestly generated $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ is returned to B. Note that by the definition of T, we have $(u^{(q)}, j, k, \widehat{x}_k^{(q)}) \notin T$.

– If $j \notin S_{\mathsf{crr},u^{(q)}}$, B submits the identity $(u^{(q)}, j, k, \widehat{x}_k^{(q)})$ to its challenger as a corruption query. Then, the challenger runs either

$$\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} \leftarrow \mathsf{IBE.KeyGen}\left(\mathsf{IBE.msk}, (u^{(q)}, j, k, \widehat{x}_k^{(q)})\right),$$

or

$$\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} \leftarrow \mathsf{IBE.SimKG}\left(\mathsf{IBE.st}, \mathsf{IBE.msk}, (u^{(q)}, j, k, \widehat{x}_k^{(q)}), \mathsf{lab}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right),$$

depending on whether B is in the real world or simulated world. Note that by the definition of T, we have $(u^{(q)}, j, k, \widehat{x}_k^{(q)}) \in T$ in this case.

3. Finally, it returns $\mathsf{sk}^{(q)} = \left(u^{(q)}, \Delta^{(q)}, \left\{\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right\}_{j\in\Delta^{(q)},k\in[\widehat{\ell}]}\right)$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_3$ for A if it is in the real world and $\mathbf{Game}_4$ if it is in the ideal world. Thus, B breaks the $\mathsf{SIM\text{-}RSO}$ security of IBE if A distinguishes between the two games. □

Lemma B.14. *We have* $|\Pr[\mathcal{E}_4] - \Pr[\mathcal{E}_5]| \leq \mathrm{negl}(\lambda)$ *by the security of* GC.

Proof. The proof is almost the same as that for Lemma B.9. We only highlight the difference.

- For the case $(i, j, k, b) \in T$, $\mathsf{IBE.ct}_{i,j,k,b}$ is simulated using $\mathsf{IBE.SimEnc}$ whereas it is simulated by encrypting a fixed string in Lemma B.9.
- The secret key queries after the encryption query are simulated by $\mathsf{IBE.SimKG}$ when we generate an IBE secret key for an identity $(i, j, k, b) \in T$.

□

Lemma B.15. *We have* $|\Pr[\mathcal{E}_5] - \Pr[\mathcal{E}_6]| \leq \mathrm{negl}(\lambda)$ *by the security of* GC.

Proof. Assume $|\Pr[\mathcal{E}_5] - \Pr[\mathcal{E}_6]|$ is non-negligle. We then describe an adversary B that breaks the multi-instance security of GC as follows. Note that since the multi-instance security of GC is implied by the single instance security as we observed in Remark A.4, this is a contradiction. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = (1^\ell, 1^s)$. B then runs $\mathsf{Setup}(1^\lambda, \mathsf{prm}, Q)$ to obtain mpk and msk and then sends mpk to A as the master public key.

Answering the secret key queries before the encryption query. When A makes a secret key query, B answers it using msk.

Answering the encryption query. When A submits an encryption query C, B answers it as $\mathbf{Game}_5$. It is straightforward to do so since no secret is required for the simulation. Recall that during the simulation of the ciphertext, B generates

$$\mathsf{IBE.st}, \quad \{u^{(q)}, \Delta^{(q)}\}_{q\in[Q_1+1,Q]}, \quad \{\widehat{C}_{i,j}\}_{i\in[Q],j\in[N]}$$

as specified in $\mathbf{Game}_5$ and keeps them for future use.

Answering the secret key queries after the encryption query. A then continues to make key queries. When A makes a q-th key query for $x^{(q)}$, B proceeds as follows.

1. It computes $\widehat{x}^{(q)} \leftarrow \mathsf{InpEnc}(1^\lambda, 1^\lambda, 1^\ell, x^{(q)})$.
2. It computes $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ for all $j \in \Delta^{(q)}$ and $k \in [\widehat{\ell}]$ as follows. There are two cases:
   - If $j \in S_{\mathsf{crr},u^{(q)}}$, B honestly generates a secret key for the identity $(u^{(q)}, j, k, \widehat{x}_k^{(q)})$ for all $k \in [\widehat{\ell}]$ using $\mathsf{msk} = \mathsf{IBE.msk}$.
   - If $j \notin S_{\mathsf{crr},u^{(q)}}$, send $\mathsf{L}_{u^{(q)},j}(\cdot) := \mathsf{Local}(\widehat{C}_{u^{(q)},j}, \cdot)$ and $\widehat{x}^{(q)}$ to its challenger. Then, the challenger computes either
   $$\left\{\mathsf{lab}_{u^{(q)},j,k,b}\right\}_{k\in[\widehat{\ell}],b\in\{0,1\}} \leftarrow \mathsf{GC.Garble}\left(1^\lambda, \mathsf{L}_{u^{(q)},j}\right).$$
   or
   $$\left\{\mathsf{lab}_{u^{(q)},j,k}\right\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}}, \mathsf{L}_{u^{(q)},j}(\widehat{x}^{(q)})\right)$$
   depending on whether B is in the real world or simulated world. Then, B is given $\left\{\mathsf{lab}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right\}_{k\in[\widehat{\ell}]}$, where $\mathsf{lab}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} = \mathsf{lab}_{u^{(q)},j,k}$ if it is in the ideal world. Then, B runs
   $$\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}} \leftarrow \mathsf{IBE.SimKG}\left(\mathsf{IBE.st}, \mathsf{IBE.msk}, (u^{(q)}, j, k, \widehat{x}_k^{(q)}), \mathsf{lab}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right),$$
   to obtain $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ for all $k \in [\widehat{\ell}]$.
3. Finally, it returns $\mathsf{sk}^{(q)} = \left(u^{(q)}, \Delta^{(q)}, \left\{\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right\}_{j\in\Delta^{(q)},k\in[\widehat{\ell}]}\right)$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_5$ for A if it is in the real world and $\mathbf{Game}_6$ if it is in the ideal world. Thus, B breaks the multi-instance security of GC if A distinguishes between the two games. □

Lemma B.16. *We have* $|\Pr[\mathcal{E}_6] - \Pr[\mathcal{E}_7]| \leq \mathrm{negl}(\lambda)$ *by the security of* RDMPC.

Proof. The proof is similar to Lemma B.10, but we have to consider secret key queries after the encryption query here. Assume $|\Pr[\mathcal{E}_6] - \Pr[\mathcal{E}_7]|$ is non-negligle. We then describe an adversary B that breaks the multi-instance security of RDMPC as follows. Note that since the multi-instance security of RDMPC is implied by the single instance security as we observed in Remark B.5, this is a contradiction.

Simulating the Setup. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = (1^\ell, 1^s)$. B then runs $\mathsf{Setup}(1^\lambda, \mathsf{prm}, Q)$ to obtain mpk and msk and then sends mpk to A as the master public key. Furthermore, B does the following.

1. It chooses $u^{(q)}$ and $\Delta^{(q)}$ for $q \in [Q]$ in advance. It then aborts and outputs a random bit if there exists $i \in [Q]$ such that $|\Gamma_i| > \lambda$ or $|S_{\mathsf{crr},i}| > t$.
2. It then declares the number of instances $M = Q$, the query bound $R = \lambda$, and the input length of circuits ℓ to its challenger. The security parameter λ and R specify the total number of parties N, number of parties n participating in any session, and threshold t. Furthermore, for $i \in [Q]$, B does the following.
   (a) It submits $S_{\mathsf{crr},i}$ to its challenger as the corruption set in the i-th instance.
   (b) It submits the sets $\{\Delta^{(q)}\}_{q\in\Gamma_i}$ to its challenger where $\Delta^{(q)}$ is the set of parties that will participate in the session in the i-th instance.

Answering the secret key queries before the encryption query. When A makes a secret key query $x^{(q)}$, B does the following.

1. It submits $x^{(q)}$ to the $u^{(q)}$-th instance. Then, the challenger runs
$$\widehat{x}^{(q)} \leftarrow \mathsf{InpEnc}(1^\lambda, 1^\lambda, 1^\ell, x^{(q)})$$
and returns $\widehat{x}^{(q)}$ to B.
2. Given $\widehat{x}^{(q)}$, it honestly computes $\mathsf{sk}^{(q)}$ using $\mathsf{IBE.msk}$ and returns it to A.

Answering the encryption query. When A submits an encryption query C, B proceeds as follows.

1. It declares the circuit C as its target for all instances of RDMPC.
2. Then, for each of $i \in [Q]$, the challenger either computes
$$(\widehat{C}_{i,1}, \ldots, \widehat{C}_{i,N}) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^\lambda, 1^\ell, C), \quad \widehat{y}^{(q)}_{i,j} := \mathsf{Local}(\widehat{C}_{i,j}, \widehat{x}^{(q)}) \quad \text{for } q \in \Gamma^{\mathsf{pre}}_i, j \in \Delta^{(q)}$$
or
$$\left(\left\{\widehat{C}_{i,j}\right\}_{j\in S_{\mathsf{crr},i}}, \left\{\widehat{y}^{(q)}_{i,j}\right\}_{q\in\Gamma^{\mathsf{pre}}_i, j\in\Delta^{(q)}}, \mathsf{RDMPC.st}_i\right) \leftarrow \mathsf{RDMPC.Sim}_0\left(1^{|C|}, S_{\mathsf{crr},i}, \left\{C(x^{(q)}), x^{(q)}\right\}_{q\in\Gamma^{\mathsf{pre}}_i}\right)$$
depending on whether B is in the real game or ideal game. Then, the challenger returns $\left(\left\{\widehat{C}_{i,j}\right\}_{j\in S_{\mathsf{crr},i}}, \left\{\widehat{y}^{(q)}_{i,j}\right\}_{q\in\Gamma^{\mathsf{pre}}_i, j\in\Delta^{(q)}}\right)$ to B for all $i \in [Q]$.
3. Given the value, B simulates the ciphertext as $\mathbf{Game}_6$ and returns it to A. This is possible without using any secret information. During the simulation B generates $\mathsf{IBE.st}$ and keeps the value for future use.

Answering the secret key queries after the encryption query. A then continues to make key queries. When A makes a q-th key query for $x^{(q)}$, B proceeds as follows.

1. It submits $x^{(q)}$ to the $u^{(q)}$-th instance. Then, the challenger either computes
$$\widehat{x}^{(q)} \leftarrow \mathsf{InpEnc}(1^\lambda, 1^\lambda, 1^\ell, x^{(q)}), \quad \widehat{y}^{(q)}_{u^{(q)},j} := \mathsf{Local}(\widehat{C}_{u^{(q)},j}, \widehat{x}^{(q)}) \quad \text{for } j \in \Delta^{(q)},$$
or
$$\left(\widehat{x}^{(q)}, \left\{\widehat{y}^{(q)}_{u^{(q)},j}\right\}_{j\in\Delta^{(q)}}\right) \leftarrow \mathsf{RDMPC.Sim}_1\left(\mathsf{RDMPC.st}_{u^{(q)}}, \Delta^{(q)}, C(x^{(q)}), x^{(q)}\right)$$
depending on whether B is in the real game or ideal game. Then, the challenger returns $\left(\widehat{x}^{(q)}, \left\{\widehat{y}^{(q)}_{u^{(q)},j}\right\}_{j\in\Delta^{(q)}}\right)$ to B.
2. It computes $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}^{(q)}_k}$ for all $j \in \Delta^{(q)}$ and $k \in [\widehat{\ell}]$ as follows. There are two cases to consider:
   - If $j \notin S_{\mathsf{crr},u^{(q)}}$, it runs
   $$\left\{\mathsf{lab}_{u^{(q)},j,k}\right\}_{k\in[\widehat{\ell}]} \leftarrow \mathsf{GC.Sim}\left(1^\lambda, 1^{\widehat{\ell}}, 1^{\widehat{s}}, \widehat{y}^{(q)}_{u^{(q)},j}\right).$$
   Then, for all $k \in [\widehat{\ell}]$, it runs
   $$\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}^{(q)}_k} \leftarrow \mathsf{IBE.SimKG}\left(\mathsf{IBE.st}, \mathsf{IBE.msk}, (u^{(q)}, j, k, \widehat{x}^{(q)}_k), \mathsf{lab}_{u^{(q)},j,k}\right).$$

– If $j \in S_{\mathsf{crr},u^{(q)}}$, it honestly generates $\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}$ for all $k \in [\widehat{\ell}]$ using $\mathsf{IBE.msk}$.

