Impossibility of Simulation Secure Functional Encryption Even with Random Oracles

Shashank Agrawal* Venkata Koppula† Brent Waters‡

Abstract

In this work we study the feasibility of achieving simulation security in functional encryption (FE) in the random oracle model. Our main result is negative in that we give a functionality for which it is impossible to achieve simulation security even with the aid of random oracles.

We begin by giving a formal definition of simulation security that explicitly incorporates the random oracles. Next, we show a particular functionality for which it is impossible to achieve simulation security. Here messages are interpreted as seeds to a (weak) pseudorandom function family F and private keys are ascribed to points in the domain of the function. On a message s and private key x one can learn $F(s, x)$. We show that there exists an attacker that makes a polynomial number of private key queries followed by a single ciphertext query for which there exists no simulator.

Our functionality and attacker access pattern closely matches the *standard model* impossibility result of Agrawal, Gorbunov, Vaikuntanathan and Wee (CRYPTO 2013). The crux of their argument is that no simulator can succinctly program in the outputs of an unbounded number of evaluations of a pseudorandom function family into a fixed size ciphertext. However, their argument does not apply in the random oracle setting since the oracle acts as an additional conduit of information which the simulator can program. We overcome this barrier by proposing an attacker who decrypts the challenge ciphertext with the secret keys issued earlier *without* using the random oracle, even though the decryption algorithm may require it. This involves collecting most of the useful random oracle queries in advance, without giving the simulator too many opportunities to program. We note that our negative result contradicts a theorem of De Caro et al. (CRYPTO 2013) which claims that random oracles can transform any indistinguishability secure functional encryption system into one that is simulation secure.

On the flip side, we demonstrate the utility of the random oracle in simulation security. Given only public key encryption and low-depth PRGs we show how to build an FE system that is simulation secure for any poly-time attacker that makes an unbounded number of message queries, but an a-priori bounded number of key queries. This bests what is possible in the standard model where it is only feasible to achieve security for an attacker that is bounded both in the number of key and message queries it makes. We achieve this by creating a system that leverages the random oracle to get one-key security and then adapt previously known techniques to boost the system to resist up to q queries.

Finally, we ask whether it is possible to achieve simulation security for an unbounded number of messages and keys, but where all key queries are made *after* the message queries. We show this too is impossible to achieve using a different twist on our first impossibility result.

*University of Texas at Austin. Email: sagrawal@cs.utexas.edu. Supported by NSF CNS-1228599 and CNS-1414082, and DARPA SafeWare.

†University of Texas at Austin. Email: k.venkata.vk@gmail.com. Supported by NSF CNS-1228599 and CNS-1414082, and DARPA SafeWare.

‡University of Texas at Austin. Email: bwaters@cs.utexas.edu. Supported by NSF CNS-1228599 and CNS-1414082, DARPA SafeWare, Microsoft Faculty Fellowship, and Packard Foundation Fellowship.

1 Introduction

The traditional notion of public key encryption systems provide "all or nothing" semantics regarding encrypted data. In such a system a message m is encrypted under a public key, pk, to produce a ciphertext ct. A user that holds the corresponding secret key can decrypt ct and learn the entire message m, while any other user will not learn anything about the contents of the message. The work of Sahai and Waters [23] conceived cryptosystems that moved beyond these limited semantics to ones where a private key would give a select view of encrypted data. These efforts [23, 10, 18] cumulated in the concept of functional encryption. In a functional encryption system an authority will generate a pair of a public key and master key pair $(\mathsf{pk}, \mathsf{msk})$. Any user can encrypt a ciphertext ct using the public key, while the authority can use the master secret key msk to generate a secret key sk_f that is tied to the functionality f. A holder of sk_f can use it to decrypt a ciphertext ct, but instead of learning the message m, the decryptor's decryption will instead output $f(m)$.

One challenge in defining and designing functional encryption (FE) systems is in finding a definition to capture security. The earliest formal definitions of functional encryption [10, 18] (back when the terminology of "predicate encryption was used") defined security in terms of an indistinguishability game. Briefly, a system is indistinguishability secure if no poly-time attacker that receives secret keys for functions $f_1, \ldots, f_Q$ can distinguish between encryptions of m_0, m_1 so long as $f_i(m_0) = f_i(m_1)\ \forall i \in [Q]$.

Subsequent works [9, 21, 4, 1] aimed to capture various notions of simulation-based security. To achieve simulation one must be able to show that for each attacker there exists a poly-time simulator $\mathcal{S}$ that can produce a transcript that emulates the attacker's real world view, but when only given access to what the evaluation of the secret key functions $f(\cdot)$ were on the attacker's messages. (We will return to describing simulation-based security in more detail shortly.) While these simulation definitions had the appeal of perhaps capturing a stronger notion of security than the indistinguishability-based ones, they were limited in that multiple works [9, 21, 4, 1] showed that this notion is impossible to achieve in the standard model for even very basic functionalities such as identity-based encryption [24, 8]. The only exception being in the restricted case where the attacker is only allowed to access an a-priori number of secret keys [15].

While these results essentially put a hard stop on realizing (collusion-resistant) simulation security in the standard model, the door to leveraging the random oracle model [5] still remained wide open. Notably, Boneh, Sahai and Waters [9] building on techniques from non-committing encryption [20] showed that the random oracle could be leveraged to turn any indistinguishability secure public index FE scheme into one that was simulation secure. Recall that a public index scheme is one where an encrypted message is split into a hidden payload and a non-hidden index and the secret key operates only on the index. The set of such schemes includes identity-based encryption [24, 8] and attribute-based encryption [23]. Thus, they showed that introducing a random oracle was enough to circumvent their own standard model IBE result. In this work we wish to understand what are the possibilities and limitations (if any) for using random oracles to achieve simulation security in FE systems. Our work begins with the question:

Is it possible to achieve simulation secure functional encryption for any functionality in the random oracle model?

Our main result is to show that there exist functionalities for which their cannot exist a simulation secure functional encryption system *even* in the random oracle model. A reader familiar with the literature might notice that our claim puts us squarely at odds with the work of De Caro et al. [12] who claim to show that any FE scheme for poly-sized circuits system that is (adaptively) secure in the indistinguishability sense can be transformed to one that is simulation secure using random oracles.

To understand the discrepancy it is instructive to review the basic tenets of a proof of security in the random oracle model. The first step is to establish a construction that has all the necessary algorithms and allows each of the algorithms to call the random oracle. Next, this random oracle construction must be proven secure under a security game where both the algorithms and adversary are allowed to make calls to the oracle. The fundamental issue of the work of De Caro et al. [12] is that the authors never establish a random oracle construction to begin with. Instead of describing a construction that accesses the random oracle, they start by describing a function that calls a hash function. Moreover, the call to the hash function is embedded inside the secret key of the underlying indistinguishability secure FE system which is invoked during decryption and its evaluation may be garbled or spread out in an arbitrary manner over several steps. Therefore there is no obvious place where the hash function "is called" and no clear random oracle construction that can be derived from the description. The proof of security in both the conference and eprint versions at the time of submission is stated to be deferred to the full version of the paper. We contacted an author [17] of the De Caro et al. paper who confirmed that there was an error in their claim due to the reasons described above.

On the flip side, we demonstrate the utility of the random oracle in simulation security. Given only public key encryption and low-depth PRGs we show how to build an FE system that is simulation secure for any poly-time attacker that makes an unbounded number of message queries, but an a-priori bounded number of key queries. This bests what is possible in the standard model where it is only feasible to achieve security for an attacker that is bounded both in the number of key and message queries it makes. We achieve this by creating a system that leverages the random oracle to get one-key security and then adapt previously known techniques to boost the system to resist up to q queries.

Finally, we ask whether it is possible to achieve simulation security for an unbounded number of messages and keys, but where all key queries are made *after* the message queries. We show this too is impossible to achieve by repurposing our main impossibility result to the new setting.

1.1 Our Main Impossibility Result

We show the impossibility result for the case where messages are interpreted as keys or seeds to a (weak) Pseudo Random Function (PRF) [13] family and secret keys are points in the domain of the PRF. Agrawal, Gorbunov, Vaikuntanathan and Wee [1] showed that such a functionality could not be simulation secure in the standard model. Here we show that this limitation holds even with the introduction of random oracles.

We begin our exposition by describing the definition of simulation security in a little more depth and briefly overviewing the AGVW impossibility analysis.

Simulation security. Simulation security for FE is defined by means of real and ideal experiments. In the real experiment, an adversary $\mathcal{A}$ gets secret keys for functions f and and ciphertexts for challenge messages m of its choice. The secret key queries can either be sent before the challenge messages (also referred to as pre-challenge queries) or after the challenge messages (post challenge queries). In the ideal world, on the other hand, a simulator $\mathcal{S}$ needs to generate challenge ciphertexts and keys given only the minimal information. In particular, when $\mathcal{A}$ requests that a challenge message m be encrypted, $\mathcal{S}$ only gets $f(m)$ on all the pre-challenge functions f queried by $\mathcal{A}$ (instead of m itself), and must generate a ciphertext that $\mathcal{A}$ cannot distinguish from the one in the real world. Similarly, when $\mathcal{A}$ makes a post-challenge key query for f', $\mathcal{S}$ must generate a secret key given just f' and $f'(m)$ for all challenge messages m.

An FE scheme is $(q_{\mathsf{pre}}, q_{\mathsf{chal}}, q_{\mathsf{post}})$-simulation secure if it can withstand adversaries that make at most q_{pre} pre-challenge key queries, q_{chal} challenge encryption requests, and q_{post} post-challenge key queries. Ideally, one would like to capture all polynomial-time adversaries, who can make any number of queries they want. However, even simple functionalities like identity-based encryption do not have a scheme secure against an arbitrary number of encryption requests followed by one post-challenge key query, i.e., IBE does not have a $(0, \mathsf{poly}, 1)$-simulation secure scheme [9, 4] in the standard model. Here poly denotes that any number of encryption requests can be made, as long as there is a polynomial bound on them.

AGVW impossibility. A different kind of impossibility was shown by Agrawal et al. [1]. They interpret messages as seeds to a weak pseudorandom family wPRF[1] and secret keys as points in the domain of the family. When a ciphertext for s is decrypted with a secret key for x, the output is $\mathsf{wPRF}(s, x)$. They show that there does not exist a simulation-secure FE scheme for this family that can tolerate adversaries which can make an arbitrary number of pre-challenge key queries and then request for the encryption of just one message (i.e., $(\mathsf{poly}, 1, 0)$-simulation security). Intuitively, when the adversary outputs a message s in the ideal world, the simulator gets $\mathsf{wPRF}(s, x_1), \ldots, \mathsf{wPRF}(s, x_q)$ (if q is the number of post-challenge key queries), which is computationally indistinguishable from q uniformly random strings. The simulator must output a ciphertext ct now that decrypts correctly with all the keys issued before. Note that when the keys were issued, simulator had no information about s, so it must somehow *compress* q random strings into ct. However, as Agrawal et al. show, the output of a pseudo-random function family is *incompressible*. Thus, by choosing a large enough q, they arrive at the impossibility result.

[1] A weak pseudorandom function family provides security only against attackers that do not get to choose the points at the which the PRF is evaluated. These points are chosen randomly by the challenger.

Random oracle model. In the random oracle model though, Agrawal et al.'s impossibility argument breaks down. Informally speaking, the random oracle acts as an additional conduit of information which the simulator can program even *after* ct appears. For instance, if the decryption algorithm makes RO queries, then the simulator could program such queries when adversary tries to decrypt ct with the secret keys issued earlier. Indeed, Boneh et al. show that their $(0, \mathsf{poly}, 1)$ impossibility for IBE can be circumvented by employing RO in the encryption and decryption algorithms.

Thus we need a very different approach. We would like to build an adversary $\mathcal{A}^\star$ that "cuts off" RO in the decryption process, and is able to work without it. This involves a delicate balancing act between cutting off too early and too late. In one extreme case, if $\mathcal{A}^\star$ does not invoke RO at all and makes up its own responses, then these would not match with the actual RO responses in encryption and key generation. Thus decryption would always fail in both the real and ideal worlds, and there will be no distinction between them. On the other extreme, if $\mathcal{A}^\star$ just used the RO all the way through, it would provide the simulator enough opportunity to program in the desired information. (As a result, we will not be able to use the incompressibility of wPRF.)

At a high level, our approach is to have an initial learning phase where $\mathcal{A}^\star$ will build a list of "high frequency" random oracle queries and responses associated with each secret key and the challenge ciphertext. Later the attacker will be able to use this list to replace the use of the actual random oracle during decryption. If some query is not found in the list, then $\mathcal{A}^\star$ will choose a random value for it on its own. Informally, we get the following result:

Theorem 1.1 (Main Theorem, informal)**.** *There does not exist a* $(\mathsf{poly}, 1, 0)$*-simulation secure FE scheme for the class of (weak) pseudo-random functions in the random oracle model.*

Related work. This bears a resemblance to the work of Canetti, Kalai and Paneth [11] who show impossibility of VBB obfuscation even with ROs. In their case they show that any obfuscated program that uses the RO can be translated into one that does not need it. They do this by collecting the frequently used RO queries and bundling this with the core obfuscated code. On one hand, these queries do not give any information about the program, but on the other, result in an obfuscation that is only approximately correct. Such imperfect correctness, however, is enough to invoke the impossibility of Bitansky and Paneth [6].

One might ask if we can show whether RO can be dispensed with in any simulation secure FE in a similar way. If we could establish this, then prior impossibility results [9, 4, 1] would imply RO impossibility as well. The answer to this is negative as we recall that Boneh, Sahai and Waters [9] showed specific functionalities that were impossible to simulate in the standard model, but possible to be simulation secure using random oracle. Therefore we cannot always remove the random oracle and must develop a more nuanced approach: we need to build a specific adversary for which simulation does not work.

In a recent work [19], Mohammad et al. show that there is no fully black-box construction of indistinguishability obfuscation (iO) from any primitive implied by a random oracle in a black-box way. In light of recent FE to iO transformations [2, 7], one might wonder if this rules out FE schemes in the RO model. However, these transformations are non-black box.

1.1.1 High level description of impossibility

Recall that we want to design an adversary $\mathcal{A}^\star$ that will build a list of "high frequency" random oracle queries and responses associated with each secret key and the challenge ciphertext. It will use this list later in the decryption phase to "cut-off" the random oracle at an appropriate time.

$\mathcal{A}^\star$ starts off by querying the key-generation oracle at random points $x_1, \ldots, x_q$ in the domain of wPRF, and gets $\mathsf{sk}_1, \ldots, \mathsf{sk}_q$ in return. The RO queries made by the key-generation oracle are *hidden* from the adversary, so $\mathcal{A}^\star$ tries to find them by encrypting several randomly chosen seeds using the master public key, and then decrypting them with $\mathsf{sk}_1, \ldots, \mathsf{sk}_q$.[2] The RO queries made during the decryption process are recorded in a list Γ. The hope is that Γ will capture the RO queries that were made in generating a key sk_i.

Note that one cannot hope to capture *all* RO queries required for decryption: Suppose a polynomial number Y of high frequency queries associated with sk_i is collected, but there is an RO call that is made during key-generation which is used during $1/2Y$ fraction of the decryptions. Then it will be the case that with some

[2] It is important that this is done before the challenge message is put out, otherwise simulator will get an opportunity to program in additional information through the random oracle.

non-negligible probability, Γ will fail to aid in the decryption of the challenge ciphertext with sk_i. Instead of trying to solve this issue, we make our analysis work with a decryption that might fail some of the time. For this purpose, we extend the incompressibility argument of Agrawal et al. to work even for *approximate* compression.

We are not quite done yet. Even though we have captured most of the hidden RO queries involved in key-generation that are also needed for decryption, we still need to capture those that are involved in the encryption of the challenge message, as they are also *hidden* and may be required during decryption.[3] Suppose $\mathcal{A}^\star$ outputs a randomly chosen seed $s^\star$ as the challenge message, and gets $\mathsf{ct}^\star$ in return. In order to find out RO queries associated with $\mathsf{ct}^\star$, $\mathcal{A}^\star$ cannot generate secret keys on its own (like in the pre-challenge phase when it generated ciphertexts); it must make-do with the secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_q$ that were issued earlier. Thus, the idea is to decrypt $\mathsf{ct}^\star$ with some fraction δ of the keys using RO, recording the queries in the list Γ. It then cuts off the random oracle, and decrypts $\mathsf{ct}^\star$ with the remaining keys using the list Γ. If a query is not found in Γ, then a random value is used for it (as well as recorded in Γ for consistent responses in future). The adversary outputs 1 if a large fraction of these decryptions are correct; that is, if the decryption of $\mathsf{ct}^\star$ using sk_i outputs $\mathsf{wPRF}(s^\star, x_i)$.

In the real world, as we will see, the adversary outputs 1 with noticeable probability. On the other hand, we show that in the ideal world, the adversary outputs 1 only with negligible probability. For the adversary to output 1 in the ideal world, the simulator needs to somehow program the ciphertext and the post-challenge random oracle queries so that a large number of decryptions succeed. The only opportunity a simulator has of programming post-challenge RO responses is when δ fraction of the keys are used for decrypting $\mathsf{ct}^\star$. By choosing δ appropriately, we can ensure that the simulator is not able program the RO queries to the extent that most of the remaining decryptions succeed.

Looking back. A simulator's success in the RO model depends on when it comes to know what to program and how much can it program. When dealing with the attacker $\mathcal{A}^\star$ described above, it gets a large amount of information, $\mathsf{wPRF}(s^\star, x_1), \ldots, \mathsf{wPRF}(s^\star, x_q)$, only in the challenge phase. Since all key queries come before that, programming the secret keys is ruled out. If there was no random oracle, then the only possible avenue to program is the challenge ciphertext, but AGVW shows that it is not possible to compress so much information into a small ciphertext. Now with the random oracle, it might have been possible to program this information *if* there were many RO queries after the challenge phase. However, our adversary makes only a bounded number of post-challenge RO queries, and as a result, it is not possible to program all of $\{\mathsf{wPRF}(s^\star, x_i)\}$ in these RO responses.

1.2 A New Possibility Result in the Random Oracle Model

Now that we know that simulation security is impossible for unbounded queries even in the random oracle model, we turn to asking whether this model can be leveraged to support simulation security in any situations where it is impossible in the standard model. We already have one such example from the work of Boneh et al. [9] which gives both a standard model impossibility and a random oracle feasibility result for public index schemes. Thus, we are interested in new examples that go beyond the public index class. In this paper, we show the following possibility result:

Theorem 1.2 (Possibility, informal)**.** *There exists a simulation secure FE scheme for the class of all polynomial-depth circuits in the random oracle model secure against any poly-time attacker who makes an unbounded number of messages queries, but an a-priori bounded number of key queries, based on semantically-secure public-key encryption and pseudo-random generators computable by low-depth circuits.*

Recall that such a security notion cannot be achieved even for the simple functionality of IBE in the standard model [9].

One-bounded FE. Our starting point is a *one-bounded* simulation-secure FE scheme for all circuits, i.e., a scheme where the attacker can only make one key query, based just on the semantic security of public-key encryption. Our scheme can be easily understood in the familiar terminology of Yao's garbled circuits, though

[3] Even the RO queries made while setting up the FE system are hidden from the adversary, but we ignore them here for simplicity.

we use decomposable randomized encodings for a more general and cleaner construction. Let $\mathcal{C}$ be a family of circuits wherein each circuit can be represented using t bits. Suppose U_x is a universal circuit that takes a $C \in \mathcal{C}$ as input, and outputs $C(x)$. The set-up algorithm of our FE scheme generates $2t$ key pairs of a semantically-secure public-key encryption scheme. The $2t$ public keys $(\mathsf{pk}_{1,0}, \mathsf{pk}_{1,1}), \ldots, (\mathsf{pk}_{t,0}, \mathsf{pk}_{t,1})$ form the master public key, and the t private keys $(\mathsf{sk}_{1,0}, \mathsf{sk}_{1,1}) \ldots, (\mathsf{sk}_{t,0}, \mathsf{sk}_{t,1})$ are kept secret. In order to encrypt a message x, a garbled circuit for U_x is generated. Suppose $w_{i,b}$ for $i = 1, \ldots, t$ and $b = 0, 1$ are the *wire-labels* of U_x for its t input bits. Then the $(i, b)^{th}$ component of the ciphertext consists of two parts: an encryption of a random value $r_{i,b}$ under $\mathsf{pk}_{i,b}$, and $w_{i,b}$ blinded with the hash of $r_{i,b}$. The key for a circuit C represented using bits $\beta_1, \ldots, \beta_t$ is simply the private keys corresponding to those bits, i.e., $\mathsf{sk}_{\beta_1}, \ldots, \mathsf{sk}_{\beta_t}$.