3. Finally, it returns $\mathsf{sk}^{(q)} = \left(u^{(q)}, \Delta^{(q)}, \left\{\mathsf{IBE.sk}_{u^{(q)},j,k,\widehat{x}_k^{(q)}}\right\}_{j\in\Delta^{(q)},k\in[\widehat{\ell}]}\right)$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_6$ for A if it is in the real world and $\mathbf{Game}_7$ if it is in the ideal world. Furthermore, for each instance $i \in [Q]$, the size of the corruption set $S_{\mathsf{crr},i}$ and the number of the sessions $|\Gamma_i|$ are at most t and λ, respectively. Thus, B breaks the multi-instance security of RDMPC if A distinguishes between the two games. □

B.5 Efficiency of Full-fledged CPFE from Section 3.4

We observe that Setup and KeyGen run in time $\mathrm{poly}(\lambda, \ell, \log 2^i) = \mathrm{poly}(\lambda, \ell)$ by the efficiency property of BCPFE. Here, we crucially use the fact that the setup and key generation algorithms of BCPFE runs polylogarithmic in the collusion bound. Otherwise, Setup and KeyGen defined above does not run in polynomial time. We also observe that Enc and Dec run in time $2^u \mathrm{poly}(\lambda, \ell) = Q \cdot \mathrm{poly}(\lambda, \ell)$ by the efficiency property of BCPFE.

B.6 Proof of Theorem 3.7

Theorem B.17 (Restate of Theorem 3.7)**.** *If* BCPFE *satisfies* $\mathsf{AD\text{-}SIM}$ *security as bounded FE scheme, then so does* CPFE *scheme as FE with delayed collusion bound. Similarly, if* BCPFE *satisfies* $\mathsf{NA\text{-}SIM}$ *security as bounded FE scheme, then so does* CPFE *scheme as FE with delayed collusion bound.*

Proof. Here, we prove the statement for the case of $\mathsf{AD\text{-}SIM}$. The proof for $\mathsf{NA\text{-}SIM}$ is similar and simpler.

Ideal world: By the definition, we have a simulators $(\mathsf{BCPFE.SimEnc}, \mathsf{BCPFE.SimKG})$ for BCPFE. We construct the simulators $(\mathsf{SimEnc}, \mathsf{SimKG})$ for CPFE using them as follows.

Simulating the ciphertext. Here, we describe SimEnc. It is given

$$\mathsf{mpk} = \mathsf{BCPFE.mpk},\ 1^{|Q|},\ 1^{|C|},\ \mathcal{V} := \left\{C(x^{(q)}), x^{(q)}, \mathsf{sk}^{(q)}\right\}_{q\in[Q_1]}$$

where Q is the collusion bound chosen by the adversary and $x^{(q)}$ and $\mathsf{sk}^{(q)}$ are the q-th secret key query and answer to it, respectively. Given the input, it proceeds as follows.

1. Find u such that $2^{u-1} < Q \leq 2^u$.
2. Parse $\mathsf{mpk} \to \{\mathsf{BCPFE.mpk}_i\}_{i\in[\lambda]}$ and retrieve $\mathsf{BCPFE.mpk}_u$.
3. Parse $\mathsf{sk}^{(q)} \to \{\mathsf{BCPFE.sk}_i^{(q)}\}_{i\in[\lambda]}$ and define $\mathcal{V}_u = \{C(x^{(q)}), x^{(q)}, \mathsf{BCPFE.sk}_u^{(q)}\}_{q\in[Q_1]}$.
4. Run $\mathsf{BCPFE.ct}_u \leftarrow \mathsf{BCPFE.SimEnc}(\mathsf{mpk}_u, \mathcal{V}_u, 1^\ell, 1^{2^u})$ and return $\mathsf{ct} = \mathsf{BCPFE.ct}_u$.

Simulating the secret key. Here, we describe SimKG. When the adversary makes the q-th key query, where $q \in [Q_1 + 1, Q_1 + Q_2]$, it is given

$$\left(\mathsf{msk} = \mathsf{BCPFE.msk},\ \mathsf{st},\ x^{(q)},\ C(x^{(q)})\right)$$

as input. Then, it proceeds as follows.

1. Parse $\mathsf{msk} \to \{\mathsf{BCPFE.msk}_i\}_{i\in[\lambda]}$.
2. Run $\mathsf{BCPFE.sk}_i \leftarrow \mathsf{BCPFE.KeyGen}(\mathsf{BCPFE.msk}_i, x, 2^i)$ for $i \in [\lambda]\backslash\{u\}$.
3. Run $\mathsf{BCPFE.sk}_u \leftarrow \mathsf{BCPFE.SimKG}(\mathsf{BCPFE.sk}_u, x^{(q)}, C(x^{(q)}))$.
4. Return $\mathsf{sk} = \{\mathsf{BCPFE.sk}_i\}_{i\in[\lambda]}$.

The games. We now provide the games to argue indistinguishability from the real world. To do so, we define $\mathbf{Game}_i$ for $i \in [0, i_{\max}]$ as follows, where $i_{\max} := \lceil \log Q_{\max} \rceil$ and $Q_{\max}$ is the upper bound on the value of Q chosen by the adversary. We can use the running time of the adversary as $Q_{\max}$ for example. Note that we have $i_{\max} < \lambda$ because we have $Q_{\max} < 2^\lambda$.

$\mathbf{Game}_i$. In this game, the ciphertext is generated as in the real world if $u \geq i$ and as in the ideal world otherwise, where u is an integer such that $2^{u-1} < Q \leq 2^u$. Similarly, the secret keys after the challenge ciphertext are generated as in the real world if $u \leq i$ and as in the ideal world otherwise.

It is easy to see that $\mathbf{Game}_0$ is equivalent to the real game $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{CPFE,A}}(1^\lambda)$ defined as per Theorem 2.7 and $\mathbf{Game}_{i_{\max}}$ is equivalent to the ideal game $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CPFE,Sim}}(1^\lambda)$. It is easy to see that $\mathbf{Game}_{i-1}$ and $\mathbf{Game}_i$ are indistinguishable by a straightforward reduction where the reduction algorithm chooses $(\mathsf{prm}, 1^{2^i})$ as its target parameter and generates BCPFE instances other than index i by itself at the beginning of the game. The theorem readily follows. □

C Missing Details from Section 4

Correctness. Here we show the correctness of our KPFE scheme from Section 4.

Theorem C.1. *For any given* $\mathsf{inp}(\lambda), \mathsf{dep}(\lambda)$ *and* $\mathsf{out}(\lambda)$*, assume* 1KPFE *supports the circuit class* $\mathcal{C}_{\mathsf{inp,dep,out}}$*. Further, assume* CPFE *supports the circuit class* $\mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$*. Then, the* KPFE *scheme Section 4 is correct.*

Proof. Consider any honestly generated ciphertext $\mathsf{ct} = \mathsf{CPFE.ct}$ and secret key $\mathsf{sk} = (\mathsf{CPFE.sk}, \mathsf{1KPFE.sk})$, where the latter is associated with some string $\mathsf{1KPFE.mpk} \in \{0,1\}^{\mathsf{inp}'}$ and some circuit $C \in \mathcal{C}_{\mathsf{inp,dep,out}}$. Then the correctness of CPFE and PRF dictates that

$$\begin{aligned} y &= \mathsf{CPFE.Dec}(\mathsf{CPFE.sk}, \mathsf{CPFE.ct}) = \mathsf{E}_{[x,\mathsf{K}]}(\mathsf{1KPFE.mpk}) \\ &= \mathsf{1KPFE.Enc}(\mathsf{1KPFE.mpk}, x; \mathsf{PRF}(\mathsf{K}, \mathsf{1KPFE.mpk})), \end{aligned}$$

since $\mathsf{E}_{[x,\mathsf{K}]} \in \mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$. Next, the correctness of KPFE further ensures that $z = \mathsf{KPFE.Dec}(\mathsf{1KPFE.sk}, y) = C(x)$. This completes the proof of the theorem. □

Efficiency. Here, we evaluate the efficiency of our KPFE construction from Section 4. In particular, we show in the following theorem that our scheme is succinct as per Remark 2.10.

Theorem C.2. *Let* $\mathsf{inp}(\lambda), \mathsf{dep}(\lambda)$ *and* $\mathsf{out}(\lambda)$ *be polynomials in* λ*. Further, let* 1KPFE *be a succinct KPFE scheme for circuit family* $\mathcal{C}_{\mathsf{inp,dep,out}}$ *and* CPFE *be a CPFE scheme for circuit family* $\mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$*. Then, the KPFE scheme* $\mathsf{KPFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is succinct and runs in polynomial time.*

Proof. By the succinctness of $\mathsf{1KPFE.Setup}(1^\lambda, \mathsf{prm})$, the running time of the encryption algorithm 1KPFE.Enc is independent of the circuit size, which is unbounded. Instead, the running time grows only with $\mathsf{inp}(\lambda), \mathsf{dep}(\lambda)$ and $\mathsf{out}(\lambda)$. This implies that $\mathsf{size}'(\lambda) = \mathrm{poly}(\lambda, \mathsf{inp}(\lambda), \mathsf{dep}(\lambda), \mathsf{out}(\lambda))$ and $\mathsf{E}_{[x,\mathsf{K}]}$ can be computed in time $\mathrm{poly}(\lambda, \mathsf{inp}(\lambda), \mathsf{dep}(\lambda), \mathsf{out}(\lambda))$. Given this, it is straightforward to bound the running time of the algorithms. In particular, the running time of the encryption algorithm is independent from the size of the circuits and thus the scheme is succinct. □

We also observe that the running time of the encryption algorithm is linear in Q, which is inherited from the same property of CPFE.

Proof of Security. Here, we prove the security of our KPFE scheme from Section 4.

Theorem C.3. *Assume* PRF *is a secure pseudorandom function. Further, assume* 1KPFE *for the circuit family* $\mathcal{C}_{\mathsf{inp,dep,out}}$ *is* 1-NA-SIM*-secure as per Definition 2.7 and* CPFE *for the circuit family* $\mathcal{C}_{\mathsf{inp}',\mathsf{size}'}$ *is* AD-SIM*-secure as per Definition 2.6. Then,* KPFE *for the circuit family* $\mathcal{C}_{\mathsf{inp,dep,out}}$ *is* AD-SIM*-secure as per Definition 2.6.*

Proof. Since CPFE satisfies AD-SIM security, we have a simulator $\mathsf{CPFE.Sim} = (\mathsf{CPFE.SimEnc}, \mathsf{CPFE.SimKG})$ for CPFE. Similarly, since 1KPFE satisfies NA-SIM security, we have a simulator $\mathsf{1KPFE.Sim} = (\mathsf{1KPFE.SimEnc}, \bot)$. We construct the simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$ for KPFE using these algorithms. The ideal world with simulator Sim for KPFE and adversary A proceeds as follows.

Ideal world: On input 1^λ, A outputs $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$, upon which the game runs $\mathsf{Setup}(1^\lambda, \mathsf{prm})$, gets $\mathsf{mpk} = \mathsf{CPFE.mpk}$, and sends it to A. For each $q \in [Q_1]$, A queries secret key for circuit $C^{(q)}$ to receive $\mathsf{sk}^{(q)} = (\mathsf{CPFE.sk}^{(q)}, \mathsf{1KPFE.sk}^{(q)})$. Note that $\mathsf{sk}^{(q)}$ contains 1KPFE master public key $\mathsf{1KPFE.mpk}^{(q)}$ and $C^{(q)}$ in $\mathsf{CPFE.sk}^{(q)}$ and $\mathsf{1KPFE.sk}^{(q)}$ respectively. It then outputs $(x, 1^Q)$.

Simulating the ciphertext after prechallenge queries. On input

$$\mathsf{mpk} = \mathsf{CPFE.mpk},\ \mathcal{V} = \left\{z^{(q)} := C^{(q)}(x), C^{(q)}, \mathsf{sk}^{(q)}\right\}_{q\in[Q_1]},\ 1^{|x|},\ 1^Q$$

SimEnc proceeds as follows.

1. For all $q \in [Q_1]$, run

$$\mathsf{1KPFE.ct}^{(q)} \leftarrow \mathsf{1KPFE.SimEnc}(\mathsf{1KPFE.mpk}^{(q)}, \mathcal{V}^{(q)}_{\mathsf{1KPFE}}, 1^{|x|}),$$

   where $\mathcal{V}^{(q)}_{\mathsf{1KPFE}} := \{z^{(q)}, C^{(q)}, \mathsf{1KPFE.sk}^{(q)}\}$.

2. Further, it computes size' and run

$$(\mathsf{CPFE.ct}, \mathsf{CPFE.st}) \leftarrow \mathsf{CPFE.SimEnc}\left(\mathsf{CPFE.mpk}, \mathcal{V}_{\mathsf{CPFE}}, 1^{\mathsf{size}'}, 1^Q\right)$$

   where $\mathcal{V}_{\mathsf{CPFE}} := \{\mathsf{1KPFE.ct}^{(q)}, \mathsf{1KPFE.mpk}^{(q)}, \mathsf{CPFE.sk}^{(q)}\}_{q\in[Q_1]}$.

3. Output the ciphertext $\mathsf{ct} = \mathsf{CPFE.ct}$ and internal state $\mathsf{st} = \mathsf{CPFE.st}$.

After giving the ciphertext ct, A makes secret key queries. The queries are answered by the simulator.

Simulating the secret key after the challenge query. For q-th query with $q \in [Q_1 + 1, Q_1 + Q_2]$, SimKG is given

$$\mathsf{CPFE.st},\ \mathsf{msk} = \mathsf{CPFE.msk},\ z^{(q)} := C^{(q)}(x),\ C^{(q)}$$

and proceeds as follows.

1. Run $(\mathsf{1KPFE.mpk}^{(q)}, \mathsf{1KPFE.msk}^{(q)}) \leftarrow \mathsf{1KPFE.Setup}(1^\lambda, \mathsf{prm})$.
2. Run $\mathsf{1KPFE.sk}^{(q)} \leftarrow \mathsf{1KPFE.KeyGen}(\mathsf{1KPFE.msk}^{(q)}, C^{(q)})$.
3. Run $\mathsf{1KPFE.ct}^{(q)} \leftarrow \mathsf{1KPFE.SimEnc}(\mathsf{1KPFE.mpk}^{(q)}, \mathcal{V}^{(q)}_{\mathsf{1KPFE}}, 1^{|x|})$, where $\mathcal{V}^{(q)}_{\mathsf{1KPFE}} := \{z^{(q)}, C^{(q)}, \mathsf{1KPFE.sk}^{(q)}\}$.
4. Run $\mathsf{CPFE.sk}^{(q)} \leftarrow \mathsf{CPFE.SimKG}(\mathsf{CPFE.st}, \mathsf{CPFE.msk}, \mathsf{1KPFE.ct}^{(q)}, \mathsf{1KPFE.mpk}^{(q)})$.

5. Return $\mathsf{sk}^{(q)} = (\mathsf{CPFE.sk}^{(q)}, \mathsf{1KPFE.sk}^{(q)})$.

The games. To prove the security, we consider the following sequence of games, where $\mathbf{Game}_0$ corresponds to the real game and $\mathbf{Game}_3$ corresponds to the ideal world. For $i \in [0,3]$, let $\mathcal{E}_i$ denote the event that A outputs 1 in $\mathbf{Game}_i$. In particular, we prove that $|\Pr[\mathcal{E}_i] - \Pr[\mathcal{E}_{i+1}]| \leq \mathrm{negl}(\lambda)$ for all $i \in [0,2]$, which implies $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda)$.

$\mathbf{Game}_0$**.** This is the real game $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{KPFE},\mathsf{A}}(1^\lambda)$ as per Definition 2.6.

$\mathbf{Game}_1$**.** In this game, we replace the encryption algorithm and key generation algorithms of CPFE with its simulator CPFE.SimEnc and CPFE.SimKG respectively. In particular, CPFE.ct is generated as follows.

1. For all $q \in [Q_1]$, compute $\mathsf{1KPFE.ct}^{(q)} = \mathsf{E}_{[x,\mathsf{K}]}(\mathsf{1KPFE.mpk}^{(q)})$.
2. Further, it computes size' and run

$$(\mathsf{CPFE.ct}, \mathsf{CPFE.st}) \leftarrow \mathsf{CPFE.SimEnc}\left(\mathsf{CPFE.mpk}, \mathcal{V}_{\mathsf{CPFE}}, 1^{\mathsf{size}'}, 1^Q\right)$$

where $\mathcal{V}_{\mathsf{CPFE}} := \{\mathsf{1KPFE.ct}^{(q)}, \mathsf{1KPFE.mpk}^{(q)}, \mathsf{CPFE.sk}^{(q)}\}_{q\in[Q_1]}$.

Furthermore, we change the way how $\mathsf{CPFE.sk}^{(q)}$ for $q \in [Q_1+1, Q_1+Q_2]$ is generated as follows.

1. Compute $\mathsf{1KPFE.ct}^{(q)} = \mathsf{E}_{[x,\mathsf{K}]}(\mathsf{1KPFE.mpk}^{(q)})$.
2. Run $\mathsf{CPFE.sk}^{(q)} \leftarrow \mathsf{CPFE.SimKG}(\mathsf{CPFE.st}, \mathsf{CPFE.msk}, \mathsf{1KPFE.ct}^{(q)}, \mathsf{1KPFE.mpk}^{(q)})$.

As we will show in Lemma C.4, using AD-SIM security of CPFE, we can prove

$$|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_2$**.** In this game, we change how $\mathsf{1KPFE.ct}^{(q)}$ is computed for all $q \in [Q_1+Q_2]$. Instead of setting it as $\mathsf{1KPFE.ct}^{(q)} = \mathsf{E}_{[x,\mathsf{K}]}(\mathsf{1KPFE.mpk}^{(q)})$, the game chooses fresh randomness $\mathsf{R}^{(q)}$ for 1KPFE.Enc and sets

$$\mathsf{1KPFE.ct}^{(q)} = \mathsf{1KPFE.Enc}(\mathsf{1KPFE.mpk}^{(q)}, x; \mathsf{R}^{(q)}).$$

Note that the previous game is equivalent to this game if we replace $\mathsf{R}^{(q)}$ with $\mathsf{PRF}(\mathsf{K}, \mathsf{1KPFE.mpk}^{(q)})$. As we will show in Lemma C.5, using the security of PRF, we can prove

$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_3$**.** In this game, we further change how $\mathsf{1KPFE.ct}^{(q)}$ is computed for all $q \in [Q_1+Q_2]$. In particular, we compute

$$\mathsf{1KPFE.ct}^{(q)} \leftarrow \mathsf{1KPFE.SimEnc}(\mathsf{1KPFE.mpk}^{(q)}, \mathcal{V}^{(q)}_{\mathsf{1KPFE}}, 1^{|x|})$$

where $\mathcal{V}^{(q)}_{\mathsf{1KPFE}} := \{C(x^{(q)}), C^{(q)}, \mathsf{1KPFE.sk}^{(q)}\}$. This removes all information about the challenge message from the entire distribution and thus corresponds to the ideal game $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{KPFE},\mathsf{Sim}}(1^\lambda)$ as per Definition 2.6. As we will show in Lemma C.6, using NA-SIM security of 1KPFE, we can prove

$$|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda).$$

To finish the proof of the theorem, it suffices to prove Lemmas C.4 to C.6.

Lemma C.4. *We have* $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \leq \mathrm{negl}(\lambda)$ *by the* AD-SIM *security of* CPFE.

Proof. Assume $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]|$ is non-negligle. We then describe an adversary B that breaks AD-SIM security of CPFE as follows. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$. B then sets $\mathsf{prm}' := (1^{\mathsf{inp}'}, 1^{\mathsf{size}'})$ using prm, sends prm' to CPFE challenger and obtains CPFE.mpk. It sends $\mathsf{mpk} := \mathsf{CPFE.mpk}$ to A. When A submits a key query for a circuit $C^{(q)}$ for some $q \in [Q_1]$, B does the following:

1. It samples $(\mathsf{1KPFE.mpk}^{(q)}, \mathsf{1KPFE.msk}^{(q)}) \leftarrow \mathsf{1KPFE.Setup}(1^\lambda, \mathsf{prm})$ and futher generates $\mathsf{1KPFE.sk}^{(q)} \leftarrow \mathsf{1KPFE.KeyGen}(\mathsf{1KPFE.msk}^{(q)}, C^{(q)})$.
2. It submits its own key query for $\mathsf{1KPFE.mpk}^{(q)}$ to CPFE challenger and receives $\mathsf{CPFE.sk}^{(q)}$.
3. It sends $\mathsf{sk}^{(q)} := (\mathsf{CPFE.sk}^{(q)}, \mathsf{1KPFE.sk}^{(q)})$ to A.

When A submits an encryption query $(x, 1^Q)$, B does the following:

1. It samples $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$ and constructs $\mathsf{E}_{[x,\mathsf{K}]}$ as per the real protocol.
2. It then submits the tuple $\left(\mathsf{E}_{[x,\mathsf{K}]}, 1^Q\right)$ to its CPFE challenger and obtains CPFE.ct. In particular, the CPFE challenger replies B with CPFE.ct, where

$$(\mathsf{CPFE.ct}, \mathsf{CPFE.st}) \leftarrow \begin{cases} \mathsf{CPFE.Enc}\left(\mathsf{CPFE.mpk}, \mathsf{E}_{[x,\mathsf{K}]}, 1^Q\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{CPFE,B}}\left(1^\lambda\right) \\ \mathsf{CPFE.SimEnc}\left(\mathsf{CPFE.mpk}, \mathcal{V}_{\mathsf{CPFE}}, 1^{\mathsf{size}'}, 1^Q\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CPFE,Sim}}\left(1^\lambda\right) \end{cases}.$$

   In the above, $\mathcal{V}_{\mathsf{CPFE}} := \{\mathsf{E}_{[x,\mathsf{K}]}(\mathsf{1KPFE.mpk}^{(q)}), \mathsf{1KPFE.mpk}^{(q)}, \mathsf{CPFE.sk}^{(q)}\}_{q\in[Q_1]}$.
3. It then sends the ciphertext CPFE.ct to A.

A then makes key queries. When A makes a q-th key query for $C^{(q)}$, B proceeds as follows.

1. It samples $(\mathsf{1KPFE.mpk}^{(q)}, \mathsf{1KPFE.msk}^{(q)}) \leftarrow \mathsf{1KPFE.Setup}(1^\lambda, \mathsf{prm})$ and futher generates $\mathsf{1KPFE.sk}^{(q)} \leftarrow \mathsf{1KPFE.KeyGen}(\mathsf{1KPFE.msk}^{(q)}, C^{(q)})$.
2. It submits its own key query for $\mathsf{1KPFE.mpk}^{(q)}$ to CPFE challenger and receives $\mathsf{CPFE.sk}^{(q)}$. In particular, the CPFE challenger replies B with $\mathsf{CPFE.sk}^{(q)}$, where

$$\mathsf{CPFE.sk}^{(q)} \leftarrow \begin{cases} \mathsf{CPFE.KeyGen}\left(\mathsf{CPFE.msk}, \mathsf{1KPFE.mpk}^{(q)}\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{CPFE,B}}\left(1^\lambda\right) \\ \mathsf{CPFE.SimKG}\left(\mathsf{CPFE.st}, \mathsf{CPFE.msk}, \mathcal{V}^{(q)}_{\mathsf{CPFE}}\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CPFE,Sim}}\left(1^\lambda\right) \end{cases}.$$

   In the above, $\mathcal{V}^{(q)}_{\mathsf{CPFE}} := \{\mathsf{E}_{[x,\mathsf{K}]}(\mathsf{1KPFE.mpk}^{(q)}), \mathsf{1KPFE.mpk}^{(q)}\}$.
3. It sends $\mathsf{sk}^{(q)} := (\mathsf{CPFE.sk}^{(q)}, \mathsf{1KPFE.sk}^{(q)})$ to A.

It is straightforward to see that B simulates $\mathbf{Game}_0$ for A if it is in the real world and $\mathbf{Game}_1$ if it is in the ideal world. Thus, B breaks the AD-SIM security of CPFE if A distinguishes between the two games. □

Lemma C.5. *We have* $|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda)$ *by the security of* PRF.

Proof. Assume $|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]|$ is non-negligle. We then describe an adversary B that breaks the PRF security. This is a straightforward reduction where B generates $\mathsf{CPFE.mpk}$ and $\mathsf{CPFE.msk}$ by itself. When B has to compute $\mathsf{1KPFE.ct}^{(q)}$, either when it answers to the challenge query or secret key query, it queries the PRF oracle on input $\mathsf{1KPFE.mpk}^{(q)}$ and gets $\mathsf{R}^{(q)}$ in response. Then, it computes $\mathsf{1KPFE.ct}^{(q)} = \mathsf{KPFE.Enc}(1.\mathsf{KPFE.mpk}^{(q)}, x; \mathsf{R}^{(q)})$ and proceeds as $\mathbf{Game}_2$. It is easy to see that B simulates $\mathbf{Game}_2$ when $\mathsf{R}^{(q)}$ is truly random. On the other hand, if $\mathsf{R}^{(q)} = \mathsf{PRF}(\mathsf{K}, \mathsf{1KPFE.mpk}^{(q)})$, where K is the PRF key chosen by the PRF oracle, it simulates $\mathbf{Game}_1$ because we have

$$\begin{aligned} \mathsf{1KPFE.ct}^{(q)} &= \mathsf{1KPFE.Enc}(\mathsf{1KPFE.mpk}^{(q)}, x; \mathsf{PRF}(\mathsf{K}, \mathsf{1KPFE.mpk}^{(q)})) \\ &= \mathsf{E}_{[x,\mathsf{K}]}(\mathsf{1KPFE.mpk}^{(q)}) \end{aligned}$$

in this case. Thus, B breaks the PRF security if A distinguishes between the two games. □

Lemma C.6. *We have* $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda)$ *by the multi-instance* 1-NA-SIM *security of* 1KPFE.

Before proving the security, we define multi-instance version of NA-SIM security for 1KPFE and observe that this is implied by the NA-SIM security as per Definition 2.7. The reason why we introduce this security notion is that it makes the description of the reduction for Lemma C.6 simpler.