It is easy to see that the one-bounded FE scheme is correct. Specifically, the secret key for C will allow one to recover r_{i,β_i} for $i = 1, \ldots, t$. Then by running the hash function on these values, the w_{i,β_i} can be unblinded and used to evaluate the garbled circuit.

Let us now see how a simulator $\mathcal{S}$ can generate ciphertexts and a key from the right distribution in the ideal world. If the only allowed key query is made before the challenge phase for a circuit C, then $\mathcal{S}$ just runs the normal key generation algorithm, and later when adversary outputs a challenge message $x^\star$, it can generate a garbled circuit using just $C(x^\star)$.[4] When the adversary's key query is after the challenge message, however, $\mathcal{S}$ does not get any information in the challenge phase. In particular, it does not know which universal circuit to garble. Here the random oracle allows the simulator to *defer* making a decision until after the key query is made. It can set the second part of the (i, b)th ciphertext component to be a random number $z_{i,b}$ because, intuitively, adversary does not know $r_{i,b}$ (it is encrypted) so a hash of it is completely random. When adversary queries with a circuit C afterwards, simulator can program the random oracle's response on $r_{i,b}$ to be $z_{i,b} \oplus w_{i,b}$, so that decryption works out properly.

Bounded collusion FE. Using the one-bounded scheme in a black-box way, we can design an FE scheme secure against any a-priori bounded collusions for the class $\mathsf{NC1}$, without making any additional complexity assumptions. We borrow Gorbunov et al.'s transformation [15] for this purpose, but it was proved secure for only one challenge message. We show that if the underlying one-bounded scheme is secure against any number of challenge messages, then so is the scheme obtained after applying their transformation.

In fact, GVW12's idea of using constant-depth randomized encodings to bootstrap from $\mathsf{NC1}$ to the class of polynomial-depth circuits can also be applied, as we will show, to FE schemes secure against an arbitrary number of challenge messages.

Related work. Sahai and Seyalioglu [22] were the first to use randomized encodings to design an FE system. Their scheme can issue one key *non-adaptively* for any function. Our one-bounded scheme can be seen as an extension of theirs to additionally support post-challenge key query. The random oracle allows a simulator to not commit to any value in the ciphertext until the function evaluation is made available.

Goldwasser et al. [14] also designed an FE system that can issue one pre-challenge key. Their scheme has succinct ciphertexts (independent of circuit size) but security is proved under stronger assumptions.

1.3 Another Impossibility Result

A natural question to ask is whether we can construct a simulation secure FE scheme in the random oracle model that can handle unbounded ciphertext queries, followed by an unbounded number of post-challenge key queries. We show that this is also impossible, assuming the existence of weak pseudorandom functions.

Theorem 1.3. *There does not exist a* $(0, \mathsf{poly}, \mathsf{poly})$*-simulation secure FE scheme for the class of (weak) pseudo-random functions in the random oracle model.*

Once again we interpret messages as seeds to a weak PRF family wPRF and secret keys as points in the domain of the PRF. A very different way to attack an FE scheme is needed though because no key query can be made before the challenge phase.

The new attacker $\mathcal{A}^\star$ starts off by outputting randomly chosen seeds $s_1, \ldots, s_k$ for wPRF, and gets ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$ in return. The RO queries made in the encryption process are *hidden* from $\mathcal{A}^\star$, and it might

[4] In fact, if we just want pre-challenge key query security, then there is no need for random oracle.

need some of them later during decryption. So, it requests secret keys for randomly chosen points $x_1, \ldots, x_q$, and gets $\mathsf{sk}_1, \ldots, \mathsf{sk}_q$ in return. Then it decrypts every ct_i with sk_j and records the RO queries made in a list Γ. An important point to note here is that the simulator gets some information about the seeds chosen earlier when key-queries are made. Specifically, it gets $\mathsf{wPRF}(s_1, x_j), \ldots, \mathsf{wPRF}(s_k, x_j)$ when x_j is the query.

$\mathcal{A}^\star$ now picks a random point x^* and requests a secret key for it. The goal is to use the key obtained, say sk^*, to decrypt the challenge ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$ later. But, in order to do so, $\mathcal{A}^\star$ also needs to find out the RO queries made during key-generation that may also be required for decryption. To solve this problem, we use the same idea as in the previous impossibility result: encrypt some random seeds on your own and decrypt them with sk^*, while adding the RO queries made to Γ.

Finally, $\mathcal{A}^\star$ decrypts $\mathsf{ct}_1, \ldots, \mathsf{ct}_k$ with sk^* *without* invoking the random oracle, using the list Γ instead. In the real world, at least a constant fraction of the decryptions succeed. The analysis is similar to that of the previous impossibility result, but with the role of ciphertext and key reversed. The ideal world analysis, on the other hand, need more care because of two reasons. First, as pointed out earlier, some information about the seeds $s_1, \ldots, s_k$ is leaked when post-challenge key queries are made. Second, the simulator needs to compress the evaluation of wPRF on seeds $s_1, \ldots, s_k$ and a common point x^*, instead of one seed and multiple points as in the $(\mathsf{poly}, 1, 0)$ impossibility. At the same time, however, the only opportunity a simulator has of programming RO responses after learning $\mathsf{wPRF}(s_1, x^*), \ldots, \mathsf{wPRF}(s_k, x^*)$ is when ciphertexts for random seeds are decrypted with sk^* with the help of RO. So, it is conceivable that one can exploit the security of wPRF to argue that it is impossible to compress $\mathsf{wPRF}(s_1, x^*), \ldots, \mathsf{wPRF}(s_k, x^*)$ into a small key and a small number of RO responses. We show that this is indeed the case in Section 7.

2 Preliminaries

We use λ to denote the security parameter. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$. If A is an algorithm, then $a \leftarrow A(\cdot)$ or $A(\cdot) \rightarrow a$ denote that a is the output of running A on the specified inputs. If $\mathcal{D}$ is a distribution, then $s \leftarrow \mathcal{D}$ denotes that s is a sample drawn according to it. Also, $x \xleftarrow{R} X$ denotes drawing a value x uniformly at random from the set X.

For two distribution ensembles $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda \in \mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda \in \mathbb{N}}$, we use $\mathcal{X} \stackrel{c}{\approx} \mathcal{Y}$ to denote that $\mathcal{X}$ is computationally indistinguishable from $\mathcal{Y}$. Lastly, for two vectors $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$, their Hamming distance $\mathsf{HD}(u, v)$ is defined to be the number of points where they don't match, i.e., the size of set $\{i \in [n] \mid u_i \neq v_i\}$.

2.1 Weak Pseudo-random Functions

Our impossibility results rely on the existence of circuit families whose output cannot be *compressed* by a significant amount. In Section 4, we will show that a specific circuit family built from pseudo-random functions (PRFs) is not compressible. In fact, like Gorbunov et al. [15], a weaker type of PRF where adversary only gets evaluation at random points suffices for our purpose.

Definition 2.1 (Weak PRFs). *Let n, m, p be polynomials in λ. Let $\mathsf{wPRF} = \{\mathsf{wPRF}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of efficiently computable functions such that $\mathsf{wPRF}_\lambda : \{0,1\}^{n(\lambda)} \times \{0,1\}^{m(\lambda)} \rightarrow \{0,1\}^{p(\lambda)}$, where the first input is called the seed. Pick a seed $s \xleftarrow{R} \{0,1\}^{n(\lambda)}$ and $\ell + 1$ points $x_1, \ldots, x_\ell, x^\star \xleftarrow{R} \{0,1\}^{m(\lambda)}$. Let D_ℓ be the ℓ-tuple of values $(x_1, \mathsf{wPRF}_\lambda(s, x_1)), \ldots, (x_\ell, \mathsf{wPRF}_\lambda(s, x_\ell))$. Then the wPRF family is a weak pseudo-random function family if for every ℓ polynomial in λ,*

$$\{D_\ell, x^\star, \mathsf{wPRF}_\lambda(s, x^\star)\}_{\lambda \in \mathbb{N}} \quad \stackrel{c}{\approx} \quad \{D_\ell, x^\star, r\}_{\lambda \in \mathbb{N}},$$

where r is a random string of length $p(\lambda)$.

Below we present two alternate definitions of security for a weak pseudorandom family. The first one is a standard definition for PRFs/weak PRFs, while the second one is introduced for our final impossibility result. They both follow from Definition 2.1 above through simple hybrid arguments.

Definition 2.2 (Weak PRFs, many points)**.** *Let* $\mathsf{wPRF} = \{\mathsf{wPRF}_\lambda\}_{\lambda\in\mathbb{N}}$ *be a family as in Definition 2.1. Pick* $s \xleftarrow{R} \{0,1\}^{n(\lambda)}$*,* $x_1, \dots, x_\ell \xleftarrow{R} \{0,1\}^{m(\lambda)}$*, and* $r_1, \dots, r_\ell \xleftarrow{R} \{0,1\}^{p(\lambda)}$*. Then the* wPRF *family is a weak PRF family for many points if for every* ℓ *polynomial in* λ*,*

$$\{(x_1, \mathsf{wPRF}_\lambda(s, x_1)), \dots, (x_\ell, \mathsf{wPRF}_\lambda(s, x_\ell))\}_{\lambda\in\mathbb{N}} \quad \stackrel{c}{\approx} \quad \{(x_1, r_1), \dots, (x_\ell, r_\ell)_{\lambda\in\mathbb{N}}.$$

Definition 2.3 (Weak PRFs, many seeds with aux)**.** *Let* $\mathsf{wPRF} = \{\mathsf{wPRF}_\lambda\}_{\lambda\in\mathbb{N}}$ *be a family as in Definition 2.1. Pick* k *seeds* $s_1, \dots, s_k \xleftarrow{R} \{0,1\}^{n(\lambda)}$ *and* $\ell+1$ *points* $x_1, \dots, x_\ell, x^\star \xleftarrow{R} \{0,1\}^{m(\lambda)}$*. Let* $D_{k,\ell}$ *be the* $k{\cdot}\ell$*-tuple of values* $(x_1, \mathsf{wPRF}_\lambda(s_1, x_1))$*, ...,* $(x_\ell, \mathsf{wPRF}_\lambda(s_1, x_\ell))$*, ...,* $(x_1, \mathsf{wPRF}_\lambda(s_k, x_1)), \dots, (x_\ell, \mathsf{wPRF}_\lambda(s_k, x_\ell))$*. Then the* wPRF *family is a weak PRF family for many seeds with auxiliary information if for every* k, ℓ *polynomial in* λ*,*

$$\{D_{k,\ell}, x^\star, \mathsf{wPRF}_\lambda(s_1, x^\star), \dots, \mathsf{wPRF}_\lambda(s_k, x^\star)\}_{\lambda\in\mathbb{N}} \quad \stackrel{c}{\approx} \quad \{D_{k,\ell}, x^\star, r_1, \dots, r_k\}_{\lambda\in\mathbb{N}},$$

where $r_1, \dots, r_k$ *are random strings of length* $p(\lambda)$*.*

2.2 Randomized Encodings

We use decomposable randomized encodings [15] to simplify the description of our FE schemes. They are known to exist for all circuits due to the works of [25, 3].

Definition 2.4 (Randomized Encodings)**.** *Let* $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ *be a family of circuits, where each circuit* $C \in \mathcal{C}_\lambda$ *takes an* $n(\lambda)$ *bit input and produces an* $m(\lambda)$ *bit output. A* decomposable randomized encoding RE *of* $\mathcal{C}$ *consists of two* PPT *algorithms:*

- $\mathsf{RE.Encode}(1^\lambda, C)$: *It takes a circuit* $C \in \mathcal{C}_\lambda$ *as input, and outputs a randomized encoding* $((w_{1,0}, w_{1,1}), \dots, (w_{n(\lambda),0}, w_{n(\lambda),1}))$*.*
- $\mathsf{RE.Decode}(1^\lambda, (\tilde{w}_1, \dots, \tilde{w}_{n(\lambda)}))$: *It takes an encoding* $(\tilde{w}_1, \dots, \tilde{w}_{n(\lambda)})$ *and outputs* $y \in \{0,1\}^{m(\lambda)} \cup \{\perp\}$*. for a circuit* $C \in \mathcal{C}_\lambda$ *evaluated at an* $x \in \{0,1\}^{n(\lambda)}$ *so that* $\tilde{w}_i = w_{i,x_i}$ *for all* $i \in [n(\lambda)]$ *(*x_i *denotes the* i*th bit of* x*), and outputs* $C(x)$*.*

Correctness *Let* $C \in \mathcal{C}_\lambda$ *be any circuit, and let* $((w_{1,0}, w_{1,1}), \dots, (w_{n,0}, w_{n,1})) \leftarrow \mathsf{RE.Encode}(1^\lambda, C)$*. For any input* $x \in \{0,1\}^{n(\lambda)}$*,* $\mathsf{RE.Decode}(1^\lambda, (w_{1,x_1}, \dots, w_{n(\lambda),x_{n(\lambda)}})) = C(x)$*.*

Security *To define the security of such a scheme, consider the following two distributions:*

- $\mathsf{Real}^{\mathsf{RE}}_{\mathcal{A}}(\lambda)$*. Run* $\mathcal{A}(1^\lambda)$ *to get a* $C \in \mathcal{C}_\lambda$ *and an* $x \in \{0,1\}^{n(\lambda)}$*. Then run* $\mathsf{RE.Encode}$ *on input* C *to get an encoding* $((w_{1,0}, w_{1,1}), \dots, (w_{n(\lambda),0}, w_{n(\lambda),1}))$*. Output* $\{w_{i,x_i}\}_{i\in[n(\lambda)]}$*.*
- $\mathsf{Ideal}^{\mathsf{RE}}_{\mathcal{S}}(\lambda)$*. Run* $\mathcal{A}(1^\lambda)$ *to get a* $C \in \mathcal{C}_\lambda$ *and an* $x \in \{0,1\}^{n(\lambda)}$*. Output* $\mathcal{S}(1^\lambda, C, C(x))$*.*

A randomized encoding scheme RE *is* secure *if for every* PPT *adversary* $\mathcal{A}$*, there exists a* PPT *simulator* $\mathcal{S}$ *such that*

$$\mathsf{Real}^{\mathsf{RE}}_{\mathcal{A}}(\lambda) \stackrel{c}{\approx} \mathsf{Ideal}^{\mathsf{RE}}_{\mathcal{S}}(\lambda).$$

3 Functional Encryption in the Random Oracle Model

A functional encryption scheme for a function space $\mathbb{F} = \{\mathbb{F}_\lambda\}_{\lambda\in\mathbb{N}}$ and a message space $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ in the random oracle model consists of four PPT algorithms that have access to a random oracle $\mathcal{O} : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{m(\lambda)}$, where ℓ and m are polynomials. The algorithms are described as follows:

- $\mathsf{Setup}^{\mathcal{O}}(1^\lambda)$: It takes the security parameter (in unary representation) as input and outputs a public key pk and a master secret key msk.
- $\mathsf{KeyGen}^{\mathcal{O}}(\mathsf{msk}, f)$: It takes the master secret key msk and a circuit $f \in \mathbb{F}_\lambda$ as inputs, and outputs a secret key sk_f for the circuit.

Experiment $\mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(\lambda)$:
1. $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}^{\mathcal{O}}(1^\lambda)$
2. $(\mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathsf{KeyGen\text{-}RO}(\mathsf{msk},\cdot,\cdot)}(\mathsf{pk})$
3. $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}^{\mathcal{O}}(\mathsf{mpk}, x_i)$ for $i \in [q_c]$
4. $\alpha \leftarrow \mathcal{A}_2^{\mathsf{KeyGen\text{-}RO}(\mathsf{msk},\cdot,\cdot)}(\{\mathsf{ct}_i\}_{i\in[q_c]}, \mathsf{st})$
5. Output α

Experiment $\mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A},\mathcal{S}}(\lambda)$:
1. $(\mathsf{pk}, \mathsf{st}') \leftarrow \mathcal{S}(1^\lambda)$
2. $(\mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathcal{S}(\cdot,\cdot)}(\mathsf{pk})$
3. $(\{\mathsf{ct}_i\}_{i\in[q_c]}) \leftarrow \mathcal{S}(\{f_j(x_i)\}_{i\in[q_c],j\in[q_1]})$ where $f_1, \ldots, f_{q_1}$ are key queries made by $\mathcal{A}_1$
4. $\alpha \leftarrow \mathcal{A}_2^{\mathcal{S}^{\mathsf{KeyIdeal}(\cdot)}(\cdot,\cdot)}(\{\mathsf{ct}_i\}_{i\in[q_c]}, \mathsf{st})$
5. Output α

Figure 1: Real and ideal experiments.

- $\mathsf{Encrypt}^{\mathcal{O}}(\mathsf{pk}, x)$: It takes the public key pk and a value $x \in \mathcal{X}_\lambda$ as inputs, and outputs a ciphertext ct_x.
- $\mathsf{Decrypt}^{\mathcal{O}}(\mathsf{pk}, \mathsf{sk}, \mathsf{ct})$: It takes the public key pk, a secret key sk, and a ciphertext ct as inputs, and outputs a value y or $\perp$.

Correctness. The four algorithms defined above must satisfy the following correctness property. For all values of the security parameter λ, for every $f \in \mathbb{F}_\lambda$ and $x \in \mathcal{X}_\lambda$, all random oracles $\mathcal{O}$, and all $(\mathsf{pk}, \mathsf{msk})$ output by $\mathsf{Setup}^{\mathcal{O}}(1^\lambda)$,

$$\mathsf{Decrypt}^{\mathcal{O}}(\mathsf{pk}, \mathsf{KeyGen}^{\mathcal{O}}(\mathsf{msk}, f), \mathsf{Encrypt}^{\mathcal{O}}(\mathsf{pk}, x)) = f(x).$$

Without loss of generality, we can assume Decrypt to be deterministic.

One could consider weaker notions of correctness where a negligible probability of error is allowed, but we chose to use the simpler notion of perfect correctness for ease of exposition.

3.1 Simulation-based Security

Definition 3.1 (Experiments)**.** *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *be a functional encryption scheme. For any* PPT *algorithms* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *and* $\mathcal{S}$*, Figure 1 defines two experiments* $\mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(\lambda)$ *and* $\mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A},\mathcal{S}}(\lambda)$*. In the figure,* q_c *denotes the length of challenge message vector* $\mathbf{x}$ *output by* $\mathcal{A}_1$ *and* q_1 *denotes the number of key generation queries made before that. The oracles* KeyGen-RO *and* KeyIdeal *work as follows:*

- KeyGen-RO *takes two inputs* inp_1 *and* inp_2*, where* inp_1 *specifies whether the query is a key generation query or a random oracle query. In the former case,* $\mathsf{KeyGen}^{\mathcal{O}}(\mathsf{msk}, \mathsf{inp}_2)$ *is invoked, while in the latter* $\mathcal{O}(\mathsf{inp}_2)$ *is invoked.*
- KeyIdeal *takes a function* f *as input and outputs* $(f(x_1), \ldots, f(x_{q_c}))$*.*

Definition 3.2 (Admissibility)**.** *An adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *is* $(q_{\mathsf{pre}}(\lambda), q_{\mathsf{chal}}(\lambda), q_{\mathsf{post}}(\lambda))$-admissible *if in any run of the experiments* $\mathsf{Real}_{\mathcal{A}}(1^\lambda)$ *and* $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$*,* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *make at most* $q_{\mathsf{pre}}(\lambda)$ *and* $q_{\mathsf{post}}(\lambda)$ *key generation queries, respectively, and* $\mathcal{A}_1$ *outputs at most* $q_{\mathsf{chal}}(\lambda)$ *challenge messages.*

An adversary $\mathcal{A}$ *is* $(\mathsf{poly}, \mathsf{q}_{\mathsf{chal}}(\lambda), \mathsf{q}_{\mathsf{post}}(\lambda))$-admissible *if in any run of the experiments* $\mathsf{Real}_{\mathcal{A}}(1^\lambda)$ *and* $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(1^\lambda)$*,* $\mathcal{A}_1$ *is allowed to make an unbounded (but polynomial) number of pre-challenge key queries,* $\mathcal{A}_2$ *makes at most* $q_{\mathsf{post}}(\lambda)$ *key generation queries, and* $\mathcal{A}_1$ *outputs at most* $q_{\mathsf{chal}}(\lambda)$ *challenge messages. We can similarly define admissible adversaries where the number of challenge messages/post challenge key queries are unbounded.*

On the other hand, a simulator $\mathcal{S}$ *is* admissible *if whenever* $\mathcal{A}_2$ *makes a key query* f*,* $\mathcal{S}$ *queries* KeyIdeal *on* f *only.*

Definition 3.3 (Simulation security)**.** *A functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is* $(q_{\mathsf{pre}}(\lambda), q_{\mathsf{chal}}(\lambda), q_{\mathsf{post}}(\lambda))$-Sim*-secure for some polynomials* q_{pre}*,* q_{chal}*, and* q_{post}*, if there exists an admissible* PPT *simulator* $\mathcal{S}$ *such that for all* $(q_{\mathsf{pre}}(\lambda), q_{\mathsf{chal}}(\lambda), q_{\mathsf{post}}(\lambda))$*-admissible* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*,*

$$\{\mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(\lambda)\}_{\lambda\in\mathbb{N}} \stackrel{c}{\approx} \{\mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A},\mathcal{S}}(\lambda)\}_{\lambda\in\mathbb{N}}.$$

We also consider security notions that allow an unbounded (but polynomial) number of pre-challenge key queries/challenge messages/post-challenge key queries.