Definition C.7 (Multi-Instance Security for 1NA-SIM)**.** Here, we define multi-instance security for 1NA-SIM. The security is defined by the real and ideal game similarly to 1NA-SIM security definition as per Definition 2.7, but there are multiple instances. We summarize the difference from the single-instance case in the following.

1. At the beginning of the game, the adversary A specifies the number of instances N along with prm.
2. Then, $(\mathsf{mpk}^{(i)}, \mathsf{msk}^{(i)}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{prm})$ is run for $i \in [N]$. Then, $\{\mathsf{mpk}^{(i)}\}_{i\in[N]}$ is returned to A.
3. After that, A can make a secret key query and an encryption query for each instance of the FE with the restriction that the secret key query should be made before the encryption query and both queries should be made only once for each instance.
4. In the real world, all the key queries and encryption queries are answered honestly. In the ideal world, it is answered by the simulator that computes $\mathsf{ct} \leftarrow \mathsf{SimEnc}(\mathsf{mpk}, (R(x^{(i)}, y^{(i)}), y^{(i)}, \mathsf{sk}^{(i)}))$, where, $x^{(i)}$ and $y^{(i)}$ are the encryption query and key query to the i-th instance respectively and $\mathsf{sk}^{(i)}$ is the corresponding secret key to $y^{(i)}$.

It is straightforward to see that the above definition is implied by the single-instance definition (Definition 2.7) by the standard hybrid argument.

Proof of Lemma C.6. Assume $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]|$ is non-negligle. We then describe an adversary B that breaks the multi-instance 1-NA-SIM security of 1KPFE as follows. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$. B then sends prm and $Q_{\max}$ to 1KPFE challenger, where $Q_{\max}$ is an upper-bound for Q that is specified by A for the challenge query. We can set $Q_{\max}$ to be the running time of A for example. The challenger then sends $\{\mathsf{1KPFE.mpk}^{(q)}\}_{q\in[Q_{\max}]}$ to B in response. B then samples $(\mathsf{CPFE.mpk}, \mathsf{CPFE.msk})$ by itself and sends $\mathsf{mpk} := \mathsf{CPFE.mpk}$ to A. When A submits a key query for a circuit $C^{(q)}$ for some $q \in [Q_1]$, B does the following:

1. It sends $C^{(q)}$ to the q-th instance of 1KPFE challenger to obtain $\mathsf{1KPFE.sk}^{(q)}$.
2. It then computes $\mathsf{CPFE.sk}^{(q)} \leftarrow \mathsf{CPFE.KeyGen}(\mathsf{CPFE.msk}, \mathsf{1KPFE.mpk}^{(q)})$.
3. Finally, it sends $\mathsf{sk}^{(q)} := (\mathsf{CPFE.sk}^{(q)}, \mathsf{1KPFE.sk}^{(q)})$ to A.

When A submits an encryption query $(x, 1^Q)$, B does the following:

1. For all $q \in [Q]$, it submits x to its q-th encryption oracle, upon which it computes

$$\mathsf{1KPFE.ct}^{(q)} \leftarrow \begin{cases} \mathsf{1KPFE.Enc}(\mathsf{mpk}^{(q)}, x), & \text{if } \mathsf{B} \text{ is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{1KPFE},\mathsf{B}}(1^\lambda) \\ \mathsf{1KPFE.SimEnc}(\mathsf{1KPFE.mpk}^{(q)}, \mathcal{V}^{(q)}_{\mathsf{1KPFE}}, 1^{|x|}), & \text{if } \mathsf{B} \text{ is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{1KPFE},\mathsf{1KPFE.Sim}}(1^\lambda) \end{cases}$$

   where $\mathcal{V}^{(q)}_{\mathsf{1KPFE}} := \{C^{(q)}(x), C^{(q)}, \mathsf{1KPFE.sk}^{(q)}\}$ and $\mathsf{1KPFE.Sim}$ refers to the simulator for multi-instance security of $\mathsf{1KPFE}$.

2. Finally, it obtains $(\mathsf{CPFE.ct}, \mathsf{CPFE.st}) \leftarrow \mathsf{CPFE.SimEnc}(\mathsf{CPFE.mpk}, \mathcal{V}_{\mathsf{CPFE}}, 1^{\mathsf{size}'}, 1^Q)$, where $\mathcal{V}_{\mathsf{CPFE}} := \{\mathsf{1KPFE.ct}^{(q)}, \mathsf{1KPFE.mpk}^{(q)}, \mathsf{CPFE.sk}^{(q)}\}_{q\in[Q_1]}$.

3. It returns $\mathsf{CPFE.ct}$ to A.

A then makes key queries. When A makes a q-th key query for $C^{(q)}$, B proceeds as follows.

1. It sends $C^{(q)}$ to the q-th instance of $\mathsf{1KPFE}$ challenger to obtain $\mathsf{1KPFE.sk}^{(q)}$.

2. It then submits x to its q-th encryption oracle, upon which it computes

$$\mathsf{1KPFE.ct}^{(q)} \leftarrow \begin{cases} \mathsf{1KPFE.Enc}(\mathsf{mpk}^{(q)}, x), & \text{if } \mathsf{B} \text{ is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{1KPFE},\mathsf{B}}(1^\lambda) \\ \mathsf{1KPFE.SimEnc}(\mathsf{1KPFE.mpk}^{(q)}, \mathcal{V}^{(q)}_{\mathsf{1KPFE}}, 1^{|x|}), & \text{if } \mathsf{B} \text{ is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{1KPFE},\mathsf{1KPFE.Sim}}(1^\lambda) \end{cases}$$

   where $\mathcal{V}^{(q)}_{\mathsf{1KPFE}} := \{C^{(q)}(x), C^{(q)}, \mathsf{1KPFE.sk}^{(q)}\}$.

3. It then runs $\mathsf{CPFE.sk}^{(q)} \leftarrow \mathsf{CPFE.SimKG}(\mathsf{CPFE.st}, \mathsf{CPFE.msk}, \mathsf{1KPFE.ct}^{(q)}, \mathsf{1KPFE.mpk}^{(q)})$

4. Finally, it sends $\mathsf{sk}^{(q)} := (\mathsf{CPFE.sk}^{(q)}, \mathsf{1KPFE.sk}^{(q)})$ to A.

It is straightforward to see that B simulates $\mathbf{Game}_2$ for A if it is in the real world and $\mathbf{Game}_3$ if it is in the ideal world (of the multi-instance $\mathsf{1NA\text{-}SIM}$ game for $\mathsf{1KPFE}$). Thus, B breaks the multi-instance $\mathsf{1NA\text{-}SIM}$ security of $\mathsf{1KPFE}$ if A distinguishes between the two games. □

□

D Missing Details from Section 5

Correctness. We prove the the correctness of our CPFE scheme in Theorem D.1 below.

Theorem D.1. *For any given* $\mathsf{inp}(\lambda), \mathsf{dep}(\lambda)$ *and* $\mathsf{out}(\lambda)$*, assume* RGC *supports the circuit class* $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$*. Further, assume* CPFE' *supports the circuit class* $\mathcal{C}_{\mathsf{inp},\mathsf{size}'}$*. Then, the* CPFE *scheme is correct.*

Proof. Consider any honestly generated ciphertext $\mathsf{ct} = (\widetilde{C}, \mathsf{CPFE}'.\mathsf{ct})$ and secret key $\mathsf{sk} = \mathsf{CPFE}'.\mathsf{sk}$, where the latter is associated with input $x \in \{0,1\}^{\mathsf{inp}}$. Then the correctness of CPFE and PRF dictates that

$$y = \mathsf{CPFE}'.\mathsf{Dec}(\mathsf{CPFE}'.\mathsf{sk}, \mathsf{CPFE}'.\mathsf{ct}) = \mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}(x) = \mathsf{RGC.Enc}(\mathsf{gsk}, x; \mathsf{PRF}(\mathsf{K}, x)),$$

since $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]} \in \mathcal{C}_{\mathsf{inp},\mathsf{size}'}$. Next, the correctness of RGC implies $z = \mathsf{RGC.Eval}(\widetilde{C}, y) = C(x)$. This completes the proof of the theorem. □

Efficiency. Here, we evaluate the efficiency of the construction in Theorem D.2 below. The running time of the key generation algorithm of our construction is "succinct" in the sense that its running time does not depend on the size of the circuits that can be encrypted. Instead, its running time only depends on input and output length and depth of the circuits.

Theorem D.2. *Let* $\mathsf{inp}(\lambda), \mathsf{dep}(\lambda)$ *and* $\mathsf{out}(\lambda)$ *be polynomials in* λ*. Further, let* RGC *be a succinct RGC scheme for circuit family* $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ *and* CPFE′ *be a CPFE scheme for circuit family* $\mathcal{C}_{\mathsf{inp},\mathsf{size}'}$*. Then, the CPFE scheme* $\mathsf{CPFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *is succinct and runs in polynomial time.*

Proof. By the efficiency properties of CPFE′, the running time of CPFE′.KeyGen depends on its circuit size, i.e. $\mathsf{size}' = |\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}|$, where $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}$ is defined in Figure 2. Further, by the succinctness of gsk and RGC.Enc, the size of $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}$ is independent of the size of underlying circuit $C \in \mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$, which is unbounded. Instead, $|\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}|$ grows only with $\mathsf{inp}(\lambda), \mathsf{dep}(\lambda)$ and $\mathsf{out}(\lambda)$. This implies that $\mathsf{size}'(\lambda) = \mathrm{poly}(\lambda, \mathsf{inp}(\lambda), \mathsf{dep}(\lambda), \mathsf{out}(\lambda))$ and $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}$ can be computed in time $\mathrm{poly}(\lambda, \mathsf{inp}(\lambda), \mathsf{dep}(\lambda), \mathsf{out}(\lambda))$. Given this, it is straightforward to bound the running time of all the algorithms by the efficiency of CPFE′ and RGC. In particular, the running time of the key generation algorithm is independent from the size of the circuits and thus the scheme is succinct. □

We also observe that the running time of the encryption algorithm is linear in Q, which is inherited from the same property of CPFE′.

Proof of Security. Theorem D.3 below asserts the security of CPFE.

Theorem D.3. *Assume* PRF *is a secure pseudorandom function. Further, assume* RGC *for the circuit family* $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ *is secure as per Definition 5.3 and* CPFE′ *for the circuit family* $\mathcal{C}_{\mathsf{inp},\mathsf{size}'}$ *is* AD-SIM*-secure as per Definition 2.6. Then,* CPFE *for the circuit family* $\mathcal{C}_{\mathsf{inp},\mathsf{dep},\mathsf{out}}$ *is* AD-SIM*-secure as per Definition 2.6.*

Proof. Since CPFE′ satisfies AD-SIM security, we have a simulator $\mathsf{CPFE}'.\mathsf{Sim} = (\mathsf{CPFE}'.\mathsf{SimEnc}, \mathsf{CPFE}'.\mathsf{SimKG})$ for CPFE′. Similarly, since RGC satisfies security as per Definition 5.3, we have a simulator RGC.Sim. We construct the simulator $\mathsf{Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$ for CPFE using these algorithms. The ideal world with simulator Sim for CPFE and adversary A proceeds as follows.

Ideal world: On input 1^λ, A outputs $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$, upon which the game runs $\mathsf{Setup}(1^\lambda, \mathsf{prm})$, gets $\mathsf{mpk} = \mathsf{CPFE}'.\mathsf{mpk}$, and sends it to A. For each $q \in [Q_1]$, A queries secret key for input $x^{(q)}$ to receive $\mathsf{sk}^{(q)} = \mathsf{CPFE}'.\mathsf{sk}^{(q)}$. It then outputs $(C, 1^Q)$.

Simulating the ciphertext after prechallenge queries. On input

$$\mathsf{mpk} = \mathsf{CPFE}'.\mathsf{mpk},\ \mathcal{V} = \left\{z^{(q)} := C\big(x^{(q)}\big), x^{(q)}, \mathsf{sk}^{(q)} = \mathsf{CPFE}'.\mathsf{sk}^{(q)}\right\}_{q\in[Q_1]}, 1^{|C|},\ 1^Q$$

SimEnc proceeds as follows.

1. Invoke RGC simulator to obtain $(\widetilde{C}, \mathsf{RGC.st}) \leftarrow \mathsf{RGC.Sim}(1^\lambda, \mathsf{prm}, 1^{|C|})$.
2. Further, for all $q \in [Q_1]$, obtain
$$c_{x^{(q)}} \leftarrow \mathsf{RGC.Sim}\big(1^\lambda, x^{(q)}, z^{(q)}, \mathsf{RGC.st}\big).$$
3. Next, it computes size' and invokes the CPFE′ simualtor to obtain
$$(\mathsf{CPFE}'.\mathsf{ct}, \mathsf{CPFE}'.\mathsf{st}) \leftarrow \mathsf{CPFE}'\mathsf{Sim}(\mathsf{CPFE}'.\mathsf{mpk}, \mathcal{V}_{\mathsf{CPFE}'}, 1^{\mathsf{size}'}, 1^Q),$$
where $\mathcal{V}_{\mathsf{CPFE}'} := \{c_{x^{(q)}}, x^{(q)}, \mathsf{CPFE}'.\mathsf{sk}^{(q)}\}_{q\in[Q_1]}$.

4. Output the ciphertext $\mathsf{ct} := (\widetilde{C}, \mathsf{CPFE}'.\mathsf{ct})$ and internal state $\mathsf{st} = (\mathsf{RGC.st}, \mathsf{CPFE}'.\mathsf{st})$.

After giving the ciphertext ct, A makes secret key queries. The queries are answered by the simulator.

Simulating the secret key after the challenge query. For q-th query with $q \in [Q_1 + 1, Q_1 + Q_2]$, SimKG is given

$$\mathsf{st} = (\mathsf{RGC.st}, \mathsf{CPFE}'.\mathsf{st}),\ \mathsf{msk} = \mathsf{CPFE}'.\mathsf{msk},\ z^{(q)} := C\big(x^{(q)}\big),\ x^{(q)}$$

and proceeds as follows.

1. Obtain $c_{x^{(q)}} \leftarrow \mathsf{RGC.Sim}\left(1^\lambda, x^{(q)}, z^{(q)}, \mathsf{RGC.st}\right)$.
2. Obtain $\mathsf{CPFE}'.\mathsf{sk}^{(q)} \leftarrow \mathsf{CPFE}'.\mathsf{SimKG}(\mathsf{CPFE}'.\mathsf{st}, \mathsf{CPFE}'.\mathsf{msk}, c_{x^{(q)}}, x^{(q)})$.
3. Return $\mathsf{sk}^{(q)} := \mathsf{CPFE}'.\mathsf{sk}^{(q)}$.

The games. To prove the security, we consider the following sequence of games, where $\mathbf{Game}_0$ corresponds to the real game and $\mathbf{Game}_3$ corresponds to the ideal world. For $i \in [0, 3]$, let $\mathcal{E}_i$ denote the event that A outputs 1 in $\mathbf{Game}_i$. In particular, we prove that $|\Pr[\mathcal{E}_i] - \Pr[\mathcal{E}_{i+1}]| \leq \mathrm{negl}(\lambda)$ for all $i \in [0, 2]$, which implies $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda)$.

$\mathbf{Game}_0$. This is the real game $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{CPFE},\mathsf{A}}\left(1^\lambda\right)$ as per Definition 2.6.

$\mathbf{Game}_1$. In this game, we replace the encryption algorithm and key generation algorithms of CPFE' with its simulator $\mathsf{CPFE}'.\mathsf{SimEnc}$ and $\mathsf{CPFE}'.\mathsf{SimKG}$ respectively. In particular, $\mathsf{CPFE}'.\mathsf{ct}$ is generated as follows.

1. For all $q \in [Q_1]$, compute an encoding $c_{x^{(q)}} = \mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}\left(x^{(q)}\right)$.
2. Further, it computes size' and run

$$(\mathsf{CPFE}'.\mathsf{ct}, \mathsf{CPFE}'.\mathsf{st}) \leftarrow \mathsf{CPFE}'.\mathsf{SimEnc}\big(\mathsf{CPFE}'.\mathsf{mpk}, \mathcal{V}_{\mathsf{CPFE}'}, 1^{\mathsf{size}'}, 1^Q\big)$$

where $\mathcal{V}_{\mathsf{CPFE}'} := \{c_{x^{(q)}}, x^{(q)}, \mathsf{CPFE}'.\mathsf{sk}^{(q)}\}_{q \in [Q_1]}$.

Furthermore, we change the way how $\mathsf{CPFE.sk}^{(q)}$ for $q \in [Q_1 + 1, Q_1 + Q_2]$ is generated as follows.

1. Compute an encoding $c_{x^{(q)}} \leftarrow \mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}\left(x^{(q)}\right)$.
2. Run $\mathsf{CPFE.sk}^{(q)} \leftarrow \mathsf{CPFE}'.\mathsf{SimKG}\left(\mathsf{CPFE}'.\mathsf{st}, \mathsf{CPFE}'.\mathsf{msk}, c_{x^{(q)}}, x^{(q)}\right)$.

As we will show in Lemma D.4, using AD-SIM security of CPFE', we can prove

$$|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_2$. In this game, we change how the RGC encoding $c_{x^{(q)}}$ is computed for all $q \in [Q_1 + Q_2]$. Instead of setting it as $c_{x^{(q)}} = \mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}(x^{(q)})$, the game chooses fresh randomness $\mathsf{R}^{(q)}$ for RGC.Enc and sets

$$c_{x^{(q)}} = \mathsf{RGC.Enc}\big(\mathsf{gsk}, x^{(q)}; \mathsf{R}^{(q)}\big).$$

Note that the previous game is equivalent to this game if we replace $\mathsf{R}^{(q)}$ with $\mathsf{PRF}(\mathsf{K}, x^{(q)})$. As we will show in Lemma D.5, using the security of PRF, we can prove

$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_3$. In this game, we further change how the garbled circuit for C in the challenge message and the encodings $\{c_{x^{(q)}}\}_{q\in[Q_1+Q_2]}$ are computed. In particular, we compute them as follows.

$$\begin{aligned}\left(\widetilde{C}, \mathsf{RGC.st}\right) &\leftarrow \mathsf{RGC.Sim}(1^\lambda, \mathsf{prm}, 1^{|C|}),\\ \left\{c_{x^{(q)}}\right. &\leftarrow \left.\mathsf{RGC.SimEnc}\left(1^\lambda, x^{(q)}, z^{(q)}, \mathsf{RGC.st}\right)\right\}_{q\in[Q_1+Q_2]}.\end{aligned}$$

This removes all information about the challenge message from the entire distribution and thus corresponds to the ideal game $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CPFE},\mathsf{Sim}}\left(1^\lambda\right)$ as per Definition 2.6. As we will show in Lemma D.6, using RGC security from Definition 5.3, we can prove

$$|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda).$$

To finish the proof of the theorem, it suffices to prove lemmas D.4 to D.6.

Lemma D.4. *We have* $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]| \leq \mathrm{negl}(\lambda)$ *by the* AD-SIM *security of* CPFE.

Proof. Assume $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_1]|$ is non-negligle. We then describe an adversary B that breaks AD-SIM security of CPFE' as follows. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$. B then sets $\mathsf{prm}' := (1^{\mathsf{inp}}, 1^{\mathsf{size}'})$ using prm, sends prm' to CPFE' challenger and obtains $\mathsf{CPFE}'.\mathsf{mpk}$. It sends $\mathsf{mpk} := \mathsf{CPFE}'.\mathsf{mpk}$ to A.

When A submits a key query for an input $x^{(q)} \in \{0,1\}^{\mathsf{inp}}$ for some $q \in [Q_1]$, B relays it to the CPFE' challenger and receives $\mathsf{CPFE}'.\mathsf{sk}^{(q)}$. It then sends $\mathsf{sk}^{(q)} := \mathsf{CPFE}'.\mathsf{sk}^{(q)}$ to A. When A submits an encryption query $(C, 1^Q)$, B does the following:

1. It samples $\mathsf{K} \leftarrow \mathsf{PRF.Setup}(1^\lambda)$, computes $(\mathsf{gsk}, \widetilde{C}) \leftarrow \mathsf{RGC.Garble}(1^\lambda, \mathsf{prm}, C)$ and constructs $\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}$ as per the real protocol.
2. Then it sends the tuple $\left(\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}, 1^Q\right)$ to the CPFE' challenger and obtains $\mathsf{CPFE}'.\mathsf{ct}$. In particular, the CPFE' challenger replies B with $\mathsf{CPFE}'.\mathsf{ct}$, where

$$(\mathsf{CPFE}'.\mathsf{ct}, \mathsf{CPFE}'.\mathsf{st}) \leftarrow \begin{cases}\mathsf{CPFE}'.\mathsf{Enc}\left(\mathsf{CPFE}'.\mathsf{mpk}, \mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}, 1^Q\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{CPFE}',\mathsf{B}}\left(1^\lambda\right)\\ \mathsf{CPFE}'.\mathsf{SimEnc}\left(\mathsf{CPFE}'.\mathsf{mpk}, \mathcal{V}_{\mathsf{CPFE}'}, 1^{\mathsf{size}'}, 1^Q\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CPFE}',\mathsf{Sim}}\left(1^\lambda\right)\end{cases}.$$

In the above, $\mathcal{V}_{\mathsf{CPFE}'} := \left\{\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}(x^{(q)}), x^{(q)}, \mathsf{CPFE}'.\mathsf{sk}^{(q)}\right\}_{q\in[Q_1]}$.

3. It then sends the ciphertext $\mathsf{ct} := (\widetilde{C}, \mathsf{CPFE}'.\mathsf{ct})$ to A.

A then makes post-challenge key queries. When A makes a q-th key query for $x^{(q)}$, B proceeds as follows.

1. It submits $x^{(q)}$ to CPFE' challenger and receives $\mathsf{CPFE}'.\mathsf{sk}^{(q)}$. In particular, the CPFE' challenger replies B with $\mathsf{CPFE}'.\mathsf{sk}^{(q)}$, where

$$\mathsf{CPFE}'.\mathsf{sk}^{(q)} \leftarrow \begin{cases}\mathsf{CPFE}'.\mathsf{KeyGen}\left(\mathsf{CPFE}'.\mathsf{msk}, x^{(q)}\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{CPFE}',\mathsf{B}}\left(1^\lambda\right)\\ \mathsf{CPFE}'.\mathsf{SimKG}\left(\mathsf{CPFE}'.\mathsf{st}, \mathsf{CPFE}'.\mathsf{msk}, \mathcal{V}^{(q)}_{\mathsf{CPFE}'}\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{CPFE}',\mathsf{Sim}}\left(1^\lambda\right)\end{cases}.$$

In the above, $\mathcal{V}^{(q)}_{\mathsf{CPFE}'} := \{\mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}(x^{(q)}), x^{(q)}\}$.

2. It sends $\mathsf{sk}^{(q)} := \mathsf{CPFE}'.\mathsf{sk}^{(q)}$ to A.

It is straightforward to see that B simulates $\mathbf{Game}_0$ for A if it is in the real world and $\mathbf{Game}_1$ if it is in the ideal world. Thus, B breaks the AD-SIM security of CPFE if A distinguishes between the two games. □

Lemma D.5. *We have* $|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda)$ *by the security of* PRF.

Proof. Assume $|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]|$ is non-negligle. We then describe an adversary B that breaks the PRF security. This is a straightforward reduction where B generates $\mathsf{CPFE}'.\mathsf{mpk}$ and $\mathsf{CPFE}'.\mathsf{msk}$ by itself. When A asks the challenge query $(C, 1^Q)$, B first computes $(\widetilde{C}, \mathsf{gsk}) \leftarrow \mathsf{RGC.Garble}(1^\lambda, \mathsf{prm}, C)$. When B has to compute the encodings $c_{x^{(q)}}$, during the challenge phase or to answer the post-challenge secret keys, it queries the PRF oracle on input $x^{(q)}$ and gets $\mathsf{R}^{(q)}$ which it uses to compute $c_{x^{(q)}} \leftarrow \mathsf{RGC.Enc}(\mathsf{gsk}, x; \mathsf{R}^{(q)})$. Then it proceeds as $\mathbf{Game}_2$. It is easy to see that B simulates $\mathbf{Game}_2$ when $\mathsf{R}^{(q)}$ is truly random. On the other hand, if $\mathsf{R}^{(q)} = \mathsf{PRF}(\mathsf{K}, x^{(q)})$, where K is the PRF key chosen by the PRF oracle, it simulates $\mathbf{Game}_1$ because we have $c_{x^{(q)}} = \mathsf{RGC.Enc}(\mathsf{gsk}, x^{(q)}; \mathsf{PRF}(\mathsf{K}, x^{(q)})) = \mathsf{E}_{[\mathsf{gsk},\mathsf{K}]}(x^{(q)})$ in this case. Thus, B breaks the PRF security if A distinguishes between the two games. □

Lemma D.6. *We have* $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda)$ *by security of* RGC.

Proof. Assume $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]|$ is non-negligle. We then describe an adversary B that breaks the security of RGC as follows. On input 1^λ, B first runs A on input 1^λ to obtain $\mathsf{prm} = (1^{\mathsf{inp}}, 1^{\mathsf{dep}}, 1^{\mathsf{out}})$. B then computes size' using prm and samples $(\mathsf{CPFE}'.\mathsf{mpk}, \mathsf{CPFE}'.\mathsf{msk})$ by itself and sends $\mathsf{mpk} := \mathsf{CPFE}'.\mathsf{mpk}$ to A. B answers the pre-challenge secret keys same as in $\mathbf{Game}_2$.