Definition 3.4 (Simulation security, unbounded queries)**.** *A functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is* $(\mathsf{poly}, \mathsf{q_{chal}}(\lambda), \mathsf{q_{post}}(\lambda))$-$\mathsf{Sim}$-*secure for some polynomials* q_{chal}*, and* q_{post}*, if there exists an admissible* PPT *simulator* $\mathcal{S}$ *such that for all* $(\mathsf{poly}, \mathsf{q_{chal}}(\lambda), \mathsf{q_{post}}(\lambda))$-*admissible* PPT *adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$,

$$\{\mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(\lambda)\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \{\mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A},\mathcal{S}}(\lambda)\}_{\lambda\in\mathbb{N}}.$$

We can similarly define simulation security when q_{chal} *and* q_{post} *are unbounded.*

Note that in the real world an adversary has explicit access to the random oracle. In the ideal world, both the key generation and random oracles are simulated by $\mathcal{S}$ throughout the experiment. This makes the simulator stronger and our security definition weaker.

4 Hardness of Approximate Compression

In this section, we will first define the notion of approximate compression, and then show that there are certain circuit families which are hard to approximately compress. This section closely follows the work of Agrawal et al. [1], who defined the notion of (exact) compressibility of circuit evaluations, and showed that there exist certain circuit families that are (exact) incompressible.

Definition 4.1. *Let* ℓ*,* t *be polynomials and* ϵ *a non-negligible function. A class of circuits* $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ *with domain* $\mathcal{D} = \{\mathcal{D}_\lambda\}_\lambda$ *and range* $\mathcal{R} = \{\mathcal{R}_\lambda\}_\lambda$ *is said to be* (ℓ, t, ϵ)*-approximately compressible if there exists a family of compression circuits* $\mathsf{Cmp} = \{\mathsf{Cmp}_\lambda\}_\lambda$*, a family of decompression circuits* $\mathsf{DeCmp} = \{\mathsf{DeCmp}_\lambda\}_\lambda$*, a polynomial* poly*, and a non-negligible function* η*, such that for all large enough* λ *the following properties hold:*

- *The circuits* Cmp_λ *and* DeCmp_λ *have size bounded by* $\mathsf{poly}(\lambda)$.
- *(compression) For all input* $s \in \mathcal{D}_\lambda$ *and circuits* $C_1, C_2, \ldots, C_{\ell(\lambda)} \in \mathcal{C}_\lambda$*,*

$$\left|\mathsf{Cmp}_\lambda\left(\{C_i, C_i(s)\}_{i\in[\ell(\lambda)]}\right)\right| \le t(\lambda).$$

- *(approximate decompression) If* s *is chosen at random from* $\mathcal{D}_\lambda$*,* $C_1, C_2, \ldots, C_{\ell(\lambda)}$ *are chosen uniformly and independently from* $\mathcal{C}_\lambda$*, then*

$$\Pr\Big[\mathsf{HD}\Big(\mathsf{DeCmp}_\lambda\Big(\{C_i\}_{i\in[\ell(\lambda)]}, \mathsf{Cmp}_\lambda\Big(\{C_i, C_i(s)\}_{i\in[\ell(\lambda)]}\Big)\Big), (C_1(s), \ldots, C_{\ell(\lambda)}(s))\Big) \le \epsilon(\lambda)\cdot t(\lambda)\Big] \ge \eta(\lambda)$$

We will now show that weak PRFs can be used to construct a class of circuits that are not approximate compressible. We will then use the more general notion of approximate incompressibility, rather than the specific case of weak PRFs, in proving our impossibility results. For simplicity of presentation, in the lemma statement below, we use specific constants which will be sufficient for our main result. However, the lemma can be easily extended to work for general ℓ, t and ϵ. We assume that the weak PRF outputs a single bit.

Lemma 4.1. *Let* $\mathsf{wPRF} = \{\mathsf{wPRF}_\lambda\}_\lambda$ *be a family of weak pseudorandom functions (for many points), where* $\mathsf{wPRF}_\lambda : \{0,1\}^{n(\lambda)} \times \{0,1\}^{m(\lambda)} \to \{0,1\}$*. Consider the family of circuits* $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$*, where* $\mathcal{C}_\lambda = \{\mathsf{wPRF}_\lambda(\cdot, x)\}_{x\in\{0,1\}^{m(\lambda)}}$*. Let* $t = t(\lambda)$ *be any polynomial such that* $t(\lambda) \ge \lambda$ *for all* $\lambda \in \mathbb{N}$*. Then* $\mathcal{C}$ *is not* $(16t, t, 1/8)$ *approximate compressible.*

Proof. Suppose, on the contrary, that the circuit family $\mathcal{C}$ is $(16t, t, 1/8)$ approximate compressible. We will use the compression circuits $\{\mathsf{Cmp}_\lambda\}_\lambda$ and decompression circuits $\{\mathsf{DeCmp}_\lambda\}_\lambda$ to break the weak PRF security of wPRF. Fix any large enough security parameter λ. For simplicity of presentation, we will drop the dependence on λ when it is clear from the context.

Suppose we are given $16t$ tuples $\{x_i, y_i\}_{i\in[16t]}$ that are either generated through wPRF or chosen uniformly at random. Define $16t$ circuits $C_1, \ldots, C_{16t}$, where $C_i(\cdot) = \mathsf{wPRF}(\cdot, x_i)$. Compute the compressed string $u = \mathsf{Cmp}\left(\{C_i, y_i\}_{i\in[16t]}\right)$ and $z = \mathsf{DeCmp}\left(\{C_i\}_{i\in[16t]}, u\right)$. If $\mathsf{HD}\left(z, (y_1 \ldots y_{16t})\right) \leq 1/8\,(16t)$, output 'pseudorandom', else output 'truly random'.

Below we show that if the y_i values are generated through wPRF, i.e., when they are pseudorandom, then 'pseudorandom' is output with a non-negligible probability (Claim 4.1). However, if the y_i values are truly random, then the same output is produced with negligible probability (Claim 4.2). Thus we are able to break the security of wPRF, leading to a contradiction. ∎

Claim 4.1. $\Pr\left[\textit{Output is 'pseudorandom'} \mid \{y_i\} \textit{ are pseudorandom}\right] \geq \eta$ *for some non-negligible function* η.

Proof. For a randomly chosen seed $s \leftarrow \{0,1\}^n$, suppose $y_i = \mathsf{wPRF}(s, x_i) = C_i(s)$ for all $i \in [16t]$. Due to the approximate decompression property, there exists a non-negligible function η such that

$$\Pr\left[\mathsf{HD}\left(\mathsf{DeCmp}\left(\{C_i\}_{i\leq 16t}, \mathsf{Cmp}\left(\{C_i, C_i(s)\}_{i\in[16t]}\right)\right), (C_1(s), \ldots, C_{16t}(s))\right) \leq 1/8\,(16t)\right] \geq \eta$$

where the probability is over the choice of s and $x_1, \ldots, x_m$. Thus 'pseudorandom' is output with at least η probability. ∎

Claim 4.2. $\Pr\left[\textit{Output is 'pseudorandom'} \mid \{y_i\} \textit{ are truly random}\right] \leq \mathsf{negl}$.

Proof. Fix any $x_1, x_2, \ldots, x_{16t}$, which also fixes the circuits $C_1, C_2, \ldots, C_{16t}$. Now,

$$\begin{aligned}
&\Pr\left[\text{Output is 'pseudorandom'} \mid \{y_i\} \text{ are truly random}\right]\\
&\quad \leq \Pr\left[\exists\, z \text{ s.t. } \mathsf{HD}\left(\mathsf{DeCmp}\left(\{C_i\}, z\right), (y_1, \ldots, y_{16t})\right) \leq 1/8\,(16t)\right]\\
&\quad \leq \sum_{z\in\{0,1\}^t} \Pr\left[\mathsf{HD}\left(\mathsf{DeCmp}\left(\{C_i\}, z\right), (y_1, \ldots, y_{16t})\right) \leq 1/8\,(16t)\right]\\
&\quad \leq \sum_{z\in\{0,1\}^t} \binom{16t}{2t} \cdot 2^{-14t}\\
&\quad \leq \sum_{z\in\{0,1\}^t} \left(\frac{16\cdot e}{2}\right)^{2t} \cdot 2^{-14t}\\
&\quad < \sum_{z\in\{0,1\}^t} 2^{-2t}\\
&\quad = 2^{-t}
\end{aligned}$$

Here, the second inequality is a simple union bound. The third inequality follows from the fact that the y_i values are chosen independent of the C_is and the string z. ∎

5 Impossibility of Simulation Secure FE

In this section we show that there does not exist a functional encryption scheme for the family of all polynomial-sized circuits that is $(\mathsf{poly}, 1, 0)$-Sim secure in the random oracle model. Specifically, we show that a simulation secure FE scheme cannot be constructed for any family of circuits that is not approximately compressible (Definition 4.1). We exhibit an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that for *any* efficient simulator $\mathcal{S}$, the output of the real experiment, $\mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda)$, is *distinguishable* from the output of the ideal experiment, $\mathsf{Ideal}^{\mathsf{FE}}_{\mathcal{A},\mathcal{S}}(1^\lambda)$ (Definition 3.4).

High level description of adversary. Let $\mathcal{C}$ be an approximate incompressible circuit family. The adversary $\mathcal{A}_1$ first asks for secret keys for a large number of randomly chosen circuits from C, and receives $\{\mathsf{sk}_1, \ldots, \mathsf{sk}_q\}$ in return. Next, it generates encryptions of many random messages. It then decrypts each of these ciphertexts using the q secret keys. The purpose of these encryptions followed by the decryptions is to capture the random oracle queries that would have occurred while computing the q secret keys, which may also be required when these keys are used again for decryption later. Let S_{keys} denote the set of random oracle queries that occur during these decryptions.

$\mathcal{A}_1$ chooses a random message x^*, and outputs it as the challenge (along with a state that consists of its view so far). $\mathcal{A}_2$ then receives a ciphertext ct^*. It decrypts ct^* using $\mathsf{sk}_1, \ldots, \mathsf{sk}_t$, for some small t. Let S_{ct^*} denote the set of random oracle queries during these t decryptions. The purpose of these t decryptions is to capture the random oracle queries that would have occurred during the encryption of x^*, which may also be required when ct^* is decrypted again in the next step.

Finally, $\mathcal{A}_2$ decrypts ct^* using the remaining $q - t$ secret keys. An important thing to note here is that $\mathcal{A}_2$ *turns off* the random oracle, and instead uses the queries that it has already recorded. If a new random oracle query is required, then it uses a randomly chosen string. It compares the decrypted values to the correct function evaluations, and outputs 1 if most decryptions are correct.

First, we show that in the real world, $\mathcal{A}_2$ outputs 1 with probability at least $3/4$. Let us focus on one of the $q - t$ decryptions, using a secret key sk_j. At a high level, this decryption can go wrong if a random oracle query is made on z, and $z \notin S_{\text{keys}} \cup S_{\mathsf{ct}}$, but z was used during the computation of either sk_j or ct. We show that this event happens with low probability.

To complete the argument, we show that in the ideal world, $\mathcal{A}_2$ outputs 1 with probability around $1/2$. In this world, the simulator receives q circuit evaluations on x^*, and must compress most of this information in the short challenge ciphertext and the random oracle queries made during the t post-challenge decryption operations. By choosing parameters carefully and appealing to the (approximate) incompressibility of the circuit family, we show that this is not possible.

5.1 Formal Description of Adversary

Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ be a family of circuits such that each circuit in $\mathcal{C}_\lambda$ takes an $n(\lambda)$-bit input and is not $(16t, t, 1/8)$ approximately compressible for all polynomials t such that $t(\lambda) \geq \lambda$. Let FE be a functional encryption scheme for this family in the random oracle model. We now formally define the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

Adversary $\mathcal{A}_1$. Let n_{key} and n_{enc} be polynomials in λ whose values will be fixed later. Let Γ be a list of (query, response) pairs that is empty at the beginning. $\mathcal{A}_1$ has four phases: setup, key query, random oracle query collection, and an output phase.

1. **Setup.** $\mathcal{A}_1$ receives the public key pk.

2. **Key query.** For $i \in [n_{\mathsf{key}}]$, it picks a circuit C_i at random from $\mathcal{C}_\lambda$, requests a secret key for C_i, and obtains sk_i in return.

3. **RO query collection 1.** $\mathcal{A}_1$ picks n_{enc} inputs $x_1, x_2, \ldots, x_{n_{\mathsf{enc}}} \xleftarrow{R} \{0,1\}^{n(\lambda)}$. For $j \in [n_{\mathsf{enc}}]$, it runs $\mathsf{Encrypt}^{\mathcal{O}}(\mathsf{pk}, x_j)$ to obtain a ciphertext ct_j. The RO queries made during the encryption process are forwarded to the random oracle.

   Now each of the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_{n_{\mathsf{enc}}}$ are decrypted with key sk_i for every $i \in [n_{\mathsf{key}}]$. If an oracle query β is made by the Decrypt algorithm, $\mathcal{A}_1$ queries the random oracle with the same. The response, say γ, is given to the algorithm, and (β, γ) is added to Γ (if it is not already present).

4. **Output.** $\mathcal{A}_1$ picks an input $x^* \xleftarrow{R} \{0,1\}^{n(\lambda)}$. It sets the state st to consist of pk, $C_1, \ldots, C_{n_{\mathsf{key}}}$, $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{key}}}$, x^*, and Γ. Then it outputs (x^*, st).

Adversary $\mathcal{A}_2$. Let n_{eval} and n_{test} be polynomials in λ s.t. $n_{\mathsf{eval}}(\lambda) + n_{\mathsf{test}}(\lambda) = n_{\mathsf{key}}(\lambda)$ for all λ. (Their values will be fixed later.) $\mathcal{A}_2$ gets ct^* and st as input, and parses the latter to get pk, $C_1, \ldots, C_{n_{\mathsf{key}}}$, $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{key}}}$, x^*, and Γ. $\mathcal{A}_2$ has three phases: random oracle query collection, test, and an output phase.

1. **RO query collection 2.** For every $i \in [n_{\mathsf{eval}}]$, ct^* is decrypted with sk_i. If an RO query β is made by the Decrypt algorithm, $\mathcal{A}_2$ queries the random oracle with the same. The response, say γ, is given to the algorithm, and (β, γ) is added to Γ (if it is not already present).

2. **Test.** In this phase, ct^* is decrypted with rest of the keys but *without* invoking the random oracle. In order to do so, a new list Δ is initialized first, then the following steps are executed for every $n_{\mathsf{eval}} + 1 \leq i \leq n_{\mathsf{eval}} + n_{\mathsf{test}}$. The decryption algorithm is run with inputs pk, sk_i, and ct^*. When it makes an RO query β, $\mathcal{A}_2$ checks whether there is an entry of the form (β, γ) in Γ or Δ (in that order) or not. If yes, then γ is given to Decrypt and it continues to run. Otherwise, a random bit-string γ' of length $m(\lambda)$ (the output length of the random oracle) is generated, (β, γ') is added to Δ, and γ' is given to Decrypt. This process of providing responses to the RO queries of Decrypt continues till it terminates. Let out_i denote the output of Decrypt, which could be $\perp$.

3. **Output.** For every $n_{\mathsf{eval}} + 1 \leq i \leq n_{\mathsf{eval}} + n_{\mathsf{test}}$, check if out_i is equal to $C_i(x^*)$ (where x^* and C_i are part of the state transferred to $\mathcal{A}_2$). Let num be the number of keys for which this check succeeds. Output 1 if $\mathsf{num}/n_{\mathsf{test}} \geq 7/8$, else output 0.

To complete the description of $\mathcal{A}$, we need to define the polynomials n_{enc}, n_{eval} and n_{test} (recall that $n_{\mathsf{key}} = n_{\mathsf{eval}} + n_{\mathsf{test}}$). Let q_{Setup}, q_{Enc}, q_{KeyGen} and q_{Dec} be upper-bounds on the number of RO queries made by Setup, Encrypt, KeyGen and Decrypt, respectively, as a function of λ. Also, let ℓ_{ct} be an upper-bound on the length of ciphertexts generated by Encrypt. Then set

- $n_{\mathsf{enc}} = 4\lambda \cdot n_{\mathsf{key}} \cdot q_{\mathsf{KeyGen}}$,
- $n_{\mathsf{eval}} = 32\lambda\,(q_{\mathsf{Setup}} + q_{\mathsf{Enc}})$,
- $n_{\mathsf{test}} = 16(\ell_{\mathsf{ct}} + n_{\mathsf{eval}} \cdot q_{\mathsf{Dec}} \cdot m)$.

5.2 Real World Analysis

First, we will show that the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ described above outputs 1 with probability at least $3/4$ in the real world experiment, as long as the scheme FE is correct. To begin with, we classify the random oracle queries made during a run of $\mathcal{A}$ into different sets as follows:

- $\mathsf{S\text{-}RO}_{C_i}$ for $i \in [n_{\mathsf{key}}]$: random oracle queries made by KeyGen while generating secret key for C_i.
- $\mathsf{S\text{-}RO}_{\mathrm{keys}} = \bigcup_{i \in [n_{\mathsf{key}}]} \mathsf{S\text{-}RO}_{C_i}$: all random oracle queries during the key query phase of $\mathcal{A}_1$.
- $\mathsf{S\text{-}RO}_{x^*}$: random oracle queries made while encrypting x^* using pk.
- $\mathsf{S\text{-}RO}_{\mathrm{Dec}\text{-}i}$ for $i \in [n_{\mathsf{test}}]$: random oracle queries made during the decryption of ct^* using $\mathsf{sk}_{n_{\mathsf{eval}}+i}$.
- $\mathsf{S\text{-}RO}_{\Gamma\text{-}b}$: random oracle queries recorded during 'RO Collection Phase b' for $b \in \{1, 2\}$. Let $\mathsf{S\text{-}RO}_{\Gamma} = \mathsf{S\text{-}RO}_{\Gamma\text{-}1} \bigcup \mathsf{S\text{-}RO}_{\Gamma\text{-}2}$.
- $\mathsf{S\text{-}RO}_{\mathsf{Setup}}$: random oracle queries made during setup phase.

Lemma 5.1. *For any functional encryption scheme* FE *for the circuit family* $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$*, the adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *described in Section 5.1 outputs* 1 *in* $\mathsf{Real}^{\mathsf{FE}}_{\mathcal{A}}(1^\lambda)$ *with probability at least* $3/4 - \mathsf{negl}(\lambda)$.

Proof. We will use the correctness property of FE to prove this claim. Recall that, for simplicity, we assume correctness to be perfect, i.e., for all random oracles $\mathcal{O} : \{0,1\}^{\ell(\lambda)} \to \{0,1\}^{m(\lambda)}$, $x \in \{0,1\}^{n(\lambda)}$, $C \in \mathcal{C}_\lambda$ and $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}^{\mathcal{O}}(1^\lambda)$,

$$\mathsf{Decrypt}^{\mathcal{O}}\left(\mathsf{pk}, \mathsf{KeyGen}^{\mathcal{O}}(\mathsf{msk}, C), \mathsf{Encrypt}^{\mathcal{O}}(\mathsf{pk}, x)\right) = C(x).$$

Thus, the decryption algorithm can be assumed to be deterministic without loss of generality.