When A submits an encryption query $(C, 1^Q)$, B does the following:

1. It submits the tuple $(1^\lambda, \mathsf{prm}, C)$ to the RGC challenger, upon which it computes
$$(\widetilde{C}, \mathsf{st}') \leftarrow \begin{cases} \mathsf{RGC.Garble}(1^\lambda, \mathsf{prm}, C) \text{ with } \mathsf{st}' = \mathsf{gsk}, & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{RGC},\mathsf{B}}(1^\lambda) \\ \mathsf{RGC.Sim}(1^\lambda, \mathsf{prm}, 1^{|C|}) \text{ with } \mathsf{st}' = \mathsf{RGC.st}, & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{RGC},\mathsf{B},\mathsf{RGC.Sim}}(1^\lambda) \end{cases}$$
and returns $\widetilde{C}$ to B.
2. For all $q \in [Q_1]$, it submits the $x^{(q)}$ to the RGC challenger, upon which it computes
$$c_{x^{(q)}} \leftarrow \begin{cases} \mathsf{RGC.Enc}(\mathsf{gsk}, x), & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{RGC},\mathsf{B}}(1^\lambda) \\ \mathsf{RGC.Sim}(1^\lambda, x^{(q)}, C(x^{(q)}), \mathsf{RGC.st}), & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{RGC},\mathsf{B},\mathsf{RGC.Sim}}(1^\lambda) \end{cases}$$
and returns $c_{x^{(q)}}$ to B.
3. Finally, it gets $(\mathsf{CPFE}'.\mathsf{ct}, \mathsf{CPFE}'.\mathsf{st}) \leftarrow \mathsf{CPFE}'.\mathsf{SimEnc}(\mathsf{CPFE}'.\mathsf{mpk}, \mathcal{V}_{\mathsf{CPFE}'}, 1^{\mathsf{size}'}, 1^Q)$, where $\mathcal{V}_{\mathsf{CPFE}'} := \{c_{x^{(q)}}, x^{(q)}, \mathsf{CPFE}'.\mathsf{sk}^{(q)}\}_{q\in[Q_1]}$.
4. It returns $\mathsf{ct} := (\widetilde{C}, \mathsf{CPFE}'.\mathsf{ct})$ to A.

A then makes post-challenge key queries. When A makes a q-th key query for $x^{(q)}$, B proceeds as follows.

1. It sends $x^{(q)}$ to the RGC challenger, upon which it computes
$$c_{x^{(q)}} \leftarrow \begin{cases} \mathsf{RGC.Enc}(\mathsf{gsk}, x), & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{RGC},\mathsf{B}}(1^\lambda) \\ \mathsf{RGC.Sim}(1^\lambda, x^{(q)}, C(x^{(q)}), \mathsf{st}_{\mathsf{RGC.Sim}}), & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{RGC},\mathsf{B},\mathsf{RGC.Sim}}(1^\lambda) \end{cases}$$
and returns $c_{x^{(q)}}$ to B.
2. It then runs $\mathsf{CPFE}'.\mathsf{sk}^{(q)} \leftarrow \mathsf{CPFE}'.\mathsf{SimKG}(\mathsf{CPFE}'.\mathsf{st}, \mathsf{CPFE}'.\mathsf{msk}, c_{x^{(q)}}, x^{(q)})$
3. Finally, it sends $\mathsf{sk}^{(q)} := \mathsf{CPFE}'.\mathsf{sk}^{(q)}$ to A.

It is straightforward to see that B simulates $\mathbf{Game}_2$ for A if it is in the real world and $\mathbf{Game}_3$ if it is in the ideal world of the security game of RGC. Thus, B breaks the security of RGC if A distinguishes between the two games. □

□

E Missing Details from Section 6

Correctness. Theorem E.1 below proves the correctness of our BFE scheme from Section 6.1.

Theorem E.1. BFE *defined in Section 6.1 is correct.*

Proof. We observe that $\mathsf{lab}'_{i,j} = \mathsf{lab}_{i,j,\mathsf{FE.mpk}_{i,j}}$ holds for all $\mathsf{lab}'_{i,j}$ recovered in Step 2b of the decryption algorithm by the correctness of IBE, since the ciphertext and secret key are both generated with respect to the identity $(i, j, \mathsf{FE.mpk}_{i,j})$. Then, by the correctness of GC, we have $c_i = \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, f(x)_i, 1^Q; r_i)$ for all $i \in \mathcal{S}(|x|) \cap \mathcal{T}(|y|)$ recovered in Step 2c of the decryption algorithm. Finally, by the correctness of FE, we have $z_i = R_i(f(x)_i, g(y)_i)$ for all z_i recovered in Step 2d of the decryption algorithm as desired. □

Efficiency. We check that all algorithms in BFE run in polynomial time in its input length. This is obvious for Setup. As for the running time of KeyGen, one can observe that the running time can be bounded by a polynomial in λ, the size of $\mathcal{T}(|y|)$, and $\max \mathcal{T}(|y|)$ by the efficiency of FE and IBE. These values are bounded by a polynomial in $|y|$ by the property of $\mathcal{T}$. Thus, the key generation algorithm runs in polynomial time in its input length. Similarly, the running time of Enc can be bounded by a polynomial in λ, Q, the size of $\mathcal{S}(|x|)$, and $\max \mathcal{S}(|x|)$ by the efficiency of FE, GC, and IBE. By the property of $\mathcal{S}$, we can see that the algorithm runs in polynomial time in its input length. Finally, having observed above, it is straightforward to see that the decryption algorithm runs in polynomial time in its input length.

Proof of Security. Theorem E.2 below asserts the security of BFE scheme from Section 6.1.

Theorem E.2. *Suppose* PRF *is a secure pseudorandom function and* GC *is secure garbled circuit scheme, and* IBE *is* IND-CPA *secure identity-based encryption scheme. Then, assuming* FE *is* NA-SIM *secure, so is* BFE. *Furthermore, if* FE *is* AD-SIM *secure, so is* BFE.

Proof. Here, we prove the statement for the case of AD-SIM. The proof for NA-SIM is similar and simpler. Since FE satisfies AD-SIM security, we have a simulator $\mathsf{FE.Sim} = (\mathsf{FE.SimEnc}, \mathsf{FE.SimKG})$ for FE. We construct the simulator $\mathsf{BFE.Sim} = (\mathsf{SimEnc}, \mathsf{SimKG})$ for BFE using these algorithms. The ideal world with simulator Sim for BFE and adversary A proceeds as follows.

Ideal world: On input 1^λ, the game runs $\mathsf{Setup}(1^\lambda)$, gets $\mathsf{mpk} = \mathsf{IBE.mpk}$, and sends it to A. For each $q \in [Q_1]$, A queries secret key for $y^{(q)}$ to receive

$$\mathsf{sk}^{(q)} = \left(\mathcal{T}(|y^{(q)}|),\ \left\{ \mathsf{FE.sk}_i^{(q)}, \left\{ \mathsf{IBE.sk}_{i,j}^{(q)} \right\}_{j \in [\ell_i]} \right\}_{i \in \mathcal{T}(|y^{(q)}|)} \right).$$

A then outputs $(x, 1^Q)$ as its target.

Simulating the ciphertext after prechallenge queries. SimEnc takes as input

$$\mathsf{mpk} = \mathsf{IBE.mpk},\ \mathcal{V},\ 1^Q,\ 1^{|x|}$$

where

$$\mathcal{V} = \left\{ R^{\mathsf{bndl}}(x, y^{(q)}) = \left\{ R_i(f(x)_i, g(y^{(q)})_i) \right\}_{i \in \mathcal{S}(|x|) \cap \mathcal{T}(|y^{(q)}|)},\ y^{(q)},\ \mathsf{sk}^{(q)} \right\}_{q \in [Q_1]}$$

and simulates the ciphertext. Before giving the description of the simulator, we prepare some notations. We define $\Gamma_i \subseteq [Q_1]$ as

$$\Gamma_i := \left\{ q \in [Q_1] : i \in \mathcal{T}(|y^{(q)}|) \right\}.$$

Intuitively, Γ_i is the set of indices q of secret key queries before the encryption query that uses the i-th instance of FE. The simulator SimEnc proceeds as follows.

1. Compute the set $\mathcal{S}(|x|)$. This is possible using $1^{|x|}$ without knowing x.
2. For all $i \in \mathcal{S}(|x|)$, compute $\mathsf{FE.mpk}_i$ as follows. There are two cases to consider:
   - If $\Gamma_i \neq \emptyset$, retrieve $\mathsf{FE.mpk}_i$ from $\mathsf{FE.sk}_i^{(q)}$ for some $q \in \Gamma_i$.
   - If $\Gamma_i = \emptyset$, run
   $$(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i)$$
   using fresh randomness. Keep the key pair $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ for future use.
3. For all $i \in \mathcal{S}(|x|)$, do the following.
   (a) Run
   $$(\mathsf{FE.ct}_i, \mathsf{FE.st}_i) \leftarrow \mathsf{FE.SimEnc}\left(\mathsf{FE.mpk}_i, \mathcal{V}_{\mathsf{FE},i}, 1^Q\right)$$
   where
   $$\mathcal{V}_{\mathsf{FE},i} = \left\{ R_i\left(f(x)_i, g(y^{(q)})_i\right),\ g(y^{(q)})_i,\ \mathsf{FE.sk}^{(q)} \right\}_{q\in\Gamma_i}.$$
   Note that $\mathcal{V}_{\mathsf{FE},i}$ can be an empty set.
   (b) Compute the length ℓ_i of $\mathsf{FE.mpk}_i$ and the size $|\mathsf{E}_i|$ of the circuit $\mathsf{E}_i = \mathsf{FE.Enc}(\cdot, f(x)_i, 1^Q; r_i)$. This is possible by our assumption on f since the length of $f(x)_i$ is computable from $|x|$ without knowing x. Then run
   $$\{\mathsf{lab}_{i,j}\}_{j\in[\ell_i]} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_i}, 1^{|\mathsf{E}_i|}, \mathsf{FE.ct}_i).$$
   (c) For all $j \in [\ell_i]$ and $b \in \{0,1\}$, compute
   $$\mathsf{IBE.ct}_{i,j,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,b), \mathsf{lab}_{i,j}\right).$$
   Note that here, we encrypt the same label for both $b = 0$ and $b = 1$.
4. Output the ciphertext
   $$\mathsf{ct} = \left(\mathcal{S}(|x|),\ \{\mathsf{IBE.ct}_{i,j,b}\}_{i\in\mathcal{S}(|x|), j\in[\ell_i], b\in\{0,1\}}\right)$$
   and keep its internal state
   $$\mathsf{st} := \left(\{\mathsf{FE.msk}_i\}_{i\in\mathcal{S}(|x|):\Gamma_i=\emptyset}, \{\mathsf{FE.st}_i\}_{i\in\mathcal{S}(|x|)}\right). \tag{E.1}$$

Simulating the secret key after the encryption query. Here, we describe SimKG. It is given

$$\mathsf{st}, \quad \mathsf{msk} = (\mathsf{IBE.msk}, \mathsf{K}), \quad R^{\mathsf{bndl}}(x, y^{(q)}) = \left\{R_i(f(x)_i, g(y^{(q)})_i)\right\}_{i\in\mathcal{S}(|x|)\cap\mathcal{T}(|y^{(q)}|)}$$

where $y^{(q)}$ is the q-th query with $q \in [Q_1+1, Q_1+Q_2]$ that the adversary makes. Then, it proceeds as follows.

1. Parse its internal state st as Eq. (E.1).
2. Compute the set $\mathcal{T}(|y^{(q)}|)$.
3. For all $i \in \mathcal{T}(|y^{(q)}|)$, do the following. There are two cases:
   - If $i \in \mathcal{S}(|x|) \wedge \Gamma_i = \emptyset$, recover $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ from st.

– Otherwise, compute

$$(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i; \mathsf{R}_i),$$

where $\mathsf{R}_i = \mathsf{PRF.Eval}(\mathsf{K}, i)$.

4. For $i \in \mathcal{T}(|y^{(q)}|)$, compute $\mathsf{FE.sk}_i^{(q)}$ as follows. There are two cases:
   – If $i \in \mathcal{S}(x)$,

$$\mathsf{FE.sk}_i^{(q)} \leftarrow \mathsf{FE.Sim}\left(\mathsf{FE.st}_i, \mathsf{FE.msk}_i, R_i\left(f(x)_i, g(y^{(q)})_i\right), g(y^{(q)})_i\right).$$

   – If $i \notin \mathcal{S}(x)$, run

$$\mathsf{FE.sk}_i^{(q)} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, g(y^{(q)})_i).$$

5. For all $i \in \mathcal{T}(|y^{(q)}|)$ and $j \in \ell_i$, generate a secret key as

$$\mathsf{IBE.sk}_{i,j} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{IBE.msk}, (i, j, \mathsf{FE.mpk}_{i,j})).$$

6. Return $\mathsf{sk}^{(q)} = \left(\mathcal{T}(|y^{(q)}|), \left\{\mathsf{FE.sk}_i^{(q)}, \{\mathsf{IBE.sk}_{i,j}\}_{j\in[\ell_i]}\right\}_{i\in\mathcal{T}(|y^{(q)}|)}\right)$.