Let Bad denote the event that the adversary outputs 0 at the end of the real world experiment. This event happens if at least $1/8$th fraction of the n_{test} decryptions fail in the test phase. If $\mathsf{I\text{-}Dec}_i$ is an indicator variable that takes the value 1 in case the ith decryption *fails*, then Bad happens iff $\sum_{i\in[n_{\mathsf{test}}]} \mathsf{I\text{-}Dec}_i > 1/8 \cdot n_{\mathsf{test}}$. To analyze the probability of this event, we need to consider the random oracle queries required for decryption in the test phase. In this phase, $\mathcal{A}_2$ does not query the random oracle, but instead uses the list Γ. If some query β is not present in Γ, then $\mathcal{A}_2$ tries to find it in Δ. If β is not found in Δ either, then a random value is chosen and recorded in Δ against β. Now the only way ith decryption fails is if there is some entry (β, γ) in Δ such that β is also among the RO queries *hidden* from the adversary (and its response is not γ), i.e., the queries made during the setup phase, key query phase or challenge ciphertext generation. In other words, the ith decryption succeeds with certainty if all the *needed* hidden RO queries are captured in either of the two RO collection phases. This is formalized in the following observation.

Observation 5.1. *For every $i \in [n_{\mathsf{test}}]$, if the decryption of ct^* using $\mathsf{sk}_{n_{\mathsf{eval}}+i}$ does not give $C_i(x^*)$, i.e. $\mathsf{I\text{-}Dec}_i = 1$, then $\mathsf{S\text{-}RO}_{\text{Dec-}i} \bigcap (\mathsf{S\text{-}RO}_{\mathsf{Setup}} \bigcup \mathsf{S\text{-}RO}_{\text{keys}} \bigcup \mathsf{S\text{-}RO}_{x^*}) \not\subseteq \mathsf{S\text{-}RO}_{\Gamma}$.*

Let $\mathsf{I\text{-}Dec\text{-}1}_i$ and $\mathsf{I\text{-}Dec\text{-}2}_i$ be indicator variables that are 1 iff $\mathsf{S\text{-}RO}_{\text{Dec-}i} \bigcap (\mathsf{S\text{-}RO}_{x^*} \bigcup \mathsf{S\text{-}RO}_{\mathsf{Setup}}) \not\subseteq \mathsf{S\text{-}RO}_{\Gamma}$ and $\mathsf{S\text{-}RO}_{\text{Dec-}i} \bigcap \mathsf{S\text{-}RO}_{\text{keys}} \not\subseteq \mathsf{S\text{-}RO}_{\Gamma}$, respectively. Then, $\mathsf{I\text{-}Dec}_i = 1$ iff either $\mathsf{I\text{-}Dec\text{-}1}_i = 1$ or $\mathsf{I\text{-}Dec\text{-}2}_i = 1$ (or both). Let $\mathsf{Bad\text{-}1}$ and $\mathsf{Bad\text{-}2}$ be events that happen iff $\sum_{i\in[n_{\mathsf{test}}]} \mathsf{I\text{-}Dec\text{-}1}_i > 1/16 \cdot n_{\mathsf{test}}$ and $\sum_{i\in[n_{\mathsf{test}}]} \mathsf{I\text{-}Dec\text{-}2}_i > 1/16 \cdot n_{\mathsf{test}}$, respectively. It is easy to see then that whenever Bad happens, at least one of $\mathsf{Bad\text{-}1}$ and $\mathsf{Bad\text{-}2}$ also happen. That is, $\Pr[\mathsf{Bad}] \leq \Pr[\mathsf{Bad\text{-}1}] + \Pr[\mathsf{Bad\text{-}2}]$. Below we show that $\Pr[\mathsf{Bad\text{-}1}] \leq \mathsf{negl}(\lambda)$ and $\Pr[\mathsf{Bad\text{-}2}] \leq 1/4$. Thus the lemma follows. ∎

Claim 5.1. $\Pr[\mathsf{Bad\text{-}1}] \leq \mathsf{negl}(\lambda)$.

Proof. Fix any random oracle $\mathcal{O}$, the randomness used in $\mathsf{Setup}^{\mathcal{O}}(1^\lambda)$, challenge message x^*, and the randomness used in $\mathsf{Encrypt}^{\mathcal{O}}(\mathsf{pk}, x^*)$. This also fixes the sets $\mathsf{S\text{-}RO}_{\mathsf{Setup}}$ and $\mathsf{S\text{-}RO}_{x^*}$. Suppose a circuit C is picked at random from $\mathcal{C}_\lambda$, and a key, sk, is generated for it by running $\mathsf{KeyGen}^{\mathcal{O}}(\mathsf{msk}, C)$. For $z \in \mathsf{S\text{-}RO}_{\mathsf{Setup}} \cup \mathsf{S\text{-}RO}_{x^*}$, let ρ_z be the probability that z is an RO query in the decryption of ct^* (the challenge ciphertext) with sk, where the probability is over the choice of C and the randomness used in KeyGen.

Let $X_{i,z}$ be an indicator variable that is 1 if an RO query on z is made during the ith decryption in post-challenge phase (either in the RO collection 2 or test phase). Note that the keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{key}}}$ are generated independently by choosing circuits $C_1, \ldots, C_{n_{\mathsf{key}}}$ uniformly at random. Thus for any z, the variables $X_{1,z}, \ldots, X_{n_{\mathsf{key}},z}$ are independent of each other, and $\Pr[X_{i,z} = 1] = \rho_z$ for every i.

We are interested in the probability that $\sum_{i\in[n_{\mathsf{test}}]} \mathsf{I\text{-}Dec\text{-}1}_i > n_{\mathsf{test}}/16$, i.e., in at least $1/16$th fraction of the decryptions in the test phase, an RO query q is made s.t. q was also an RO query in either set-up or encryption of x^*, but it was not captured in either of the collection phases. Thus, there must exist a z s.t. $z \notin \mathsf{S\text{-}RO}_{\Gamma}$ (in particular, $z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}2}$) but an RO query on z is made in at least $n_{\mathsf{test}}/16|Q|$ of the decryptions, where $Q = \mathsf{S\text{-}RO}_{\mathsf{Setup}} \cup \mathsf{S\text{-}RO}_{x^*}$. (If $Q = \phi$ then $\mathsf{Bad\text{-}1}$ cannot happen, and we are done.) Therefore,

$$\Pr\left[\sum_{i\in[n_{\mathsf{test}}]} \mathsf{I\text{-}Dec\text{-}1}_i > \frac{n_{\mathsf{test}}}{16}\right] \quad \leq \quad \sum_{z\in Q} \Pr\left[z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}2} \quad \wedge \quad \sum_{i\in[n_{\mathsf{test}}]} X_{i,z} > \frac{n_{\mathsf{test}}}{16|Q|}\right]$$

Based on the value of ρ_z, we can divide rest of the analysis into two parts. Intuitively, if ρ_z is large, then the probability that z is not captured during RO collection phase is negligible. And when it is small, the probability that z causes too many decryptions to fail in the test phase is negligible. Since Q is polynomial in the security parameter, this will prove that the probability of $\mathsf{Bad\text{-}1}$ is negligible as well. So now,

- If $\rho_z \geq 1/32|Q|$ then

$$\begin{aligned}\Pr[z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}2}] &= \Pr[X_{1,z} = 0 \wedge \ldots \wedge X_{n_{\mathsf{eval}},z} = 0] \\ &= \prod_{i\in[n_{\mathsf{eval}}]} \Pr[X_{i,z} = 0] \\ &= (1-\rho_z)^{n_{\mathsf{eval}}} \leq e^{-n_{\mathsf{eval}}/32|Q|},\end{aligned}$$

  where the second equality follows from the independence of $X_{i,z}$. Recall that we set n_{eval} to be $32\lambda(q_{\mathsf{Setup}} + q_{\mathsf{Enc}})$, where q_{Setup} and q_{Enc} are upper-bounds on the number of RO queries made during Setup and $\mathsf{Encrypt}$, respectively. Thus, $e^{-n_{\mathsf{eval}}/32|Q|}$ is at most $e^{-\lambda}$.

- If $\rho_z < 1/32|Q|$ then expected value of $\sum_{i\in[n_{\mathsf{test}}]} X_{i,z}$ is at most $n_{\mathsf{test}}/32|Q|$. Using Chernoff bounds we can argue that,

$$\Pr\left[\sum_{i\in[n_{\mathsf{test}}]} X_{i,z} > \frac{n_{\mathsf{test}}}{16|Q|}\right] < e^{-\frac{1}{3}\cdot\frac{n_{\mathsf{test}}}{32|Q|}}.$$

We know that $n_{\mathsf{test}} \geq n_{\mathsf{eval}}$. Thus, $e^{-\frac{1}{3}\cdot\frac{n_{\mathsf{test}}}{32|Q|}}$ is at most $e^{-\lambda}$ as well.

■

Claim 5.2. $\Pr[\mathsf{Bad\text{-}2}] \leq 1/4$.

Proof. Fix any random oracle $\mathcal{O}$, the randomness used in $\mathsf{Setup}^{\mathcal{O}}(1^\lambda)$, the circuits $C_1, \ldots, C_{n_{\mathsf{key}}}$ chosen in the key query phase, and the randomness used in $\mathsf{KeyGen}^{\mathcal{O}}(\mathsf{msk}, C_i)$ for $i \in [n_{\mathsf{key}}]$. This, in particular, fixes secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{key}}}$ and the set $\mathsf{S\text{-}RO}_{\mathrm{keys}}$. Consider the following experiment: $x \xleftarrow{R} \{0,1\}^{n(\lambda)}$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}^{\mathcal{O}}(\mathsf{pk}, x)$, and decrypt ct using sk_i for $i \in [n_{\mathsf{eval}} + 1, n_{\mathsf{key}}]$. Let $\hat{\rho}_z$ be the probability that at least $n_{\mathsf{test}}/16|\hat{Q}|$ of the decryptions make an RO query on z, where $\hat{Q} = \mathsf{S\text{-}RO}_{\mathrm{keys}}$.

Let $Y_{j,z}$ be an indicator variable that is 1 iff an RO query on z is made in at least $n_{\mathsf{test}}/16|\hat{Q}|$ of the decryptions of ct_j with keys $\mathsf{sk}_{n_{\mathsf{eval}}+1}, \ldots, \mathsf{sk}_{n_{\mathsf{key}}}$ in the first phase of RO query collection. Note that the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_{n_{\mathsf{enc}}}$ are generated independently by choosing $x_1, \ldots, x_{n_{\mathsf{key}}}$ uniformly at random. Thus for any z, the variables $Y_{1,z}, \ldots, Y_{n_{\mathsf{enc}},z}$ are independent of each other, and $\Pr[Y_{j,z} = 1] = \hat{\rho}_z$ for every j. In a similar way, we can also define a random variable Y_z^* that indicates whether an RO query on z is made in at least $n_{\mathsf{test}}/16|\hat{Q}|$ of the decryptions of ct^* with keys $\mathsf{sk}_{n_{\mathsf{eval}}+1}, \ldots, \mathsf{sk}_{n_{\mathsf{key}}}$ in the test phase. Y_z^* is independent of $Y_{1,z}, \ldots, Y_{n_{\mathsf{enc}},z}$ and $\Pr[Y_z^* = 1] = \hat{\rho}_z$.

In a manner similar to the previous claim, we can argue that

$$\Pr\left[\sum_{i\in[n_{\mathsf{test}}]} \mathsf{I\text{-}Dec\text{-}2}_i > \frac{n_{\mathsf{test}}}{16}\right] \quad \leq \quad \sum_{z\in\hat{Q}} \Pr\left[z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}1} \wedge Y_z^* = 1\right]$$

If $z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}1}$, then none of the decryptions in the first phase of RO collection make a query on z. In particular, the variables $Y_{1,z}, \ldots, Y_{n_{\mathsf{enc}},z}$ are all zero in such a case. Therefore,

$$\begin{aligned}\Pr\left[z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}1} \wedge Y_z^* = 1\right] &\leq \Pr\left[Y_{1,z} = 0 \wedge \ldots \wedge Y_{n_{\mathsf{enc}},z} = 0 \wedge Y_z^* = 1\right] \\ &= \Pr\left[Y_z^* = 1\right] \quad \cdot \prod_{j\in[n_{\mathsf{enc}}]} \Pr\left[Y_{j,z} = 0\right] \\ &= \hat{\rho}_z(1-\hat{\rho}_z)^{n_{\mathsf{enc}}}\end{aligned}$$

Once again we have two cases. If $\hat{\rho}_z \leq 1/4|\hat{Q}|$, then $\hat{\rho}_z(1-\hat{\rho}_z)^{n_{\mathsf{enc}}}$ is at most $1/4|\hat{Q}|$ as well. Otherwise, $(1-\hat{\rho}_z)^{n_{\mathsf{enc}}} \leq e^{-n_{\mathsf{enc}}/4|\hat{Q}|} \leq e^{-\lambda}$ because, recall that, n_{enc} is set to be $4\lambda \cdot n_{\mathsf{key}} \cdot q_{\mathsf{KeyGen}}$, where q_{KeyGen} is an upper-bound on the number of RO queries made during KeyGen. As a result, $\sum_{z\in\hat{Q}} \hat{\rho}_z(1-\hat{\rho}_z)^{n_{\mathsf{enc}}}$ is at most $1/4$. ■

5.3 Ideal world analysis

Next, we will show that any for PPT simulator, our adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ outputs 1 in the ideal world with negligible probability. Let t be a polynomial in λ such that $t = \ell_{\mathsf{ct}} + n_{\mathsf{eval}} \cdot q_{\mathsf{Dec}} \cdot m$ (so that $n_{\mathsf{test}} = 16t$) where, recall that, ℓ_{ct} is the maximum length of any ciphertext generated by Encrypt. Note that $q_{\mathsf{Dec}} \cdot m$ is the maximum number of bits obtained through the random oracle during any decryption, $n_{\mathsf{eval}} \cdot q_{\mathsf{Dec}} \cdot m$ is the maximum number of bits sent to the adversary during the second RO query collection phase, and $\ell_{\mathsf{ct}} + n_{\mathsf{eval}} \cdot q_{\mathsf{Dec}} \cdot m$ is the total number of bits the adversary receives after sending the challenge message.

Lemma 5.2. *If $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ is an $(16t, t, 1/8)$ approximately incompressible circuit family, then for any* PPT *simulator $\mathcal{S}$, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ outputs 1 with probability at most* $\mathsf{negl}(\lambda)$.

Proof. Suppose there exists a simulator $\mathcal{S}$ such that our adversary $\mathcal{A}$ outputs 1 with a non-negligible probability η. We will use $\mathcal{S}$ to show that $\mathcal{C}$ is $(16t, t, 1/8)$ approximately compressible. In particular, we will use $\mathcal{S}$ and $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to construct Cmp and DeCmp circuits satisfying the three properties of an approximately compressible circuit family.

Note that $\mathcal{A}_1$ picks $C_{n_{\mathsf{eval}}+1}, \ldots, C_{n_{\mathsf{eval}}+n_{\mathsf{test}}}$ and x^* uniformly at random and independent of its other choices. Let $r_{\mathcal{S}}$ and $r_{\mathcal{A}}$ denote the randomness used by the simulator $\mathcal{S}$ and adversary $\mathcal{A}$ (in choosing circuits $C_1, \ldots, C_{n_{\mathsf{eval}}}$, and in RO query collection 1 and test phases), respectively. The compression circuit takes as input (C_1, …, C_{16t}, y_1, …, y_{16t}), has a randomly chosen string for $r_{\mathcal{S}}$ and $r_{\mathcal{A}}$ hardwired, and works as follows:

- Use $\mathcal{S}$ to generate a public key pk. Give pk to $\mathcal{A}_1$.
- Use $\mathcal{S}$ to generate secrets keys sk_1, …, $\mathsf{sk}_{n_{\mathsf{key}}}$ for C'_1, …, $C'_{n_{\mathsf{eval}}}$, C_1, …, C_{16t}, where C'_1, …, $C'_{n_{\mathsf{eval}}}$ are sampled using $r_{\mathcal{A}}$. Give the secret keys to $\mathcal{A}_1$.
- Run the first phase of RO query collection. When $\mathcal{A}_1$ makes an RO query in this phase, forward it to $\mathcal{S}$. Give $\mathcal{S}$'s response back to $\mathcal{A}_1$.
- Provide $y_1, \ldots, y_{16t}$ to $\mathcal{S}$. It generates a ciphertext ct^*.
- Run the second phase of RO query collection. Respond to $\mathcal{A}_2$'s RO queries in the same way as before. Let $z_1, \ldots, z_v$ be the responses in order, where $z_i \in \{0,1\}^m$.
- Output ct^* and $z_1, \ldots, z_v$.

The decompression circuit takes $C_1, \ldots, C_{16t}$ and the compressed string $\mathsf{str\text{-}cmp}$ as inputs, which can be parsed as $\mathsf{str\text{-}cmp} = (\mathsf{ct}^*, \{z_i\})$. It also has the random value chosen before for $r_{\mathcal{S}}$ and $r_{\mathcal{A}}$ hardwired, and works as follows:

- Use $\mathcal{S}$ to generate pk and secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{key}}}$ as before. Give both to $\mathcal{A}_1$.
- Run the first phase of RO query collection. Respond to $\mathcal{A}_1$'s RO queries in the same way as before. Let Γ be the list of RO queries and responses recorded in this phase.
- Run the second phase of RO query collection, where $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{eval}}}$ are used to decrypt ct^*. The RO responses required in this step are available as part of the input ($z_1, \ldots, z_v$). They are also added to Γ.
- Run the test phase with the help of Γ. Let y'_i denote the outcome of decrypting ct^* with $\mathsf{sk}_{n_{\mathsf{eval}}+i}$ for $i \in [n_{\mathsf{test}}]$.
- Output $y'_1, \ldots, y'_{16t}$.

First, note that the size of both compression and decompression circuit is bounded by a polynomial in λ. Next, the output length of the compression circuit is at most $\ell_{\mathsf{ct}} + v \cdot m$, but v is no more than $n_{\mathsf{eval}} \cdot q_{\mathsf{Dec}}$. Thus the output length is bounded by t.

Finally, we need to show that the decompression property works with probability η. When $C_1, \ldots, C_{16t}$ are chosen uniformly at random and $y_1, \ldots, y_{16t}$ is the evaluation of these circuits on a randomly chosen point, then it is easy to see that the decompression circuit emulates the ideal world experiment perfectly. We know that $\mathcal{A}_2$ outputs 1 if and only if for at least 7/8th of the decryptions, $y'_i = y_i$. Hence, if 1 is output with probability η, then the hamming distance of $\mathsf{DeCmp}(\{C_i\}, \mathsf{Cmp}(\{C_i\}, \{y_i\}))$ and $\{y_i\}$ is at most $1/8$ with probability at least η. ■

6 Simulation Secure FE for Bounded Collusions

In this section, we will show an FE scheme that is $(q_1, \mathsf{poly}, \mathsf{q}_2)$ simulation secure in the random oracle model, where q_1, q_2 are a-priori fixed polynomials. Since both the pre-challenge and post-challenge queries are bounded, we will simply refer to the total number of key queries. An FE scheme is q-key poly-ciphertext secure if it is $(q_1, \mathsf{poly}, \mathsf{q}_2)$ simulation secure as in Definition 3.4 for all non-negative integers q_1, q_2 s.t. $q_1 + q_2 = q$.

We first show a scheme that can handle 1 key query in Section 6.1. Then, in Section 6.2 and Appendix 6.3, we show how to transform a 1-key poly-ciphertext scheme to one that is q-key poly-ciphertext simulation secure for an a-priori fixed q, by first building a scheme for log-depth circuits and then for all poly-size circuits. This transformation is very similar to the one showed by Gorbunov et al. [16], except that they dealt with only one ciphertext.

6.1 Simulation Secure FE for One Key Query

We will now describe our 1-key poly-ciphertext scheme. Recall that in the standard model, it is impossible to have simulation security even for IBE if the adversary is allowed to query for an unbounded number of ciphertexts, followed by one adaptive key query [9, 4]. Here, we show how the random oracle can be used to bypass this impossibility result.

Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ be a class of circuits, where each circuit $C \in \mathcal{C}_\lambda$ takes an $n(\lambda)$ bit input and produces an $m(\lambda)$ bit output, and can be represented using $t(\lambda)$ bits. For $x \in \{0,1\}^{n(\lambda)}$, let $U_x^{(\lambda)}$ be a universal circuit that takes any $C \in \mathcal{C}_\lambda$ as input and outputs $C(x)$. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_\lambda$ be a circuit family such that $\mathcal{U}_\lambda = \{U_x^{(\lambda)} \mid x \in \{0,1\}^{n(\lambda)}\}$. Our one-bounded FE scheme $\mathsf{One\text{-}FE} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{KeyGen}, \mathsf{Decrypt})$ uses a decomposable randomized encoding scheme $(\mathsf{RE.Encode}, \mathsf{RE.Decode})$ for $\mathcal{U}$ and a public key encryption scheme $\mathsf{PKE} = (\mathsf{Setup}_{\mathrm{PKE}}, \mathsf{Enc}_{\mathrm{PKE}}, \mathsf{Dec}_{\mathrm{PKE}})$ that can operate on messages of length λ. For simplicity of presentation, we will skip the dependence on λ.