The games. To prove the security, we consider the following sequence of games, where $\mathbf{Game}_0$ corresponds to the real game and $\mathbf{Game}_8$ corresponds to the ideal world. For i, let $\mathcal{E}_i$ denote the event that A outputs 1 in $\mathbf{Game}_i$. The theorem can be proven by showing $|\Pr[\mathcal{E}_0] - \Pr[\mathcal{E}_8]| \leq \mathrm{negl}(\lambda)$.

$\mathbf{Game}_0$: This is the real game $\mathsf{Exp}^{\mathsf{real}}_{\mathsf{BFE,A}}\left(1^\lambda\right)$ for AD-SIM security defined as per Definition 2.6.

$\mathbf{Game}_1$: In this game, we change the game so that it computes all the key pairs $\{(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)\}_i$ that are necessary in advance instead of generating them on the fly when they are needed. Namely, it sets $i_{\max}$ sufficiently large so that it always satisfies

$$i_{\max} \geq \max\left\{i : i \in \mathcal{S}(|x|) \cup \left(\bigcup_{q\in[Q_1+Q_2]} \mathcal{T}(|y^{(q)}|)\right)\right\} \tag{E.2}$$

and computes $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i; \mathsf{R}_i)$ where $\mathsf{R}_i = \mathsf{PRF.Eval}(\mathsf{K}, i)$ for all $i \in [i_{\max}]$ at the beginning of the game. These keys will be used for answering the secret key queries later in the game. Note that by setting $i_{\max}$ to be sufficiently large polynomial, we can ensure that it satisfies Equation (E.2), since $|x|$ and $|y^{(q)}|$ are bounded by the running time of A and each element in $\mathcal{S}(|x|)$ and $\mathcal{T}(|y|)$ is bounded by a polynomial in $|x|$ and $|y|$. This change is clearly conceptual and thus we have

$$\Pr[\mathcal{E}_0] = \Pr[\mathcal{E}_1].$$

$\mathbf{Game}_2$. In this game, we replace $\mathsf{R}_i = \mathsf{PRF.Eval}(\mathsf{K}, i)$ with true randomness. Namely, the game generates

$$(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i)$$

for $i \in [i_{\max}]$ using true randomness instead of R_i at the beginning of the game. As we show in Lemma E.3, using the security of PRF, we can prove

$$|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_3$: In this game, we change the way the encryption query is answered. In particular, $\mathsf{IBE.ct}_{i,j,b}$ is generated as

$$\mathsf{IBE.ct}_{i,j,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,b), \mathsf{lab}_{i,j,\mathsf{FE.mpk}_{i,j}}\right)$$

for all $i \in \mathcal{S}(|x|)$, $j \in [\ell_i]$, and $b \in \{0,1\}$ in this game, where $\mathsf{FE.mpk}_{i,j}$ is the j-th bit of $\mathsf{FE.mpk}_i$. Note that we encrypt the same label for $b = 0$ and $b = 1$. As we show in Lemma E.4, using IND-CPA security of IBE, we can prove

$$|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_4$: In this game, we change the way the encryption query is answered. In particular, it generates $\{\mathsf{lab}_{i,j,\mathsf{FE.mpk}_{i,j}}\}_{i\in\mathcal{S}(|x|),j\in[\ell_i]}$ as follows. It first runs

$$\mathsf{FE.ct}_i \leftarrow \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, f(x)_i, 1^Q),$$

and computes

$$\{\mathsf{lab}_{i,j}\}_{j\in[\ell_i]} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_i}, 1^{|\mathsf{E}_i|}, \mathsf{FE.ct}_i)$$

for all $i \in \mathcal{S}(|x|)$. Then, it sets

$$\mathsf{lab}_{i,j,\mathsf{FE.mpk}_i} := \mathsf{lab}_{i,j}$$

for all $j \in [\ell_i]$. The labels obtained above are then encrypted by IBE as specified in the previous game. As we show in Lemma E.5, using security of GC, we can prove

$$|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_5$: In this game, we change the way both the encryption query and secret key queries are answered. In particular, when the game answers the encryption query, it generates $\mathsf{FE.ct}_i$ as

$$(\mathsf{FE.ct}_i, \mathsf{FE.st}_i) \leftarrow \mathsf{FE.SimEnc}\left(\mathsf{FE.mpk}_i, \mathcal{V}_{\mathsf{FE},i}, 1^Q\right)$$

where

$$\mathcal{V}_{\mathsf{FE},i} = \left\{ R_i\left(f(x)_i, g(y^{(q)})_i\right),\ g(y^{(q)})_i,\ \mathsf{FE.sk}^{(q)} \right\}_{q\in\Gamma_i}$$

for all $i \in \mathcal{S}(|x|)$. Then, $\{\mathsf{FE.ct}_i\}_{i\in\mathcal{S}(|x|)}$ obtained above is used to generate the ciphertext as specified in the previous games. In addition, it answers the secret key query after the encryption query as follows. When the adversary makes the q-th secret key query $y^{(q)}$ with $q \in [Q_1+1, Q_1+Q_2]$, it computes $\mathsf{FE.sk}_i^{(q)}$ for $i \in \mathcal{T}(|y^{(q)}|)$ as follows. There are two cases:

- If $i \in \mathcal{S}(x)$, it computes $\mathsf{FE.sk}_i^{(q)}$ as

$$\mathsf{FE.sk}_i^{(q)} \leftarrow \mathsf{FE.Sim}\left(\mathsf{FE.st}_i, \mathsf{FE.msk}_i, R_i\left(f(x)_i, g(y^{(q)})_i\right), g(y^{(q)})_i\right).$$

- If $i \notin \mathcal{S}(x)$, it computes $\mathsf{FE.sk}_i^{(q)}$ as

$$\mathsf{FE.sk}_i^{(q)} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, g(y^{(q)})_i).$$

As we show in Lemma E.6, using AD-SIM security of FE, we can prove

$$|\Pr[\mathcal{E}_4] - \Pr[\mathcal{E}_5]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_6$: In this game, we stop generating all the key pairs $\{(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)\}_{i\in[i_{\max}]}$ at the beginning of the game. Instead, each key pair is generated when it is necessary to answer a secret key or an encryption query. Note that we use the same key pair for each instance i throughout the game, once it is generated. This is clearly a conceptual change and thus we have

$$\Pr[\mathcal{E}_5] = \Pr[\mathcal{E}_6].$$

$\mathbf{Game}_7$: In this game, we change the way we generate $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ when it is required for answering the encryption query or secret key queries. There are three cases to consider:

- If the key pair $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ has been generated before, the game uses the same key pair.
- If the key pair $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ is necessary for answering a key query and it has not been generated before, the game generates it using a PRF key K as $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i; \mathsf{R}_i)$ where $\mathsf{R}_i = \mathsf{PRF.Eval}(\mathsf{K}, i)$.
- If the key pair $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ is necessary for answering an encryption query and the pair has not been generated before (i.e., if $i \in \mathcal{S}(|x|)$ and $\Gamma_i = \emptyset$), the game samples fresh pair as $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i)$ by using true randomness.

As we show in Lemma E.7, using the security of PRF, we can prove

$$|\Pr[\mathcal{E}_6] - \Pr[\mathcal{E}_7]| \leq \mathrm{negl}(\lambda).$$

$\mathbf{Game}_8$: This is the real game $\mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE,Sim}}(1^\lambda)$ for AD-SIM security defined as per Theorem 2.6. By inspection, it can easily be seen that this game is the same as the previous game. We therefore have

$$\Pr[\mathcal{E}_7] = \Pr[\mathcal{E}_8].$$

To finish the proof of the theorem, it suffices to prove Lemma E.3 to E.7. □

Lemma E.3. *We have* $|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]| \leq \mathrm{negl}(\lambda)$ *by the security of* PRF.

Proof. Assume $|\Pr[\mathcal{E}_1] - \Pr[\mathcal{E}_2]|$ is non-negligle. We then describe an adversary B that breaks the PRF security. At the beginning of the game, B makes a PRF query for i and receives R_i for all $i \in [i_{\max}]$, where R_i is set as $\mathsf{PRF.Eval}(\mathsf{K}, i)$ or random. Then, B generates $\mathsf{FE.mpk}_i$ and $\mathsf{FE.msk}_i$ as $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i; \mathsf{R}_i)$ for $i \in [i_{\max}]$ and $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$. Then, the rest of the game is simulated using $\mathsf{IBE.msk}$ and $\{\mathsf{FE.msk}_i\}_{i\in[i_{\max}]}$. It is easy to see that B simulates $\mathbf{Game}_1$ if $\mathcal{B}$ has access to a PRF function and $\mathbf{Game}_2$ otherwise. Thus, B breaks the PRF security if A distinguishes between the two games. □

Lemma E.4. *We have* $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]| \leq \mathrm{negl}(\lambda)$ *by the security of* IBE.

Proof. Assume $|\Pr[\mathcal{E}_2] - \Pr[\mathcal{E}_3]|$ is non-negligle. We then describe an adversary B that breaks multi-challenge IND-CPA security of IBE as follows. Note that as we remarked in Remark A.9, the multi-challenge security notion is equivalent to more standard single-challenge security notion.

Simulating the Setup. At the beginning of the game, B is given $\mathsf{IBE.mpk}$ from the challenger. B then gives $\mathsf{mpk} := \mathsf{IBE.mpk}$ to A. Furthermore, B generates $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i)$ for $i \in [i_{\max}]$. B then makes secret key query for identity $(i, j, \mathsf{FE.mpk}_{i,j})$ for all $i \in [i_{\max}]$ and $j \in [\ell_i]$ to receive $\mathsf{IBE.sk}_{i,j}$.

Answering the secret key queries. Throughout the game, B can easily answer the secret key queries made by A using $\{\mathsf{IBE.sk}_{i,j}\}_{i\in[i_{\max}],j\in[\ell_i]}$ and $\{\mathsf{FE.msk}_i\}_{i\in[i_{\max}]}$.

Answering the encryption query. When A submits an encryption query $(x, 1^Q)$, B does the following:

1. It computes $\{\mathsf{lab}_{i,j,b}\}_{i\in\mathcal{S}(|x|),j\in\ell_i,b\in\{0,1\}}$ as honest encryption algorithm.
2. It computes $\{\mathsf{IBE.ct}_{i,j,b}\}_{i\in\mathcal{S}(|x|),j\in[\ell_i],b\in\{0,1\}}$ as follows. There are two cases:
   - If $b = \mathsf{FE.mpk}_{i,j}$, it honestly encrypts $\mathsf{lab}_{i,j,b}$ to obtain $\mathsf{IBE.ct}_{i,j,b}$.
   - If $b = 1 - \mathsf{FE.mpk}_{i,j}$, B submits identity (i,j,b) and messages $(\mathsf{lab}_{i,j,b}, \mathsf{lab}_{i,j,1-b})$ to its challenger. Then, the challenger encrypts

$$\mathsf{IBE.ct}_{i,j,b} \leftarrow \begin{cases} \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,b), \mathsf{lab}_{i,j,b}\right), & \text{if } \beta = 0 \\ \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,b), \mathsf{lab}_{i,j,1-b}\right) & \text{if } \beta = 1 \end{cases}$$

   and returns $\mathsf{IBE.ct}_{i,j,b}$ to B. Note that the same label $\mathsf{lab}_{i,j,\mathsf{FE.mpk}_{i,j}}$ is encrypted for identities $(i,j,0)$ and $(i,j,1)$ if $\beta = 1$.
3. It then returns the ciphertext $\mathsf{ct} = \left(\mathcal{S}(|x|), \{\mathsf{IBE.ct}_{i,j,b}\}_{i\in\mathcal{S}(|x|),j\in\ell_i,b\in\{0,1\}}\right)$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_2$ for A if $\beta = 0$ and $\mathbf{Game}_3$ if $\beta = 1$. Furthermore, B does not make a secret key query for an identity that is also submitted to the challenger for an encryption query. To see this, observe that B submits identities of the form $(i,j,\mathsf{FE.mpk}_{i,j})$ for secret key queries and $(i,j,1-\mathsf{FE.mpk}_{i,j})$ for encryption queries to the challenger. This means that B breaks the IND-CPA security of IBE without violating the query restriction of the game, if A distinguishes the games. □

Lemma E.5. *We have* $|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]| \leq \mathrm{negl}(\lambda)$ *by the security of* GC.

Proof. Assume $|\Pr[\mathcal{E}_3] - \Pr[\mathcal{E}_4]|$ is non-negligle. We then describe an adversary B that breaks the multi-instance security of GC as follows. Note that since the multi-instance security of GC is implied by the single instance security as we observed in Remark A.4, this is a contradiction.

Simulating the Setup. At the beginning of the game, B runs $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$ and then gives $\mathsf{mpk} := \mathsf{IBE.mpk}$ to A. Furthermore, B generates $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i)$ for $i \in [i_{\max}]$.