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$: The setup algorithm chooses $2t$ PKE public key/secret key pairs $(\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b}) \leftarrow \mathsf{Setup}_{\mathrm{PKE}}(1^\lambda)$ for $i \in [t], b \in \{0,1\}$. It sets $\mathsf{mpk} = \left\{\mathsf{pk}_{i,b}\right\}_{i\in[t],b\in\{0,1\}}$ and $\mathsf{msk} = \{\mathsf{sk}_{i,b}\}_{i\in[t],b\in\{0,1\}}$.

- $\mathsf{Enc}(\mathsf{mpk}, x) \to \mathsf{ct}$: The encryption algorithm first chooses $2t$ random strings $r_{i,b} \leftarrow \{0,1\}^\lambda$ for all $i \in [t]$, $b \in \{0,1\}$. Next, it computes a randomized encoding for the universal circuit U_x, i.e., $\{w_{i,b}\}_{i\in[t],b\in\{0,1\}} \leftarrow \mathsf{RE.Encode}(1^\lambda, U_x)$. Now, let $\mathsf{ct}_{i,b} = \mathsf{Enc}_{\mathrm{PKE}}(\mathsf{pk}_{i,b}, r_{i,b})$ and $\widetilde{\mathsf{ct}}_{i,b} = w_{i,b} \oplus \mathcal{O}(r_{i,b})$ for all $i \in [t]$, $b \in \{0,1\}$. The algorithm outputs $\mathsf{ct} = \left\{\mathsf{ct}_{i,b}, \widetilde{\mathsf{ct}}_{i,b}\right\}_{i\in[t],b\in\{0,1\}}$.

- $\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{sk}_C$: Let $(\beta_1, \ldots, \beta_t)$ be the bit representation of circuit C. The key generation algorithm outputs $\{\mathsf{sk}_{i,\beta_i}\}_{i\in[t]}$ as the secret key for C.

- $\mathsf{Dec}(\mathsf{mpk}, \mathsf{sk}_C, \mathsf{ct})$: Let $\mathsf{sk}_C = \{\mathsf{sk}_{i,\beta_i}\}_{i\in[t]}$ and $\mathsf{ct} = \left\{\mathsf{ct}_{i,b}, \widetilde{\mathsf{ct}}_{i,b}\right\}_{i\in[t],b\in\{0,1\}}$. The decryption algorithm first decrypts the relevant randomized encoding components, i.e., for each $i \in [t]$, it computes $r_{i,\beta_i} = \mathsf{Dec}_{\mathrm{PKE}}(\mathsf{sk}_{i,\beta_i}, \mathsf{ct}_{i,\beta_i})$ and $w_{i,\beta_i} = \widetilde{\mathsf{ct}}_{i,\beta_i} \oplus \mathcal{O}(r_{i,\beta_i})$. Finally, it outputs $\mathsf{RE.Decode}(\{w_{i,\beta_i}\}_{i\in[t]})$.

The correctness of our scheme follows directly from the correctness of the randomized encoding scheme and the public key encryption scheme.

6.1.1 Simulator

Suppose an adversary outputs M messages in the challenge phase. A simulator $\mathcal{S}$ for our scheme can be defined as follows.

- **Setup.** $\mathcal{S}$ runs $\mathsf{Setup}(1^\lambda)$ honestly to obtain $\mathsf{mpk} = \left\{\mathsf{pk}_{i,b}\right\}_{i\in[t],b\in\{0,1\}}$ and $\mathsf{msk} = \{\mathsf{sk}_{i,b}\}_{i\in[t],b\in\{0,1\}}$. It initializes an empty list Γ that will be used to record random oracle queries and responses. For each $k \in [M]$, it also picks $2t$ random strings $\{r_{k,i,b}\}_{i\in[t],b\in\{0,1\}}$. $\mathcal{S}$ then sends mpk to the adversary.

- **Challenge phase.** There are two cases:
  - *No key query made before.* $\mathcal{S}$ computes $\mathsf{ct}_{k,i,b} \leftarrow \mathsf{Enc}_{\mathrm{PKE}}(\mathsf{pk}_{i,b}, r_{k,i,b})$ and chooses random strings $\widetilde{\mathsf{ct}}_{k,i,b}$, for each $k \in [M]$, $i \in [t]$, $b \in \{0,1\}$. The kth ciphertext ct_k is $\left\{\mathsf{ct}_{k,i,b}, \widetilde{\mathsf{ct}}_{k,i,b}\right\}_{i\in[t],b\in\{0,1\}}$ for $k \in [M]$.

– *A key query was made before.* Suppose $C = (\beta_1, \ldots, \beta_t)$ was the key query. $\mathcal{S}$ receives evaluations $y_1, \ldots, y_M$ of C at all challenge messages. Let $\mathsf{RE.Sim}$ be the simulator for the randomized encoding scheme. $\mathcal{S}$ computes, for each $k \in [M]$, $(w_{k,1}, \ldots, w_{k,t}) \leftarrow \mathsf{RE.Sim}(1^\lambda, C, y_k)$. It also computes $\mathsf{ct}_{k,i,b} \leftarrow \mathsf{Enc}_{\mathrm{PKE}}(\mathsf{pk}_{i,b}, r_{k,i,b})$ and chooses random strings $\widetilde{\mathsf{ct}}_{k,i,b}$, for each $k \in [M]$, $i \in [t]$, $b \in \{0,1\}$. The kth ciphertext ct_k is $\left\{\mathsf{ct}_{k,i,b}, \widetilde{\mathsf{ct}}_{k,i,b}\right\}_{i\in[t], b\in\{0,1\}}$ for $k \in [M]$. Further, $\mathcal{S}$ adds $(r_{k,i,\beta_i}, \widetilde{\mathsf{ct}}_{k,i,\beta_i} \oplus w_{k,i})$ to Γ for $k \in [M]$, $i \in [t]$.

- **Random oracle queries.** At any time before making the only allowed key query, if the adversary makes an RO query q that lies in the set $\{r_{k,i,b}\}_{k\in[M], i\in[t], b\in\{0,1\}}$, the simulator outputs $\perp$ and aborts. Otherwise, it checks if q is present in the list Γ or not. If it is, then the associated response is returned to the adversary. Else, a random bit-string γ of length r is chosen, r is given to the adversary, and (q, γ) is added to Γ.
- **Key query.** Let $C = (\beta_1, \ldots, \beta_t)$ be the key query. There are two cases:
  – *Adaptive query.* In this case, the simulator receives the circuit C as well as evaluations $(y_1, \ldots, y_M)$ at all challenge messages. $\mathcal{S}$ computes, for each $k \in [M]$, $(w_{k,1}, \ldots, w_{k,t}) \leftarrow \mathsf{RE.Sim}(1^\lambda, C, y_k)$. It adds $(r_{k,i,\beta_i}, \widetilde{\mathsf{ct}}_{k,i,\beta_i} \oplus w_{k,i})$ to Γ for all $k \in [M], i \in [t]$, and sends $\mathsf{sk}_C = \{\mathsf{sk}_{i,\beta_i}\}_{i\in[t]}$ to the adversary.
  – *Non-adaptive query.* In this case, the simulator only receives the circuit C. It runs the honest key generation procedure, i.e., it outputs $\{\mathsf{sk}_{i,\beta_i}\}_{i\in[t]}$ as the secret key for C.
- **Random oracle queries.** After making the key query and getting a secret key back, if the adversary makes an RO query q that lies in the set $\{r_{k,i,1-\beta_i}\}_{k\in[M], i\in[t]}$, the simulator outputs $\perp$ and aborts. Otherwise, it behaves in the same way as before.

We prove security of $\mathsf{One\text{-}FE}$ with the help of $\mathcal{S}$ in Appendix A.1.

6.2 Simulation Secure FE with Bounded Key Queries for NC1

In this section, we will show how to transform a scheme that handles one key query to one that handles a bounded number of key queries for the class of log-depth circuits. This transformation is identical to the one in [16]. However, the proof is slightly different because we handle unbounded challenge ciphertext queries.

Formal Description Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ be a class of circuits, where each circuit $C \in \mathcal{C}_\lambda$ takes $n(\lambda)$ bit inputs, outputs a single bit and can be represented using an $n(\lambda)$ variate polynomial of degree $D(\lambda)$ over a (large enough) field $\mathbb{F}$. Let q denote a bound on the number of secret key queries. Our FE scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ uses a 1-key poly-ciphertext simulation secure FE scheme ($\mathsf{Setup}_{\mathrm{one}}$, $\mathsf{Encrypt}_{\mathrm{one}}$, $\mathsf{KeyGen}_{\mathrm{one}}$, $\mathsf{Decrypt}_{\mathrm{one}}$) as a building block. Our scheme is parameterized by four polynomials: N, S, v and t, whose values depend on D and q. As in GVW, we set $t(\lambda) = \Theta(q^2\lambda)$, $N(\lambda) = \Theta(N^2q^2t)$ and $v(\lambda) = \Theta(\lambda)$ and $S(\lambda) = \Theta(vq^2)$. We will skip the dependence on λ when it is clear from the context.

For any circuit $C \in \mathcal{C}_\lambda$ and set $\Delta \subset [S]$, we define a circuit $G_{C,\Delta}$ which takes $n + S$ bit inputs and works as follows:

$$G_{C,\Delta}(x_1, \ldots, x_n, y_1, \ldots, y_S) = C(x_1, \ldots, x_n) + \sum_{h\in\Delta} y_h$$

Let $\mathcal{O} = \mathcal{O}_1 \times \ldots \mathcal{O}_N$ be a hash function, where each $\mathcal{O}_i : \{0,1\}^\ell \to \{0,1\}^m$. Each of these hash functions $\mathcal{O}_i$ will be modeled as a random oracle in our security proof.

- $\mathsf{Setup}^{\mathcal{O}}(1^\lambda) \to (\mathsf{MPK}, \mathsf{MSK})$: The setup algorithm runs the one-key FE scheme's setup N times. Let $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}_{\mathrm{one}}^{\mathcal{O}_i}(1^\lambda)$. The master public key MPK is set to be $\{\mathsf{mpk}_i\}_{i\in[N]}$, and the master secret key MSK is $\{\mathsf{msk}_i\}_{i\in[N]}$.
- $\mathsf{Enc}^{\mathcal{O}}(\mathsf{MPK}, x) \to \mathsf{ct}$: Let $\mathsf{MPK} = \{\mathsf{mpk}_i\}_{i\in[N]}$ and $x = (x_1, \ldots, x_n)$. The encryption algorithm works as follows:

– It chooses n uniformly random polynomials $\mu_1, \ldots, \mu_n$ of degree t over field $\mathbb{F}$ subject to the constraint that the constant term of μ_i is x_i.

– It chooses S uniformly random polynomials $\zeta_1, \ldots, \zeta_S$ of degree Dt over field $\mathbb{F}$ and constant term 0.

– It computes N ciphertexts using the $\mathsf{Encrypt}_{\mathrm{one}}$ algorithm. For $i \in [N]$, it computes $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}_{\mathrm{one}}^{\mathcal{O}_i}(\mathsf{mpk}_i, (\mu_1(i), \ldots, \mu_n(i), \zeta_1(i), \ldots, \zeta_S(i)))$.

The encryption algorithm outputs $(\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$ as the final ciphertext.

• $\mathsf{KeyGen}^{\mathcal{O}}(\mathsf{MSK}, C)$: Let $\mathsf{MSK} = \{\mathsf{msk}_i\}_{i \in [N]}$. The key generation algorithm works as follows:

– It chooses a uniformly random set $\Gamma \subset [N]$ of size $Dt + 1$.

– It chooses a uniformly random set $\Delta \subset [S]$ of size v.

– It uses the $\mathsf{KeyGen}_{\mathrm{one}}$ algorithm to generate $Dt + 1$ secret keys for the function $G_{C,\Delta}$. For $i \in \Gamma$, it computes $\mathsf{sk}_i \leftarrow \mathsf{KeyGen}_{\mathrm{one}}^{\mathcal{O}_i}(\mathsf{msk}_i, G_{C,\Delta})$.

The key generation algorithm outputs $(\Gamma, \Delta, \{\mathsf{sk}_i\}_{i \in \Gamma})$ as the secret key for C.

• $\mathsf{Dec}^{\mathcal{O}}(\mathsf{sk}, \mathsf{ct})$: Let $\mathsf{sk} = (\Gamma, \Delta, \{\mathsf{sk}_i\}_{i \in \Gamma})$ and $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$. The decryption algorithm works as follows:

– For each $i \in \Gamma$, let $\alpha_i = \mathsf{Decrypt}_{\mathrm{one}}^{\mathcal{O}_i}(\mathsf{sk}_i, \mathsf{ct}_i)$.

– It computes a polynomial η of degree Dt over field $\mathbb{F}$ such that for all $i \in \Gamma$, $\eta(i) = \alpha_i$.

The decryption algorithm outputs $\eta(0^{n+S})$ as the final decryption.

Correctness The correctness proof is identical to the one in [16]. Let $\mu_1, \ldots, \mu_n, \zeta_1, \ldots, \zeta_S$ be the polynomials chosen during encryption, and let Γ, Δ be the sets chosen during key generation. From the correctness of the one-key FE scheme, it follows that the decryption algorithm computes $\alpha_i = C(\mu_1(i), \ldots, \mu_n(i)) + \sum_{j \in \Delta} \zeta_j(i)$ for all $i \in \Gamma$. Now, since the polynomial $\eta = C(\mu_1, \ldots, \mu_n) + \sum_{j \in \Gamma} \zeta_j$ has degree Dt and $|\Gamma| = Dt + 1$, the decryption algorithm can compute the polynomial η using the set $\{\alpha_i\}_{i \in [N]}$. Finally, note that $\eta(0^{n+S}) = C(\mu_1(0), \ldots, \mu_n(0)) + \sum_j \zeta_j(0) = C(x_1, \ldots, x_n)$.

6.2.1 Simulation Security

We will first describe our simulator Sim. Let $\mathsf{Sim}_{\mathrm{one}}$ be the simulator for the one-key FE scheme. Our simulator will perform N parallel executions of $\mathsf{Sim}_{\mathrm{one}}$. Let $\left\{\mathsf{Sim}_{\mathrm{one}}^i\right\}_{i \in [N]}$ denote the N parallel executions. Let q_1 denote the number of pre-challenge secret key queries, q_2 the number of post-challenge secret key queries ($q = q_1 + q_2$) and M the number of challenge messages. In the remaining section, the variable $k \in [M]$ will be used for indexing the ciphertexts, $j \in [q]$ will be used to index the secret key query, and $i \in [N]$ will be used to index the components of public key/secret key/ciphertext.

• **Setup**

– The simulator first chooses, for each $j \leq q$, uniformly random sets $\Gamma_j \subset [N]$ of size $Dt + 1$ and $\Delta_j \subset [S]$ of size v. Let $\mathcal{I} = \bigcup_{j \neq j'} (\Gamma_j \bigcap \Gamma_{j'})$.

– For each $i \in \mathcal{I}$, the simulator honestly chooses the master public key/secret key. For each $i \in \mathcal{I}$, it chooses $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}_{\mathrm{one}}(1^\lambda)$.
For all $i \notin \mathcal{I}$, the simulator runs $\mathsf{Sim}_{\mathrm{one}}^i$ to generate the i^{th} public key. Let $\mathsf{mpk}_i \leftarrow \mathsf{Sim}_{\mathrm{one}}^i(1^\lambda)$ for $i \in [N] \setminus \mathcal{I}$. The simulator sets $\mathsf{MPK} = \{\mathsf{mpk}_i\}_{i \in [N]}$ and sends MPK to the adversary.

• **Pre-Challenge Key Generation Queries** Let q_1 denote the number of pre-challenge key queries. For the j^{th} key query C_j, the simulator does the following:

- For each $i \in \Gamma_j \cap \mathcal{I}$, the simulator generates secret keys honestly. It sets $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}^{\mathcal{O}_i}_{\text{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.
- For each $i \in \Gamma_j \setminus \mathcal{I}$, the simulator computes $\mathsf{sk}_{j,i} \leftarrow \mathsf{Sim}^i_{\text{one}}(G_{C_j,\Delta_j})$.

The simulator sends $(\Gamma_j, \Delta_j, \{\mathsf{sk}_{j,i}\}_{i\in\Gamma_j})$ as the j^{th} secret key and sends it to the adversary.

- **Challenge Ciphertexts** The adversary queries for M ciphertexts. Let $x^1, \ldots, x^M$ denote the M messages queried by the adversary. For each $j \in [q_1]$, $k \in [M]$, the simulator receives $C_j(x_k)$. It must output M ciphertexts $\mathsf{ct}^1, \ldots, \mathsf{ct}^M$, and each of these ciphertexts ct^k consists of N components $\mathsf{ct}^k_1, \ldots, \mathsf{ct}^k_N$.

  - *Ciphertext components honestly generated*: For each $k \in [M]$, $i \in \mathcal{I}$, the simulator chooses uniformly random $z^k_{1,i}, \ldots, z^k_{n,i}$, $z'^k_{1,i}, \ldots, z'^k_{S,i}$ and computes honest encryptions. It sets $\mathsf{ct}^k_i \leftarrow \mathsf{Encrypt}_{\text{one}}(\mathsf{mpk}_i, (z^k_{1,i}, \ldots, z^k_{n,i}, z'^k_{1,i}, \ldots, z'^k_{S,i}))$.
  - *Ciphertext components generated by pre-challenge query simulators*: Next, the simulator simulates the ciphertext components for each $k \in [M], i \in \left(\bigcup_{j\in[q_1]} \Gamma_j\right) \setminus \mathcal{I}$. In order to do so, the simulator uses the relevant $\mathsf{Sim}_{\text{one}}$ execution that have been used for generating secret keys in the pre-challenge phase. For each $j \leq q_1$,

    * It chooses uniformly random polynomials $\psi_{j,1}, \ldots, \psi_{j,M}$ of degree Dt subject to the restrictions that $\psi_{j,k}(0^{n+S}) = C_j(x^k)$ and for all $i \in \Gamma_j \bigcap \mathcal{I}$, $\psi_{j,k}(i) = C_j(z^k_{1,i}, \ldots, z^k_{n,i}) + \sum_{h\in\Delta_j} z'^k_{h,i}$.
    * For all $i \in \Gamma_j$, it computes $(\mathsf{ct}^1_i, \ldots, \mathsf{ct}^M_i) \leftarrow \mathsf{Sim}^i_{\text{one}}(\{\psi_{j,k}(i)\}_k)$. [5]

  - *Ciphertext components generated by remaining (post-challenge query) simulators*: Finally, the simulator simulates ciphertext components for each $k \in [M]$, $i \notin (\bigcup_{j\in[q_1]} \Gamma_j \bigcup \mathcal{I})$. For each $i \notin \left(\bigcup_{j\in[q_1]} \Gamma_j\right) \bigcup \mathcal{I}$, it computes $(\mathsf{ct}^1_i, \ldots, \mathsf{ct}^M_i) \leftarrow \mathsf{Sim}^i_{\text{one}}()$. [6]

  The simulator sends $\left(\mathsf{ct}^1 = \left(\mathsf{ct}^1_1, \ldots, \mathsf{ct}^1_N\right), \ldots, \mathsf{ct}^M = \left(\mathsf{ct}^M_1, \ldots, \mathsf{ct}^M_N\right)\right)$ to the adversary.

- **Post Challenge Key Generation Queries** Let q_2 denote the number of post challenge key queries. For the j^{th} query C_j, the simulator also receives circuit evaluations $\left\{C_j(x^k)\right\}_{k\in[M]}$ at all inputs queried during the challenge ciphertext phase.

  It chooses uniformly random polynomials $\psi_{j,1}, \ldots, \psi_{j,M}$ of degree Dt subject to the restrictions that $\psi_{j,k}(0^{n+S}) = C_j(x^k)$ and for all $i \in \Gamma_j \bigcap \mathcal{I}$, $\psi_{j,k}(i) = C_j(z^k_1, \ldots, z^k_n) + \sum_{h\in\Delta_j} z'^k_h$.

  - For each $i \in \Gamma_j \cap \mathcal{I}$, the simulator generates secret keys honestly. It sets $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}^{\mathcal{O}_i}_{\text{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.
  - For each $i \in \Gamma_j \setminus \mathcal{I}$, the simulator uses $\mathsf{Sim}^i_{\text{one}}$. It computes the secret key component $\mathsf{sk}_{j,i} \leftarrow \mathsf{Sim}^i_{\text{one}}(G_{C_j,\Delta_j}, \{\psi_{j,1}(i), \ldots, \psi_{j,M}(i)\})$. [7]

  The secret key $\mathsf{sk}_j = \left(\Gamma_j, \Delta_j, \{\mathsf{sk}_{j,i}\}_{i\in\Gamma_j}\right)$ is sent to the adversary.

- **Random Oracle Queries** For each random oracle query r, the simulator forwards it to each one-query simulator $\mathsf{Sim}^i_{\text{one}}$, and receives responses $y_1, \ldots, y_N$. It forwards these responses to the adversary.

We prove security of our scheme for NC1 in Appendix A.2.

[5] For these indices i, the simulator has been queried for a pre-challenge secret key, and it receives the function evaluations in the ciphertext generation phase.

[6] For these indices i, the simulator has not yet received a secret key query. As a result, it does not receive any additional input for generating the ciphertext.

[7] The simulator receives both the circuit G_{C_j,Δ_j} as well as M evaluations.

6.3 Bootstrapping from NC1 to Poly

In this section, we show how to use the FE scheme $\mathsf{FE}_{\mathsf{NC1}}$ we constructed earlier for NC1 circuits to build an FE scheme that can issue keys for any polynomial-depth circuit. We use the same high-level idea as that of GVW12: in order to generate a secret key for a circuit C, use the $\mathsf{FE}_{\mathsf{NC1}}$ scheme to get a key for a constant-depth randomized encoding of C that derives fresh randomness from a subset of random values encrypted with the input.

Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ be a family of polynomial size circuits. If $\widetilde{C}$ represents a randomized encoding of a circuit $C \in \mathcal{C}_\lambda$, then define a circuit $G_{C,\Delta}$ such that

$$G_{C,\Delta}(x; r_1, \dots, r_s) := \widetilde{C}(x; \oplus_{i\in\Delta} r_i), \tag{1}$$

i.e., a subset of values $r_1, \dots, r_s$ based on Δ is used to compute the randomness for evaluating the encoding $\widetilde{C}$. From the work of Applebaum, Ishai and Kushilevitz [3], we know that any uniform family of polynomial-size circuits admits a constant-degree (perfectly-correct) randomized encoding, assuming the existence of a *minimal* PRG, one that stretches its seed by just one bit, in uniform $\oplus\mathsf{L}/\mathsf{poly}$ (a subclass of $\mathsf{NC1}$). Thus $G_{C,\Delta}$ is computable by a constant-degree polynomial, and we can use our $\mathsf{FE}_{\mathsf{NC1}}$ FE scheme to generate a secret key for it.

Our FE scheme $\mathsf{FE}_{\mathsf{poly}}$ is parameterized by positive integers v and s, just like GVW12. Let $\mathsf{FE}_{\mathsf{NC1}}$ be a $(q_1, \mathsf{poly}, \mathsf{q}_2)$ simulation-secure FE scheme for $\mathsf{NC1}$. The four algorithms for $\mathsf{FE}_{\mathsf{poly}}$ are as follows:

- $\mathsf{Setup}^{\mathcal{O}}(1^\lambda) \to (\mathsf{MPK}, \mathsf{MSK})$: Set MPK and MSK to be the master public and private key, respectively, obtained from $\mathsf{FE}_{\mathsf{NC1}}.\mathsf{Setup}^{\mathcal{O}}(1^\lambda)$.
- $\mathsf{Encrypt}^{\mathcal{O}}(\mathsf{MPK}, x) \to \mathsf{ct}_x$: Pick random numbers $r_1, r_2, \dots, r_s$ and output

  $$\mathsf{FE}_{\mathsf{NC1}}.\mathsf{Encrypt}^{\mathcal{O}}(\mathsf{MPK}, (x, r_1, r_2, \dots, r_s))$$

  as the ciphertext.
- $\mathsf{KeyGen}^{\mathcal{O}}(\mathsf{MSK}, C) \to \mathsf{sk}_C$: Pick a v-sized subset Δ of $[s]$ uniformly at random. Output

  $$\mathsf{FE}_{\mathsf{NC1}}.\mathsf{KeyGen}^{\mathcal{O}}(\mathsf{MSK}, G_{C,\Delta})$$

  as the key, where $G_{C,\Delta}$ is defined in (1).
- $\mathsf{Decrypt}^{\mathcal{O}}(\mathsf{MPK}, \mathsf{sk}_C, \mathsf{ct}_x)$: First run $\mathsf{FE}_{\mathsf{NC1}}.\mathsf{Decrypt}^{\mathcal{O}}(\mathsf{MPK}, \mathsf{sk}_C, \mathsf{ct}_x)$ to get $\widetilde{C}(x; \oplus_{i\in\Delta} r_i)$. Then run the decoder of randomized encoding to get $C(x)$.

The (perfect) correctness of $\mathsf{FE}_{\mathsf{poly}}$ follows from the (perfect) correctness of $\mathsf{FE}_{\mathsf{NC1}}$ and that of randomized encoding.

6.3.1 Simulator

Suppose $\mathsf{Sim}_{\mathsf{NC1}}$ is a simulator for the $\mathsf{FE}_{\mathsf{NC1}}$ scheme. For any adversary $\mathcal{A}$ who makes at most q key queries overall, we can construct a simulator $\mathcal{S}$ that exploits $\mathsf{Sim}_{\mathsf{NC1}}$ as follows. (We will suppress λ below to make the presentation simpler.)

- **Setup.** Run $\mathsf{Sim}_{\mathsf{NC1}}$ to get $\mathsf{FE}_{\mathsf{NC1}}.\mathsf{MPK}$ and $\mathsf{FE}_{\mathsf{NC1}}.\mathsf{MSK}$. Pick v-sized subsets $\Delta_1, \dots, \Delta_q$ of $[s]$ uniformly at random such that for all $j \in [q]$, Δ_i has a unique number a_j that is not present in any other subset. Give $\mathsf{FE}_{\mathsf{NC1}}.\mathsf{MPK}$ to $\mathcal{A}$.
- **Pre-challenge key queries.** When $\mathcal{S}$ receives the jth key query for a circuit C_j, it generates a key sk_{C_j} by running $\mathsf{Sim}_{\mathsf{NC1}}(G_{C_j,\Delta_j})$ (using Δ_j picked earlier as the random subset). Let q_1 be the total number of queries made in this phase.
- **Challenge ciphertexts.** Suppose $\mathcal{A}$ outputs $x_1, \dots, x_M$ as the challenge messages. Then $\mathcal{S}$ gets $C_j(x_k)$ for all $j \in [q_1]$ and $k \in [M]$. It invokes $\mathsf{Sim}_{\mathsf{NC1}}$ on inputs $\{\mathsf{RE}.\mathsf{Sim}(C_j(x_k))\}_{j\in[q_1],k\in[M]}$ to get $\mathsf{ct}_1, \dots, \mathsf{ct}_M$, which is passed onto $\mathcal{A}$.

- **Post-challenge key queries.** When $\mathcal{A}$ makes a query C_j, $\mathcal{S}$ gets C_j along with $C_j(x_k)$ for $k \in [M]$. It invokes $\mathsf{Sim}_{\mathsf{NC1}}$ on inputs G_{C_j,Δ_j} and $\{\mathsf{RE.Sim}(C_j(x_k))\}_{k\in[M]}$ to get a key sk_{C_j}, which is passed onto $\mathcal{A}$.

We prove security of $\mathsf{FE}_{\mathsf{poly}}$ in Appendix A.3.

7 Another Impossibility for Simulation Secure FE

In this section we show that there does not exist a $(0, \mathsf{poly}, \mathsf{poly})$-Sim secure FE scheme for all polynomial-sized circuits in the random oracle model. Thus we get a complete picture of what can and cannot be achieved in the random oracle mode. Once again we rely on circuit families that cannot be approximately compressed for proving the impossibility result. See Section 1.3 for an overview of the result.

Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ be a family of circuits such that each circuit in $\mathcal{C}_\lambda$ takes an $n(\lambda)$-bit input. Let FE be a functional encryption scheme for this family in the random oracle model. We formally define an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

Adversary $\mathcal{A}_1$. Let n_{chal} be a polynomial in λ whose value will be fixed later. $\mathcal{A}_1$ has just two phases, setup and output.

1. **Setup.** $\mathcal{A}_1$ receives the public key pk.
2. **Output.** It picks an x_i^* uniformly at random from $\{0,1\}^{n(\lambda)}$, for $i \in [n_{\mathsf{chal}}]$. It sets the state st to consist of $x_1^*, \ldots, x_{n_{\mathsf{chal}}}^*$, and pk. Then it outputs $((x_1^*, \ldots, x_{n_{\mathsf{chal}}}^*), \mathsf{st})$.

Adversary $\mathcal{A}_2$. Let n_{key} and n_{enc} be polynomials in λ whose values will be fixed later. Let Γ be a list of (query, response) pairs that is empty at the beginning. $\mathcal{A}_2$ gets $(\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{n_{\mathsf{chal}}}^*)$ and st as input, and parses the latter to get $x_1^*, \ldots, x_{n_{\mathsf{chal}}}^*$, and pk. $\mathcal{A}_2$ has seven phases: key query (1 and 2), random oracle query collection (1 and 2), encryption, test, and an output phase.

1. **Key query 1.** For $i \in [n_{\mathsf{key}}]$, $\mathcal{A}_2$ picks a circuit C_i at random from $\mathcal{C}_\lambda$, requests a secret key for C_i, and obtains sk_i in return.
2. **RO query collection 1.** Each of the ciphertexts $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{n_{\mathsf{chal}}}^*$ are decrypted with key sk_i for every $i \in [n_{\mathsf{key}}]$. If an oracle query β is made by the $\mathsf{Decrypt}$ algorithm, $\mathcal{A}_1$ queries the random oracle with the same. The response, say γ, is given to the algorithm, and (β, γ) is added to Γ (if it is not already present).
3. **Encryption.** $\mathcal{A}_2$ picks $x_1, x_2, \ldots, x_{n_{\mathsf{enc}}}$ independently and uniformly at random from $\{0,1\}^{n(\lambda)}$. For $j \in [n_{\mathsf{enc}}]$, it runs $\mathsf{Encrypt}^{\mathcal{O}}(\mathsf{pk}, x_j)$ to obtain a ciphertext ct_j. The RO queries made during the encryption process are forwarded to the random oracle.
4. **Key query 2.** $\mathcal{A}_2$ requests a secret key for a circuit $C_{n_{\mathsf{key}}+1}$, picked at random from $\mathcal{C}_\lambda$, and obtains $\mathsf{sk}_{n_{\mathsf{key}}+1}$ in return.
5. **RO query collection 2.** In this phase, $\mathsf{sk}_{n_{\mathsf{key}}+1}$ is used to decrypt $\mathsf{ct}_1, \ldots, \mathsf{ct}_{n_{\mathsf{enc}}}$. If an oracle query β is made in the process, then $\mathcal{A}_2$ queries the random oracle with the same. The response, say γ, is given to the algorithm, and (β, γ) is added to Γ (if it is not already present).
6. **Test.** In this phase, $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{n_{\mathsf{chal}}}^*$ is decrypted with $\mathsf{sk}_{n_{\mathsf{key}}+1}$ but *without* invoking the random oracle. In order to do so, a new list Δ is initialized first, then the following steps are executed for every $i \in [n_{\mathsf{chal}}]$. The decryption algorithm is run with inputs pk, $\mathsf{sk}_{n_{\mathsf{key}}+1}$, and ct_i^*. When it makes an RO query β, $\mathcal{A}_2$ checks whether there is an entry of the form (β, γ) in Γ or Δ (in that order) or not. If yes, then γ is given to $\mathsf{Decrypt}$ and it continues to run. Otherwise, a random bit-string γ' of length $m(\lambda)$ (the output length of the random oracle) is generated, (β, γ') is added to Δ, and γ' is given to $\mathsf{Decrypt}$. This process of providing responses to the RO queries of $\mathsf{Decrypt}$ continues till it terminates. Let out_i denote the output of $\mathsf{Decrypt}$, which could be $\perp$.

7. **Output.** For every $i \in [n_{\mathsf{chal}}]$, check if out_i is equal to $C_{n_{\mathsf{key}}+1}(x_i^*)$ (where x_i^* is part of the state transferred to $\mathcal{A}_2$). Let num be the number of keys for which this check succeeds. Output 1 if $\mathsf{num}/n_{\mathsf{test}} \geq 7/8$, else output 0.

To complete the description of $\mathcal{A}$, we need to define the polynomials n_{chal}, n_{key} and n_{enc}. Let q_{Setup}, q_{Enc} and q_{KeyGen} be upper-bounds on the number of RO queries made by Setup, $\mathsf{Encrypt}$ and KeyGen, respectively, as a function of λ. Also let ℓ_{Key} be the maximum length of any key generated by KeyGen. Then set

- $n_{\mathsf{chal}} = 16(\ell_{\mathrm{Key}} + n_{\mathsf{enc}} \cdot q_{\mathsf{Dec}} \cdot m)$
- $n_{\mathsf{key}} = 4\lambda \cdot n_{\mathsf{chal}} \cdot q_{\mathsf{Enc}}$,
- $n_{\mathsf{enc}} = 32\lambda(q_{\mathsf{Setup}} + q_{\mathsf{KeyGen}})$.

The real and ideal world analysis are in Appendix B.

References

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

[2] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *CRYPTO*, 2015.

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

[4] Mihir Bellare and Adam O'Neill. Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In *CANS*, 2013.

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

[6] Nir Bitansky and Omer Paneth. On the impossibility of approximate obfuscation and applications to resettable cryptography. In *STOC*, 2013.

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

[8] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil Pairing. CRYPTO '01, 2001.

[9] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: definitions and challenges. TCC'11.

[10] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. TCC'07.

[11] Ran Canetti, Yael Tauman Kalai, and Omer Paneth. On obfuscation with random oracles. In *TCC*, 2015.

[12] Angelo De Caro, Vincenzo Iovino Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In *CRYPTO*, 2013.

[13] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, pages 464–479, 1984.

[14] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Succinct functional encryption and applications: Reusable garbled circuits and beyond. In *STOC*, 2013.

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

[16] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[17] Abhishek Jain. Personal Communication. Aug 7th, 2016.

[18] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. EUROCRYPT'08, 2008.

[19] Mohammad Mahmoody, Ameer Mohammed, Soheil Nematihaji, Rafael Pass, and Abhi Shelat. Lower bounds on assumptions behind indistinguishability obfuscation. In *TCC*, 2016.

[20] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *CRYPTO*, 2002.

[21] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptology ePrint Archive*, 2010:556, 2010.

[22] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, 2010.

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

[24] Adi Shamir. Identity-based cryptosystems and signature schemes. CRYPTO '84.

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

A Simulation Security Proofs

A.1 Simulation Security of One Query FE scheme

For simplicity, we will only prove the adaptive key query case. The non-adaptive key query proof is easier to handle.

First we write down the real experiment in detail for our One-FE scheme.

- **Setup.** $\mathsf{Setup}(1^\lambda)$ is run to obtain $\mathsf{mpk} = \left\{\mathsf{pk}_{i,b}\right\}_{i\in[t],b\in\{0,1\}}$ and $\mathsf{msk} = \{\mathsf{sk}_{i,b}\}_{i\in[t],b\in\{0,1\}}$. Adversary is given mpk.

- **Challenge phase.** Let M denote the number of challenge message queries. For each $k \in [M]$, choose $2t$ random strings $\{r_{k,i,b}\}_{i\in[t],b\in\{0,1\}}$ and compute $\mathsf{ct}_{k,i,b} \leftarrow \mathsf{Enc}_{\mathrm{PKE}}(\mathsf{pk}_{i,b}, r_{k,i,b})$. Also, compute $\{w_{k,i,b}\}_{i\in[t],b\in\{0,1\}} \leftarrow \mathsf{RE.Encode}(1^\lambda, U_{x_k})$. The kth ciphertext ct_k is $\{\mathsf{ct}_{k,i,b}, w_{k,i,b} \oplus \mathcal{O}(\mathsf{ct}_{k,i,b})\}_{i\in[t],b\in\{0,1\}}$ for $k \in [M]$.

- **Key query.** When the adversary makes a key query $C = (\beta_1, \ldots, \beta_t)$, respond with $\mathsf{sk}_C = \{\mathsf{sk}_{i,\beta_i}\}_{i\in[t]}$.

- **Random oracle queries.** All the random oracle queries (including the ones required during encryption) are forwarded to $\mathcal{O}$.

To prove security of our one-bounded scheme, we define two intermediate hybrids, Hyb_1 and Hyb_2. Without loss of generality, assume that any adversary outputs a single bit only. In Hyb_1, the interaction with an adversary is as follows:

- **Setup.** $\mathsf{Setup}(1^\lambda)$ is run to obtain $\mathsf{mpk} = \left\{\mathsf{pk}_{i,b}\right\}_{i\in[t],b\in\{0,1\}}$ and $\mathsf{msk} = \{\mathsf{sk}_{i,b}\}_{i\in[t],b\in\{0,1\}}$. Initialize an empty list Γ that will be used to record random oracle queries and responses. For each $k \in [M]$, pick $2t$ random strings $\{r_{k,i,b}\}_{i\in[t],b\in\{0,1\}}$. Finally, send mpk to the adversary.

- **Challenge phase.** Compute $\mathsf{ct}_{k,i,b} \leftarrow \mathsf{Enc}_{\mathrm{PKE}}(\mathsf{pk}_{i,b}, r_{k,i,b})$ and choose random strings $\widetilde{\mathsf{ct}}_{k,i,b}$, for each $k \in [M]$, $i \in [t]$, $b \in \{0, 1\}$. The kth ciphertext ct_k is set to be $\left\{\mathsf{ct}_{k,i,b}, \widetilde{\mathsf{ct}}_{k,i,b}\right\}_{i\in[t],b\in\{0,1\}}$ for $k \in [M]$. Also, compute $\{w_{k,i,b}\}_{i\in[t],b\in\{0,1\}} \leftarrow \mathsf{RE.Encode}(1^\lambda, U_{x_k})$ for every k, which will be used later.

- **Key query.** When the adversary makes a key query $C = (\beta_1, \ldots, \beta_t)$, respond with $\mathsf{sk}_C = \{\mathsf{sk}_{i,\beta_i}\}_{i\in[t]}$.
- **Random oracle queries.** If the adversary makes an RO query on $r_{k,i,b}$ for some $k \in [M], i \in [t], b \in \{0, 1\}$ *before* the challenge phase, output 0 and abort. If such a query is made after the challenge phase, then return $(\widetilde{\mathsf{ct}}_{k,i,b} \oplus w_{k,i,b})$. For any other query (at any point in the experiment), return a random value. (The list Γ is used to ensure that the responses are consistent.)

It is easy to see that the Real and Hyb_1 are statistically indistinguishable. In particular, the probability that Hyb_1 aborts is negligible because no information about $\{r_{k,i,b}\}$ is available before the challenge phase.

Hyb_2 is exactly the same as Hyb_1 except the manner in which it responds to RO queries:

- **Random oracle queries.** If the adversary makes an RO query on $r_{k,i,b}$ for some k, i, b before the *key phase*, or on $r_{k,i,1-\beta_i}$ for some k, i afterwards, then output 0 and abort. If a query is made on r_{k,i,β_i} after the key phase, return $\widetilde{\mathsf{ct}}_{k,i,\beta_i} \oplus w_{k,i,\beta_i}$. For any other query (at any point in the experiment), return a random value.

Fix any adversary $\mathcal{A}$. In Hyb_1, let p_1 be the probability that $\mathcal{A}$ queries for $r_{k,i,b}$ (for some k, i, b) after the challenge phase but before the key phase or for $r_{k,i,1-\beta_i}$ (for some k, i) after the key phase. If we show that p_1 is negligible, then it is easy to see that Hyb_1 and Hyb_2 are indistinguishable.

Consider a new hybrid $\mathsf{Hyb}_{1,1}$ that differs from Hyb_1 as follows:

- Pick $k^* \xleftarrow{R} [M]$, $i^* \xleftarrow{R} [t]$, $b^* \xleftarrow{R} \{0, 1\}$, and $r \xleftarrow{R} \{0, 1\}^\lambda$ at the beginning.
- If the adversary makes an RO query on r before the challenge phase, output 0 and abort. If such a query is made after the challenge phase, then return $(\widetilde{\mathsf{ct}}_{k^*,i^*,b^*} \oplus w_{k^*,i^*,b^*})$.
- If $\mathcal{A}$ makes a key query $C = (\beta_1, \ldots, \beta_t)$ such that $\beta_{i^*} = b^*$, then output 0 and abort.