Answering the secret key queries. Throughout the game, B can easily answer the secret key queries made by A using $\mathsf{IBE.msk}$ and $\{\mathsf{FE.msk}_i\}_{i\in[i_{\max}]}$.

Answering the encryption query. When A submits an encryption query $(x, 1^Q)$, B proceeds as follows.

1. It computes $\mathcal{S}(|x|) \subset \mathbb{N}$ and $f(x) = \{f(x)_i\}_{i\in\mathcal{S}(|x|)}$.
2. It does the following for $i \in \mathcal{S}(|x|)$.
   (a) It samples r_i and defines a circuit $\mathsf{E}_i(\cdot) := \mathsf{FE.Enc}(\cdot, f(x)_i, 1^Q; r_i)$.
   (b) It submits the circuit $\mathsf{E}_i(\cdot)$ and $\mathsf{FE.mpk}_i$ to the challenger. The challenger either runs

$$\{\mathsf{lab}_{i,j,b}\}_{j\in[\ell_i],b\in\{0,1\}} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{E}_i)$$

   or

$$\{\mathsf{lab}_{i,j}\}_{j\in[\ell_i]} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_i}, 1^{|\mathsf{E}_i|}, \mathsf{E}_i(\mathsf{FE.mpk}_i))$$

   depending on whether B is in the real game or ideal game. Then, B is given $\{\mathsf{lab}_{i,j}\}_{j\in[\ell_i]}$ where $\mathsf{lab}_{i,j}$ is set $\mathsf{lab}_{i,j} = \mathsf{lab}_{i,j,\mathsf{FE.mpk}_{i,j}}$ in the real case.

(c) For all $j \in [\ell_i]$ and $b \in \{0,1\}$, it computes

$$\mathsf{IBE.ct}_{i,j,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,b), \mathsf{lab}_{i,j}\right).$$

3. Finally, it returns the ciphertext $\mathsf{ct} = \left(\mathcal{S}(|x|), \{\mathsf{IBE.ct}_{i,j,b}\}_{i\in\mathcal{S}(|x|), j\in[\ell_i], b\in\{0,1\}}\right)$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_3$ for A if it is in the real world and $\mathbf{Game}_4$ if it is in the ideal world. Thus, B breaks the multi-instance security of GC if A distinguishes between the two games. □

Lemma E.6. *We have* $|\Pr[\mathcal{E}_4] - \Pr[\mathcal{E}_5]| \leq \mathrm{negl}(\lambda)$ *by* AD-SIM *security of* FE.

Proof. Assume $|\Pr[\mathcal{E}_4] - \Pr[\mathcal{E}_5]|$ is non-negligle. We then describe an adversary B that breaks the multi-instance AD-SIM security of FE as follows. Note that since the multi-instance AD-SIM security of FE is implied by the single instance security as we observed in Remark 2.8, this is a contradiction.

Simulating the Setup. At the beginning of the game, B runs $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$ and then gives $\mathsf{mpk} := \mathsf{IBE.mpk}$ to A. Furthermore, B declares the number $i_{\max}$ of instances and the parameter $\mathsf{prm} = 1^i$ for the i-th instance of FE for all $i \in [i_{\max}]$. Then, the game runs $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i)$ for $i \in [i_{\max}]$ and gives $\{\mathsf{FE.mpk}_i\}_{i\in[i_{\max}]}$ to A.

Answering the secret key queries before the encryption query. When A makes a secret key query for $y^{(q)}$, B does the following.

1. It computes $\mathcal{T}(|y^{(q)}|) \subseteq \mathbb{N}$ and $g(y^{(q)}) = \left\{g(y^{(q)})_i \in \mathcal{Y}_i\right\}_{i\in\mathcal{T}(|y|)}$.
2. For all $i \in \mathcal{T}(|y^{(q)}|)$, B does the following. It makes a secret key query for $g(y^{(q)})_i$ for the i-th instance of FE to its challenger. Then, the challenger computes

$$\mathsf{FE.sk}_i^{(q)} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, g(y^{(q)})_i)$$

and returns $\mathsf{FE.sk}_i^{(q)}$ to B.

3. B generates a secret key $\mathsf{IBE.sk}_{i,j} \leftarrow \mathsf{IBE.KeyGen}(\mathsf{IBE.msk}, (i, j, \mathsf{FE.mpk}_{i,j}))$ for all $i \in \mathcal{T}(|y^{(q)}|)$ and $j \in \ell_i$.
4. It returns $\mathsf{sk}^{(q)} = \left(\mathcal{T}(|y^{(q)}|), \left\{\mathsf{FE.sk}_i^{(q)}, \{\mathsf{IBE.sk}_{i,j}\}_{j\in[\ell_i]}\right\}_{i\in\mathcal{T}(|y^{(q)}|)}\right)$ to A.

Answering the encryption query. When A submits an encryption query $(x, 1^Q)$, B proceeds as follows.

1. It computes the set $\mathcal{S}(|x|)$ and $f(x) = \{f(x)_i\}_{i\in\mathcal{S}(|x|)}$.
2. For all $i \in \mathcal{S}(|x|)$, B does the following.

(a) It makes an encryption query for $(f(x)_i, 1^Q)$ for the i-th instance. The challenger runs

$$(\mathsf{FE.ct}_i, \mathsf{FE.st}_i) \leftarrow \begin{cases} \mathsf{FE.Enc}(\mathsf{FE.mpk}_i, f(x)_i), & \text{if B is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE,B}}(1^\lambda) \\ \mathsf{FE.SimEnc}\left(\mathsf{FE.mpk}_i, \mathcal{V}_{\mathsf{FE},i}, 1^Q\right), & \text{if B is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE,Sim}}(1^\lambda) \end{cases}$$

where

$$\mathcal{V}_{\mathsf{FE},i} = \left\{ R_i\left(f(x)_i, g(y^{(q)})_i\right),\ g(y^{(q)})_i,\ \mathsf{FE.sk}_i^{(q)} \right\}_{q\in\Gamma_i}$$

and returns $\mathsf{FE.ct}_i$ to B. Recall that Γ_i is defined as $\Gamma_i := \left\{q \in [Q_1] : i \in \mathcal{T}(|y^{(q)}|)\right\}$.

(b) It runs $\{\mathsf{lab}_{i,j}\}_{j\in[\ell_i]} \leftarrow \mathsf{GC.Sim}(1^\lambda, 1^{\ell_i}, 1^{|\mathsf{E}_i|}, \mathsf{FE.ct}_i)$.

(c) For all $j \in [\ell_i]$ and $b \in \{0,1\}$, it computes $\mathsf{IBE.ct}_{i,j,b} \leftarrow \mathsf{IBE.Enc}\left(\mathsf{IBE.mpk}, (i,j,b), \mathsf{lab}_{i,j}\right)$.

3. It returns the ciphertext $\mathsf{ct} = \left(\mathcal{S}(|x|), \{\mathsf{IBE.ct}_{i,j,b}\}_{i\in\mathcal{S}(|x|), j\in[\ell_i], b\in\{0,1\}}\right)$ to A.

Simulating the secret key after the encryption query. When A makes a secret key query for $y^{(q)}$ with $q \in [Q_1, Q_1+Q_2]$, B proceeds as follows.

1. It computes the set $\mathcal{T}(|y^{(q)}|)$ and $g(y^{(q)}) = \left\{g(y^{(q)})_i \in \mathcal{Y}_i\right\}_{i\in\mathcal{T}(|y^{(q)}|)}$.
2. For all $i \in \mathcal{T}(|y^{(q)}|)$, B makes a secret key query for $g(y^{(q)})_i$ for the i-th instance to its challenger.
   - If $i \in \mathcal{S}(x)$, the challenger runs
   $$\mathsf{FE.sk}_i^{(q)} \leftarrow \begin{cases} \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, g(y^{(q)})_i), & \text{if } \mathsf{B} \text{ is in } \mathsf{Exp}^{\mathsf{real}}_{\mathsf{FE},\mathsf{B}}(1^\lambda) \\ \mathsf{FE.SimKG}\left(\mathsf{FE.st}_i, \mathsf{FE.msk}_i, \mathcal{V}^{(q)}_{\mathsf{FE},i}, 1^Q\right), & \text{if } \mathsf{B} \text{ is in } \mathsf{Exp}^{\mathsf{ideal}}_{\mathsf{FE},\mathsf{Sim}}(1^\lambda) \end{cases}$$
   where
   $$\mathcal{V}^{(q)}_{\mathsf{FE},i} = \left\{ R_i\left(f(x)_i, g(y^{(q)})_i\right),\ g(y^{(q)})_i \right\}$$
   and returns $\mathsf{FE.sk}_i^{(q)}$ to B.
   - If $i \notin \mathcal{S}(x)$, the challenger runs
   $$\mathsf{FE.sk}_i^{(q)} \leftarrow \mathsf{FE.KeyGen}(\mathsf{FE.msk}_i, g(y^{(q)})_i)$$
   and returns it to B regardless of whether B is in the real world or ideal world, since B has not made an encryption query for the i-th instance.
3. For all $i \in \mathcal{T}(|y^{(q)}|)$ and $j \in \ell_i$, it honestly generates $\mathsf{IBE.sk}_{i,j}$.
4. It returns $\mathsf{sk}^{(q)} = \left(\mathcal{T}(|y^{(q)}|),\ \left\{\mathsf{FE.sk}_i^{(q)}, \{\mathsf{IBE.sk}_{i,j}\}_{j\in[\ell_i]}\right\}_{i\in\mathcal{T}(|y^{(q)}|)}\right)$ to A.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is straightforward to see that B simulates $\mathbf{Game}_4$ for A if it is in the real world and $\mathbf{Game}_5$ if it is in the ideal world. Furthermore, B restricts the restrictions on queries for AD-SIM security. Namely, for each instance i, B chooses Q as its collusion bound and makes at most Q secret key queries. Thus, B breaks the multi-instance AD-SIM security of FE without violating the query restrictions if A distinguishes between the two games. □

Lemma E.7. *We have* $|\Pr[\mathcal{E}_6] - \Pr[\mathcal{E}_7]| \le \mathrm{negl}(\lambda)$ *by the security of* PRF.

Proof. Assume $|\Pr[\mathcal{E}_6] - \Pr[\mathcal{E}_7]|$ is non-negligle. We then describe an adversary B that breaks the PRF security.

Simulating the Setup. At the beginning of the game, B runs $(\mathsf{IBE.mpk}, \mathsf{IBE.msk}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$ and gives $\mathsf{mpk} = \mathsf{IBE.mpk}$ to A.

Answering the secret key queries before the encryption query. When A makes a secret key query for $y^{(q)}$ with $q \in [Q_1]$, B does the following.

1. Compute $\mathcal{T}(|y^{(q)}|) \subseteq \mathbb{N}$ and $g(y^{(q)}) = \left\{g(y^{(q)})_i \in \mathcal{Y}_i\right\}_{i\in\mathcal{T}(|y^{(q)}|)}$.

2. For all $i \in \mathcal{T}(|y^{(q)}|)$, B does the following. It checks whether $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ has been generated or not. If not, it makes a PRF query on input i and obtains R_i, where either $\mathsf{R}_i = \mathsf{PRF.Eval}(\mathsf{K}, i)$ or R_i is random. It then runs $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i; \mathsf{R}_i)$.

3. Then, B answers the secret key query using $\mathsf{IBE.msk}$ and $\{\mathsf{FE.msk}_i\}_{i \in \mathcal{T}(|y^{(q)}|)}$.

Answering the encryption query. When A submits an encryption query $(x, 1^Q)$, B proceeds as follows.

1. Compute $\mathcal{S}(|x|) \subseteq \mathbb{N}$ and $f(x) = \{f(x)_i \in \mathcal{X}_i\}_{i \in \mathcal{S}(|x|)}$.

2. For all $i \in \mathcal{S}(|x|)$, B does the following. It checks whether $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i)$ has been generated or not. If not, it runs $(\mathsf{FE.mpk}_i, \mathsf{FE.msk}_i) \leftarrow \mathsf{FE.Setup}(1^\lambda, 1^i)$ using true randomness.

3. Then, B generates the ciphertext using $\mathsf{IBE.mpk}$ and $\{\mathsf{FE.mpk}_i\}_{i \in \mathcal{S}(|x|)}$. During the simulation of the ciphertext, B generates $\mathsf{FE.st}_i$ for $i \in \mathcal{S}(|x|)$. It keeps them for future use.

Answering the secret key queries after the encryption query. The secret key quries after the encryption queries are handled similarly to those before the encryption query. However, in this case, B may use $\mathsf{FE.st}_i$ for $i \in \mathcal{S}(|x|)$.

Final guess. At the end of the game, A outputs its guess. B outputs the same guess as A.

It is easy to see that B simulates $\mathbf{Game}_7$ if $\mathcal{B}$ has access to a PRF function and $\mathbf{Game}_6$ otherwise. Thus, B breaks the PRF security if A distinguishes between the two games. □