Note that RO queries for r_{k^*,i^*,b^*} and r are treated identically. $\mathsf{Hyb}_{1,1}$ outputs 1 if and only if $\mathcal{A}$ queries for r_{k^*,i^*,b^*} after the challenge phase. One can see that 1 is output with probability at least $p_1/2Mt - \mathsf{negl}$.

Define yet another hybrid, $\mathsf{Hyb}_{1,2}$, which is exactly the same as $\mathsf{Hyb}_{1,1}$ except that it outputs 1 if and only if $\mathcal{A}$ queries for r after the challenge phase. It is clear that $\mathsf{Hyb}_{1,2}$ outputs 1 with negligible probability.

We show that $\mathsf{Hyb}_{1,1}$ and $\mathsf{Hyb}_{1,2}$ are computationally indistinguishable, which implies that p_1 is also negligible. We build an adversary $\mathcal{B}$ for the IND-CPA game of PKE by using the adversary $\mathcal{A}$ as follows. $\mathcal{B}$ picks $2t$ random strings $\{r_{k,i,b}\}_{i\in[t],b\in\{0,1\}}$ for each $k \in [M]$ as well as $k^* \xleftarrow{R} [M]$, $i^* \xleftarrow{R} [t]$, $b^* \in \{0, 1\}$, and $r \xleftarrow{R} \{0, 1\}^\lambda$ (just like in $\mathsf{Hyb}_{1,1}$ and $\mathsf{Hyb}_{1,2}$). It gets a public key pk from the IND-CPA challenger, generates $(\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b})$ pairs for all $(i, b) \neq (i^*, b^*)$, sets $\mathsf{pk}_{i^*,b^*} = \mathsf{pk}$, and gives $\{\mathsf{pk}_{i,b}\}$ to $\mathcal{A}$. $\mathcal{B}$ initializes an empty list Γ to record RO queries and responses.

If $\mathcal{A}$ makes an RO query on $\{r_{k,i,b}\}$ or r before submitting its challenge messages $x_1, \ldots, x_M$, then output 0 and abort. Otherwise, $\mathcal{B}$ sends (r_{k^*,i^*,b^*}, r) to the IND-CPA challenger, and get ct^* in return. It encrypts $x_1, \ldots, x_M$ as in $\mathsf{Hyb}_{1,1}$ or $\mathsf{Hyb}_{1,2}$, except that it sets $\mathsf{ct}_{k^*,i^*,b^*} = \mathsf{ct}^*$. In response to an RO query on $r_{k,i,b}$, $(\widetilde{\mathsf{ct}}_{k,i,b} \oplus w_{k,i,b})$ is returned. Further, if r is queried, then the response is $(\widetilde{\mathsf{ct}}_{k^*,i^*,b^*} \oplus w_{k^*,i^*,b^*})$. If $\mathcal{A}$ makes a key query $C = (\beta_1, \ldots, \beta_t)$ such that $\beta_{i^*} = b^*$, then output 0 and abort. Else generate a key for C using $\{\mathsf{sk}_{i,b}\}$ for $(i, b) \neq (i^*, b^*)$. Finally, output 1 if $\mathcal{A}$ queries for r_{k^*,i^*,b^*} after the challenge phase.

One can see that if IND-CPA challenger encrypts r_{k^*,i^*,b^*}, then $\mathcal{B}$'s output is identically distributed to $\mathsf{Hyb}_{1,1}$, otherwise it is identically distributed to $\mathsf{Hyb}_{1,2}$. Thus, due to the security of PKE, $\mathsf{Hyb}_{1,1}$ is computationally indistinguishable from $\mathsf{Hyb}_{1,2}$, and p_1 is negligible.

Finally, the only difference between Hyb_2 and simulator $\mathcal{S}$ is that in the former, a randomized encoding $\{w_{k,i,b}\}$ is computed for x_k in the challenge phase itself, while $\mathcal{S}$ computes it through RE.Sim in the key phase when $C(x_k)$ is made available. But, by the security of randomized encoding, no PPT adversary can distinguish between the two cases.

A.2 Simulation Security of FE Scheme for NC1

A.2.1 Sequence of Hybrids

In order to prove security, we will first define a sequence of hybrid experiments $H_0, \ldots, H_4$ such that H_0 corresponds to the Real experiment and H_4 corresponds to the Ideal World experiment. Let q_1 denote the number of pre-challenge key queries, M the number of challange ciphertexts and q_2 the number of post-challenge key queries. Let $q = q_1 + q_2$ denote the total number of key queries made by the adversary.

Hybrid H_0 This corresponds to the real experiment.

- **Setup Phase**

  1. Choose $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}_{\mathrm{one}}(1^\lambda)$ for $i \in [N]$.
  2. For each $j \in [q]$, choose uniformly random sets $\Gamma_j \subset [N]$ of size $Dt + 1$ and $\Delta_j \subset [S]$ of size v. Let $\mathcal{I} = \bigcup_{j \neq j'} (\Gamma_j \bigcap \Gamma_{j'})$.

  Send $\{\mathsf{mpk}_i\}_{i \in [N]}$ to the adversary.

- **Pre-Challenge Key Query Phase** For each key query C_j

  1. For each $i \in \Gamma_j$, let $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}_{\mathrm{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.

  Send $\mathsf{sk}_j = \left(\Gamma_j, \Delta_j, \{\mathsf{sk}_{j,i}\}_{i \in \Gamma_j}\right)$ to the adversary.

- **Challenge Phase** Let $x^1, \ldots, x^M$ denote the challenge messages, where each message x^k has bit representation $(x_1^k, \ldots x_n^k)$.

  1. For each $k \in [M]$, $h \in [n]$, choose random polynomials $\mu_{k,h}$ of degree t s.t. $\mu_{k,h}(0) = x_h^k$.
  2. For each $k \in [M]$, $h \in [S]$, choose random polynomials $\zeta_{k,h}$ of degree Dt s.t. $\zeta_{k,h}(0) = 0$.
  3. For each $k \in [M]$, $i \in [N]$, compute the ciphertext component $\mathsf{ct}_i^k \leftarrow \mathsf{Encrypt}_{\mathrm{one}}(\mathsf{mpk}_i, (\mu_{k,1}(i), \ldots, \mu_{k,n}(i), \zeta_{k,1}(i), \ldots, \zeta_{k,S}(i)))$.

  Send $\left\{\mathsf{ct}^k = \left(\mathsf{ct}_1^k, \ldots, \mathsf{ct}_N^k\right)\right\}_{k \in [M]}$ to the adversary.

- **Pre-Challenge Key Query Phase** For each key query C_j

  1. For each $i \in \Gamma_j$, let $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}_{\mathrm{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.

  Send $\mathsf{sk}_j = \left(\Gamma_j, \Delta_j, \{\mathsf{sk}_{j,i}\}_{i \in \Gamma_j}\right)$ to the adversary.

- **Random Oracle Queries** Maintain a table $\mathcal{T}$ containing random oracle queries and their responses. For each new query z, choose uniformly random strings κ_i for $i \in [N]$ and send $(\kappa_1, \ldots, \kappa_N)$ to the adversary. Add $(z, (\kappa_1, \ldots, \kappa_N))$ to $\mathcal{T}$.

Hybrid H_1 This experiment is identical to the previous one, except that the challenger aborts if the sets Γ_j have too many indices in common, or if the sets Δ_j are not cover-free. More formally, the setup phase is modified as follows.

Setup Phase

1. Choose $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}_{\mathrm{one}}(1^\lambda)$ for $i \in [N]$.
2. For each $j \in [q]$, choose uniformly random sets $\Gamma_j \subset [N]$ of size $Dt + 1$ and $\Delta_j \subset [S]$ of size v. Let $\mathcal{I} = \bigcup_{j \neq j'} (\Gamma_j \bigcap \Gamma_{j'})$.
3. The adversary wins if either $|\mathcal{I}| > t$ or the sets $\{\Delta_j\}_{j \in [q]}$ are not cover-free.

   If $\{\Delta_j\}_{j \in [q]}$ is cover-free, for each $j \in [q]$, let $\mathsf{rep}(j)$ denote the first index in Δ_j that is not present in $\bigcup_{j' \neq j} \Delta_{j'}$.

Send $\{\mathsf{mpk}_i\}_{i \in [N]}$ to the adversary.

Hybrid H_2 In this experiment, the challenger modifies its response to the challenge messages. Instead of choosing random polynomials $\mu_{k,h}$, it first chooses random field elements $z^k_{h,i}$ for all $i \in \mathcal{I}$, and chooses $\mu_{k,h}$ subject to the restriction that $\mu_{k,h}(i) = z^k_{h,i}$ for all $i \in \mathcal{I}$. Similarly, it chooses random field elements $z'^k_{h,i}$ and chooses $\zeta_{k,h}$ subject to the restriction that $\zeta_{k,h}(i) = z'^k_{h,i}$.

Challenge Phase Let $x^1, \ldots, x^M$ denote the challenge messages, where each message x^k has bit representation $(x^k_1, \ldots x^k_n)$.

1. For each $k \in [M]$, $h \in [n]$, $h' \in [S]$, $i \in \mathcal{I}$, choose random field elements $z^k_{h,i} \leftarrow \mathbb{F}$ and $z'^k_{h',i} \leftarrow \mathbb{F}$.
2. For each $k \in [M]$, $h \in [n]$, choose polynomials $\mu_{k,h}$ of degree t s.t. $\mu_{k,h}(0) = x^k_h$ and for all $i \in \mathcal{I}$, $\mu_{k,h}(i) = z^k_{h,i}$.
3. For each $k \in [M]$, $h \in [S]$, choose random polynomials $\zeta_{k,h}$ of degree Dt s.t. $\zeta_{k,h}(0) = 0$ and for all $i \in \mathcal{I}$, $\zeta_{k,h}(i) = z'^k_{h,i}$.
4. For each $k \in [M]$, $i \in [N]$, compute the ciphertext component $\mathsf{ct}^k_i \leftarrow \mathsf{Encrypt}_{\text{one}}(\mathsf{mpk}_i, (\mu_{k,1}(i), \ldots, \mu_{k,n}(i), \zeta_{k,1}(i), \ldots, \zeta_{k,S}(i)))$.

Send $\{\mathsf{ct}^k = (\mathsf{ct}^k_1, \ldots, \mathsf{ct}^k_N)\}_{k \in [M]}$ to the adversary.

Hybrid H_3 In this experiment, the challenger further modifies the challenge ciphertexts. The modifications in this step are done to ensure that the some of the ciphertext components can be simulated using the one-key simulators.

Challenge Phase Let $x^1, \ldots, x^M$ denote the challenge messages, where each message x^k has bit representation $(x^k_1, \ldots x^k_n)$.

1. For each $k \in [M]$, $h \in [n]$, $h' \in [S]$, $i \in \mathcal{I}$, choose random field elements $z^k_{h,i} \leftarrow \mathbb{F}$ and $z'^k_{h',i} \leftarrow \mathbb{F}$.
2. For each $j \in [q_1]$, $k \in [M]$, choose random polynomials $\psi_{j,k}$ of degree Dt subject to the restrictions that $\psi_{j,k}(0) = C_j(x^k)$ and for each $i \in \mathcal{I}$, $\psi_{j,k}(i) = C_j(z^k_{1,i}, \ldots, z^k_{n,i}) + \sum_{h \in \Delta_j} z'^k_{h,i}$.
3. For each $k \in [M]$, $h \in [n]$, choose polynomials $\mu_{k,h}$ of degree t s.t. $\mu_{k,h}(0) = x^k_h$ and for all $i \in \mathcal{I}$, $\mu_{k,h}(i) = z^k_{h,i}$.
4. For each $k \in [M]$, $h \in [S] \setminus \{\mathsf{rep}(1), \ldots, \mathsf{rep}(q_1)\}$, choose random polynomials $\zeta_{k,h}$ of degree Dt s.t. $\zeta_{k,h}(0) = 0$ and for all $i \in \mathcal{I}$, $\zeta_{k,h}(i) = z'^k_{h,i}$.
5. For each $k \in [M]$, $j \in [q_1]$, set $\zeta_{k,\mathsf{rep}(j)} = \psi_{j,k} - C_j(\mu_{k,1}, \ldots, \mu_{k,n}) - \sum_{h \in \Delta_j \setminus \{\mathsf{rep}(j)\}} \zeta_{k,h}$.
6. For each $k \in [M]$, $i \in [N]$, compute the ciphertext component $\mathsf{ct}^k_i \leftarrow \mathsf{Encrypt}_{\text{one}}(\mathsf{mpk}_i, (\mu_{k,1}(i), \ldots, \mu_{k,n}(i), \zeta_{k,1}(i), \ldots, \zeta_{k,S}(i)))$.

Send $\{\mathsf{ct}^k = (\mathsf{ct}^k_1, \ldots, \mathsf{ct}^k_N)\}_{k \in [M]}$ to the adversary.

Hybrid H_{3,j^*,i^*} **for** $j^* \in [q_1]$**,** $i^* \in \Gamma_{j^*} \setminus \mathcal{I}$ Next, we define a sequence of hybrids where the pre-challenge key queries (and the corresponding public keys and ciphertext components) are simulated by the one-key simulators.

- **Setup Phase**
  1. For each $j \in [q]$, choose uniformly random sets $\Gamma_j \subset [N]$ of size $Dt + 1$ and $\Delta_j \subset [S]$ of size v. Let $\mathcal{I} = \bigcup_{j \neq j'} (\Gamma_j \bigcap \Gamma_{j'})$.
  2. The adversary wins if either $|\mathcal{I}| > t$ or the sets $\{\Delta_j\}_{j \in [q]}$ are not cover-free.

     If $\{\Delta_j\}_{j \in [q]}$ is cover-free, for each $j \in [q]$, let $\mathsf{rep}(j)$ denote the first index in Δ_j that is not present in $\bigcup_{j' \neq j} \Delta_{j'}$.

3. For each $j < j^*$, $i \in \Gamma_j \setminus \mathcal{I}$, choose $\mathsf{mpk}_i \leftarrow \mathsf{Sim}^i_{\text{one}}()$.
   For each $i \in \Gamma_{j^*} \setminus \mathcal{I}$, $i \leq i^*$, choose $\mathsf{mpk}_i \leftarrow \mathsf{Sim}^i_{\text{one}}()$.
4. Choose $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}_{\text{one}}(1^\lambda)$ for the remaining indices.

Send $\{\mathsf{mpk}_i\}_{i\in[N]}$ to the adversary.

- **Pre-Challenge Key Query Phase** For each key query C_j

1. For each $i \in \mathcal{I}$, set $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}_{\text{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.
2. If $j < j^*$ and $i \in \Gamma_j \setminus \mathcal{I}$, let $\mathsf{sk}_{j,i} \leftarrow \mathsf{Sim}^i_{\text{one}}(G_{C_j,\Delta_j})$.
   If $j = j^*$ and $i \in \Gamma_{j^*} \setminus \mathcal{I}$, $i \leq i^*$, let $\mathsf{sk}_{j,i} \leftarrow \mathsf{Sim}^i_{\text{one}}(G_{C_j,\Delta_j})$. For $i > i^*$, let $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}_{\text{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.
   If $j > j^*$ and $i \in \Gamma_j$, let $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}_{\text{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.

Send $\mathsf{sk}_j = \left(\Gamma_j, \Delta_j, \{\mathsf{sk}_{j,i}\}_{i\in\Gamma_j}\right)$ to the adversary.

- **Challenge Phase** Let $x^1, \ldots, x^M$ denote the challenge messages, where each message x^k has bit representation $(x^k_1, \ldots x^k_n)$.

1. For each $k \in [M]$, $h \in [n]$, $h' \in [S]$, $i \in \mathcal{I}$, choose random field elements $z^k_{h,i} \leftarrow \mathbb{F}$ and $z'^k_{h',i} \leftarrow \mathbb{F}$ and compute the ciphertext component $\mathsf{ct}^k_i \leftarrow \mathsf{Encrypt}_{\text{one}}(\mathsf{mpk}_i, (z^k_{1,i}, \ldots, z^k_{n,i}, z'^k_{1,i}, \ldots, z'^k_{S,i}))$.
2. For each $j \in [q_1]$, $k \in [M]$, choose random polynomials $\psi_{j,k}$ of degree Dt subject to the restrictions that $\psi_{j,k}(0) = C_j(x^k)$ and for each $i \in \mathcal{I}$, $\psi_{j,k}(i) = C_j(z^k_{1,i}, \ldots, z^k_{n,i}) + \sum_{h\in\Delta_j} z'^k_{h,i}$.
3. For each $i \in \bigcup_{j<j^*} \Gamma_j \setminus \mathcal{I}$, set $(\mathsf{ct}^1_i, \ldots, \mathsf{ct}^M_i) \leftarrow \mathsf{Sim}^i_{\text{one}}(\psi_{j,k}(i))$.
   For each $i \in \Gamma_{j^*} \setminus \mathcal{I}$, $i \leq i^*$, set $(\mathsf{ct}^1_i, \ldots, \mathsf{ct}^M_i) \leftarrow \mathsf{Sim}^i_{\text{one}}(\psi_{j,k}(i))$.
4. For each $k \in [M]$, $h \in [n]$, choose polynomials $\mu_{k,h}$ of degree t s.t. $\mu_{k,h}(0) = x^k_h$ and for all $i \in \mathcal{I}$, $\mu_{k,h}(i) = z^k_{h,i}$.
5. For each $k \in [M]$, $h \in [S] \setminus \{\mathsf{rep}(1), \ldots, \mathsf{rep}(q_1)\}$, choose random polynomials $\zeta_{k,h}$ of degree Dt s.t. $\zeta_{k,h}(0) = 0$ and for all $i \in \mathcal{I}$, $\zeta_{k,h}(i) = z'^k_{h,i}$.
6. For each $k \in [M]$, $j \in [q_1]$, set $\zeta_{k,\mathsf{rep}(j)} = \psi_{j,k} - C_j(\mu_{k,1}, \ldots, \mu_{k,n}) - \sum_{h\in\Delta_j\setminus\{\mathsf{rep}(j)\}} \zeta_{k,h}$.
7. The remaining ciphertext components are generated honestly.
   For the remaining indices k, i, compute the ciphertext component $\mathsf{ct}^k_i \leftarrow \mathsf{Encrypt}_{\text{one}}(\mathsf{mpk}_i, (\mu_{k,1}(i), \ldots, \mu_{k,n}(i), \zeta_{k,1}(i), \ldots, \zeta_{k,S}(i)))$.

Send $\left\{\mathsf{ct}^k = (\mathsf{ct}^k_1, \ldots, \mathsf{ct}^k_N)\right\}_{k\in[M]}$ to the adversary.

- **Post-Challenge Key Query Phase and Random Oracle Queries** Same as in previous game.

Hybrids H_{3,j^*,i^*} for $j \in \{q_1+1, \ldots, q_2\}$, $i \in \Gamma_{j^*}$ Next, we define a sequence of hybrids where the post challenge queries are handled by the simulator.

- **Setup Phase**

1. For each $j \in [q]$, choose uniformly random sets $\Gamma_j \subset [N]$ of size $Dt + 1$ and $\Delta_j \subset [S]$ of size v. Let $\mathcal{I} = \bigcup_{j\neq j'} (\Gamma_j \bigcap \Gamma_{j'})$.
2. The adversary wins if either $|\mathcal{I}| > t$ or the sets $\{\Delta_j\}_{j\in[q]}$ are not cover-free.
   If $\{\Delta_j\}_{j\in[q]}$ is cover-free, for each $j \in [q]$, let $\mathsf{rep}(j)$ denote the first index in Δ_j that is not present in $\bigcup_{j'\neq j} \Delta_{j'}$.
3. For each $j < j^*$, $i \in \Gamma_j \setminus \mathcal{I}$, choose $\mathsf{mpk}_i \leftarrow \mathsf{Sim}^i_{\text{one}}()$.
   For each $i \in \Gamma_{j^*} \setminus \mathcal{I}$, $i \leq i^*$, choose $\mathsf{mpk}_i \leftarrow \mathsf{Sim}^i_{\text{one}}()$.

4. For the remaining indices, choose the public/secret keys honestly. Choose $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}_{\mathrm{one}}(1^\lambda)$ for the remaining i.

Send $\{\mathsf{mpk}_i\}_{i\in[N]}$ to the adversary.

- **Pre-Challenge Key Query Phase** Same as in previous game.
- **Challenge Phase** Let $x^1, \dots, x^M$ denote the challenge messages, where each message x^k has bit representation $(x_1^k, \dots x_n^k)$.

1. For each $k \in [M]$, $h \in [n]$, $h' \in [S]$, $i \in \mathcal{I}$, choose random field elements $z_{h,i}^k \leftarrow \mathbb{F}$ and $z'^k_{h',i} \leftarrow \mathbb{F}$ and compute the ciphertext component $\mathsf{ct}_i^k \leftarrow \mathsf{Encrypt}_{\mathrm{one}}(\mathsf{mpk}_i, (z_{1,i}^k, \dots, z_{n,i}^k, z'^k_{1,i}, \dots, z'^k_{S,i}))$.
2. For each $j \in [q_1]$, $k \in [M]$, choose random polynomials $\psi_{j,k}$ of degree Dt subject to the restrictions that $\psi_{j,k}(0) = C_j(x^k)$ and for each $i \in \mathcal{I}$, $\psi_{j,k}(i) = C_j(z_{1,i}^k, \dots, z_{n,i}^k) + \sum_{h\in\Delta_j} z'^k_{h,i}$.
3. For each $i \in \bigcup_{j\le q_1} \Gamma_j \setminus \mathcal{I}$, set $(\mathsf{ct}_i^1, \dots, \mathsf{ct}_i^M) \leftarrow \mathsf{Sim}^i_{\mathrm{one}}(\psi_{j,k}(i))$.
4. For each $i \in \bigcup_{q_1<j<j^*} \Gamma_j \setminus \mathcal{I}$, set $(\mathsf{ct}_i^1, \dots, \mathsf{ct}_i^M) \leftarrow \mathsf{Sim}^i_{\mathrm{one}}()$.
   For each $i \in \Gamma_{j^*} \setminus \mathcal{I}$, $i \le i^*$, set $(\mathsf{ct}_i^1, \dots, \mathsf{ct}_i^M) \leftarrow \mathsf{Sim}^i_{\mathrm{one}}()$.
5. For each $k \in [M]$, $h \in [n]$, choose polynomials $\mu_{k,h}$ of degree t s.t. $\mu_{k,h}(0) = x_h^k$ and for all $i \in \mathcal{I}$, $\mu_{k,h}(i) = z_{h,i}^k$.
6. For each $k \in [M]$, $h \in [S] \setminus \{\mathsf{rep}(1), \dots, \mathsf{rep}(q_1)\}$, choose random polynomials $\zeta_{k,h}$ of degree Dt s.t. $\zeta_{k,h}(0) = 0$ and for all $i \in \mathcal{I}$, $\zeta_{k,h}(i) = z'^k_{h,i}$.
7. For each $k \in [M]$, $j \in [q_1]$, set $\zeta_{k,\mathsf{rep}(j)} = \psi_{j,k} - C_j(\mu_{k,1}, \dots, \mu_{k,n}) - \sum_{h\in\Delta_j\setminus\{\mathsf{rep}(j)\}} \zeta_{k,h}$.
8. The remaining ciphertext components are generated honestly.
   For the remaining indices k, i, compute the ciphertext component $\mathsf{ct}_i^k \leftarrow \mathsf{Encrypt}_{\mathrm{one}}(\mathsf{mpk}_i, (\mu_{k,1}(i), \dots, \mu_{k,n}(i), \zeta_{k,1}(i), \dots, \zeta_{k,S}(i)))$.

Send $\left\{\mathsf{ct}^k = (\mathsf{ct}_1^k, \dots, \mathsf{ct}_N^k)\right\}_{k\in[M]}$ to the adversary.

- **Post-Challenge Key Query Phase** For each key query C_j

1. For each $i \in \mathcal{I}$, set $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}_{\mathrm{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.
2. If ($q_1 < j < j^*$ and $i \in \Gamma_j \setminus \mathcal{I}$) or ($j = j^*$ and $i \in \Gamma_{j^*} \setminus \mathcal{I}$, $i \le i^*$), let $\mathsf{sk}_{j,i} \leftarrow \mathsf{Sim}^i_{\mathrm{one}}(G_{C_j,\Delta_j}, \left\{G_{C_j,\Delta_j}(\mu_{k,1}(i), \dots, \mu_{k,n}(i), \zeta_{k,1}(i), \dots, \zeta_{k,S}(i))\right\}_k)$.
   For all remaining indices, let $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}_{\mathrm{one}}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.

Send $\mathsf{sk}_j = \left(\Gamma_j, \Delta_j, \{\mathsf{sk}_{j,i}\}_{i\in\Gamma_j}\right)$ to the adversary.

Hybrid H_4 In this hybrid, the public key and ciphertext components for $i \notin \mathcal{I}$ are also simulated. This includes indices corresponding to which no secret key components are given.

- **Setup Phase**

1. For each $j \in [q]$, choose uniformly random sets $\Gamma_j \subset [N]$ of size $Dt + 1$ and $\Delta_j \subset [S]$ of size v. Let $\mathcal{I} = \bigcup_{j\ne j'} (\Gamma_j \bigcap \Gamma_{j'})$.
2. The adversary wins if either $|\mathcal{I}| > t$ or the sets $\{\Delta_j\}_{j\in[q]}$ are not cover-free.
   If $\{\Delta_j\}_{j\in[q]}$ is cover-free, for each $j \in [q]$, let $\mathsf{rep}(j)$ denote the first index in Δ_j that is not present in $\bigcup_{j'\ne j} \Delta_{j'}$.
3. For each $i \notin \mathcal{I}$, choose $\mathsf{mpk}_i \leftarrow \mathsf{Sim}^i_{\mathrm{one}}()$.
4. For each $i \in \mathcal{I}$, let $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}_{\mathrm{one}}(1^\lambda)$.

Send $\{\mathsf{mpk}_i\}_{i\in[N]}$ to the adversary.

- **Pre-Challenge Key Query Phase** Same as in previous hybrid.
- **Challenge Phase** Let $x^1, \ldots, x^M$ denote the challenge messages, where each message x^k has bit representation $(x_1^k, \ldots x_n^k)$.
  1. For each $k \in [M]$, $h \in [n]$, $h' \in [S]$, $i \in \mathcal{I}$, choose random field elements $z_{h,i}^k \leftarrow \mathbb{F}$ and $z_{h',i}'^k \leftarrow \mathbb{F}$ and compute the ciphertext component $\mathsf{ct}_i^k \leftarrow \mathsf{Encrypt}_{\text{one}}(\mathsf{mpk}_i, (z_{1,i}^k, \ldots, z_{n,i}^k, z_{1,i}'^k, \ldots, z_{S,i}'^k))$.
  2. For each $j \in [q_1]$, $k \in [M]$, choose random polynomials $\psi_{j,k}$ of degree Dt subject to the restrictions that $\psi_{j,k}(0) = C_j(x^k)$ and for each $i \in \mathcal{I}$, $\psi_{j,k}(i) = C_j(z_{1,i}^k, \ldots, z_{n,i}^k) + \sum_{h \in \Delta_j} z_{h,i}'^k$.
  3. For each $i \in \bigcup_{j \le q_1} \Gamma_j \setminus \mathcal{I}$, set $(\mathsf{ct}_i^1, \ldots, \mathsf{ct}_i^M) \leftarrow \mathsf{Sim}_{\text{one}}^i(\psi_{j,k}(i))$.
  4. For the remaining indices i, set $(\mathsf{ct}_i^1, \ldots, \mathsf{ct}_i^M) \leftarrow \mathsf{Sim}_{\text{one}}^i()$.
  5. For each $k \in [M]$, $h \in [n]$, choose polynomials $\mu_{k,h}$ of degree t s.t. $\mu_{k,h}(0) = x_h^k$ and for all $i \in \mathcal{I}$, $\mu_{k,h}(i) = z_{h,i}^k$.
  6. For each $k \in [M]$, $h \in [S] \setminus \{\mathsf{rep}(1), \ldots, \mathsf{rep}(q_1)\}$, choose random polynomials $\zeta_{k,h}$ of degree Dt s.t. $\zeta_{k,h}(0) = 0$ and for all $i \in \mathcal{I}$, $\zeta_{k,h}(i) = z_{h,i}'^k$.
  7. For each $k \in [M]$, $j \in [q_1]$, set $\zeta_{k,\mathsf{rep}(j)} = \psi_{j,k} - C_j(\mu_{k,1}, \ldots, \mu_{k,n}) - \sum_{h \in \Delta_j \setminus \{\mathsf{rep}(j)\}} \zeta_{k,h}$.

  Send $\left\{\mathsf{ct}^k = (\mathsf{ct}_1^k, \ldots, \mathsf{ct}_N^k)\right\}_{k \in [M]}$ to the adversary.
- **Pre-Challenge Key Query Phase** Same as in previous hybrid.

Hybrid H_5 In the previous hybrid, note that the polynomials $\mu_{k,h}$ and $\zeta_{k,h}$ are not required during the encryption phase. In fact, even during the post-challenge key generation phase, the one-key simulator only requires $\alpha_{k,j,i} = C_j(\mu_{k,1}(i), \ldots, \mu_{k,n}(i)) + \sum_{h \in \Delta_j} \zeta_{k,h}(i)$. This value can be simulated using $C_j(x^k)$ (that is, $\alpha_{k,j,i}$ can be simulated without knowing x^k).

- **Setup Phase** Same as in previous hybrid.
- **Pre-Challenge Key Query Phase** Same as in previous hybrid.
- **Challenge Phase** Same as in previous hybrid, except that the challenger does not choose the polynomials $\mu_{k,h}$ and $\zeta_{k,h}$.
- **Post-Challenge Key Query Phase** For the j^{th} query C_j, the simulator also receives circuit evaluations $\left\{C_j(x^k)\right\}_{k \in [M]}$ at all inputs queried during the challenge ciphertext phase.

  Choose uniformly random polynomials $\psi_{j,1}, \ldots, \psi_{j,M}$ of degree Dt subject to the restrictions that $\psi_{j,k}(0^{n+S}) = C_j(x^k)$ and for all $i \in \Gamma_j \bigcap \mathcal{I}$, $\psi_{j,k}(i) = C_j(z_1^k, \ldots, z_n^k) + \sum_{h \in \Delta_j} z_h'^k$.

  – For each $i \in \Gamma_j \bigcap \mathcal{I}$, compute $\mathsf{sk}_{j,i} \leftarrow \mathsf{KeyGen}_{\text{one}}^{\mathcal{O}_i}(\mathsf{msk}_i, G_{C_j,\Delta_j})$.

  – For each $i \in \Gamma_j \setminus \mathcal{I}$, compute $\mathsf{sk}_{j,i} \leftarrow \mathsf{Sim}_{\text{one}}^i(G_{C_j,\Delta_j}, \{\psi_{j,1}(i), \ldots, \psi_{j,M}(i)\})$.

  The secret key $\mathsf{sk}_j = \left(\Gamma_j, \Delta_j, \{\mathsf{sk}_{j,i}\}_{i \in \Gamma_j}\right)$ is sent to the adversary.

A.2.2 Analysis

We will now show that any PPT adversary's advantage in each of the above hybrids is negligible. For any adversary $\mathcal{A}$, let $\mathsf{Adv}_x^{\mathcal{A}}$ denote the advantage of $\mathcal{A}$ in hybrid H_x.

Claim A.1. *For any adversary $\mathcal{A}$,* $\mathsf{Adv}_1^{\mathcal{A}} = \mathsf{Adv}_0^{\mathcal{A}} - \mathsf{negl}(\lambda)$.

Proof. This follows from our choice of parameters N, t, S, v. As discussed in [16] (Section 5.2), setting $t = \Theta(q^2\lambda)$, $N = \Theta(D^2q^2t)$ ensures that $|\bigcup_{j \neq j'} \Gamma_j \bigcap \Gamma_{j'}| \le t$. For cover-freeness, we set $v = \Theta(\lambda)$ and $S = \Theta(vq^2)$. ∎

Claim A.2. *For any adversary $\mathcal{A}$,* $\mathsf{Adv}_1^{\mathcal{A}} = \mathsf{Adv}_2^{\mathcal{A}}$.

Proof. The adversary's view in hybrids H_1 and H_2 is identical. In H_2, each polynomial $\mu_{k,h}$ is uniformly random, subject to one constraint : $\mu_{k,h}(0) = x_h^k$. In H_2, the challenger chooses t random points $z_{1,i}^k, \ldots, z_{h,i}^k$ and then chooses $\mu_{k,h}$ subject to the restrictions that $\mu_{k,h}(i) = z_{h,i}^k$. Since $\mu_{k,h}$ has degree t, these two views are identical. Similarly, we have the choice of $\zeta_{k,h}$. This shows that the views in the two experiments are identical. ∎

Claim A.3. *For any adversary $\mathcal{A}$,* $\mathsf{Adv}_2^{\mathcal{A}} = \mathsf{Adv}_3^{\mathcal{A}}$.

Proof. The proof of this claim relies on the cover-free property of the sets $\{\Delta_1, \ldots, \Delta_{q_2}\}$. Since these sets are cover-free, we can choose a representative $\mathsf{rep}(j) \in \Gamma_j \setminus \bigcup_{j' \neq j} \Gamma_{j'}$ for each $j \leq q$. Notice the only difference in the two hybrids is the choice of $\zeta_{k,\mathsf{rep}(j)}$ for each $j \in [q_1]$. In H_2, $\zeta_{k,\mathsf{rep}(j)}$ is a uniformly random polynomial subject to restrictions at $t+1$ points $\{0\} \bigcup \mathcal{I}$. In H_3, $\zeta_{k,\mathsf{rep}(k)}$ is defined as $\psi_{j,k} - P$, where P is some fixed polynomial [8]. Here $\psi_{j,k}$ is a uniformly random polynomial of degree Dt subject to restrictions at $\{0\} \bigcup \mathcal{I}$. Moreover, in both hybrids, $\zeta_{k,\mathsf{rep}(j)}$ takes the same value at all points in $\{0\} \bigcup \mathcal{I}$. This shows that the two distributions are identical. ∎

Claim A.4. *Assuming* $\mathsf{FE}_{\mathrm{one}}$ *is one-key many-ciphertext simulation secure, for any $j^* \in [q_1]$, $i^* \in \Gamma_{j^*}$, PPT adversary $\mathcal{A}$,* $|\mathsf{Adv}_{3,j^*,i^*}^{\mathcal{A}} - \mathsf{Adv}_{3,j^*,i^*+1}^{\mathcal{A}}| \leq \mathsf{negl}(\lambda)$.

Proof. The proof of this claim follows directly from the one-key many-ciphertext simulation security of $\mathsf{FE}_{\mathrm{one}}$. ∎

Claim A.5. *Assuming* $\mathsf{FE}_{\mathrm{one}}$ *is one-key many-ciphertext simulation secure, for any $j^* \in \{q_1+1, \ldots, q\}$, $i \in \Gamma_{j^*}$, PPT adversary $\mathcal{A}$,* $|\mathsf{Adv}_{3,j^*,i^*}^{\mathcal{A}} - \mathsf{Adv}_{3,j^*,i^*+1}^{\mathcal{A}}| \leq \mathsf{negl}(\lambda)$.

Proof. The proof of this claim follows directly from the one-key many-ciphertext simulation security of $\mathsf{FE}_{\mathrm{one}}$. ∎

Claim A.6. *Let i^* be the last index in Γ_{q_2}. Assuming* $\mathsf{FE}_{\mathrm{one}}$ *is zero-key many-ciphertext secure, for any PPT adversary $\mathcal{A}$,* $|\mathsf{Adv}_{3,q,i^*}^{\mathcal{A}} - \mathsf{Adv}_4^{\mathcal{A}}| \leq \mathsf{negl}(\lambda)$.

Proof. The proof of this claim follows directly from the zero-key many-ciphertext simulation security of $\mathsf{FE}_{\mathrm{one}}$. ∎

Claim A.7. *For any adversary $\mathcal{A}$,* $\mathsf{Adv}_4^{\mathcal{A}} = \mathsf{Adv}_5^{\mathcal{A}}$.

Proof. This proof is identical to the proof of Claim A.3. ∎

A.3 Simulation Security of FE Scheme for All Circuits

A.3.1 Sequence of Hybrids

Hybrid 1. We switch from the real experiment to using the simulator of $\mathsf{FE}_{\mathsf{NC1}}$, but at the same time, unlike the ideal world experiment, challenge messages are directly used. Formally,

- **Setup.** Same as the set-up phase of $\mathcal{S}$.
- **Pre-challenge key queries.** Same as the corresponding phase of $\mathcal{S}$.
- **Challenge ciphertexts.** Suppose $\mathcal{A}$ outputs $x_1, \ldots, x_M$ as the challenge messages. Then pick $r_{k,1}, r_{k,2}, \ldots, r_{k,s}$ at random for every $k \in [M]$ and invoke $\mathsf{Sim}_{\mathsf{NC1}}$ on inputs $\{G_{C_j,\Delta_j}(x_k; r_{k,1}, \ldots, r_{k,s})\}_{j \in [q_1], k \in [M]}$ to get $\mathsf{ct}_1, \ldots, \mathsf{ct}_M$.
- **Post-challenge key queries.** When $\mathcal{A}$ makes a query C_j, invoke $\mathsf{Sim}_{\mathsf{NC1}}$ on inputs G_{C_j,Δ_j} and $\{G_{C_j,\Delta_j}(x_k; r_{k,1}, \ldots, r_{k,s})\}_{k \in [M]}$ to get a key sk_{C_j}.

[8] $P = C_j(\mu_{k,1}, \ldots, \mu_{k,n}) + \sum_{h \in \Delta_j \setminus \{\mathsf{rep}(j)\}} \zeta_{k,h}$

Hybrid 2. Hyb_2 is exactly the same as Hyb_1 except that in order to compute G_{C_j,Δ_j} on an input x_k for any j, k, a uniformly chosen value $\hat{r}_{j,k}$ is used instead of $\oplus_{i\in\Delta_j} r_{k,i}$. (In other words, a randomized encoding of C_j on x_k is computed using fresh randomness $r_{k,i}$.)

A.3.2 Analysis

Claim A.8. *If* $\mathsf{Sim}_{\mathsf{NC1}}$ *is an admissible simulator for* $(q_1, \mathsf{poly}, \mathsf{q}_2)$ *simulation security of* $\mathsf{FE}_{\mathsf{NC1}}$*, then* $\mathsf{Real}_{\mathcal{A}}^{\mathsf{FE}_{\mathsf{poly}}}$ *is computationally indistinguishable from* Hyb_1.

Proof. This claim can be easily verified by observing that Hyb_1 is identical to the ideal world for $\mathsf{FE}_{\mathsf{NC1}}$. ∎

Claim A.9. *The output of* Hyb_1 *is statistically close to that of* Hyb_2.

Proof. This follows from the cover-freeness of the sets $\Delta_1, \ldots, \Delta_q$. With high probability over the choice of these sets, $\Delta_j \setminus \bigcup_{i\in[q], i\neq j} \Delta_i$ is not empty for any j. Thus for every $k \in [M]$, $\oplus_{i\in\Delta_1} r_{k,i}$, …, $\oplus_{i\in\Delta_q} r_{k,i}$ are uniformly distributed. ∎

Claim A.10. Hyb_2 *is computationally indistinguishable from* $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}^{\mathsf{FE}_{\mathsf{poly}}}$ *due to the security of randomized encodings.*

Proof. The only difference between Hyb_2 and the ideal world is that $\widetilde{C}_j(x_k; r_{j,k})$ is replaced by $\mathsf{RE.Sim}(C_j(x_k))$ for every j, k. However, since $r_{j,k}$ is a uniformly random value, the output of randomized encoding can be simulated by $\mathsf{RE.Sim}$ given just the function evaluation. ∎

B Real And Ideal Analysis of Second Impossibility

B.1 Real World Analysis

First, we will show that the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ described above outputs 1 with probability at least $3/4$ in the real world experiment. We will refer to the special key $\mathsf{sk}_{n_{\mathsf{key}}+1}$ as sk^* below. To begin with, we classify the random oracle queries made during a run of $\mathcal{A}$ into different sets as follows:

- $\mathsf{S\text{-}RO}_{\mathsf{Setup}}$: random oracle queries made during setup phase.
- $\mathsf{S\text{-}RO}_{x_i^*}$ for $i \in [n_{\mathsf{chal}}]$: random oracle queries made while encrypting x_i^* using pk.
- $\mathsf{S\text{-}RO}_{\mathsf{chal}} = \bigcup_{i\in[n_{\mathsf{chal}}]} \mathsf{S\text{-}RO}_{x_i^*}$: all random oracle queries made during the encryption of $x_1^*, \ldots, x_{n_{\mathsf{chal}}}^*$.
- $\mathsf{S\text{-}RO}_{\mathsf{s\text{-}key}}$: random oracle queries made by KeyGen while generating secret key for $C_{n_{\mathsf{key}}+1}$ in the second key query phase.
- $\mathsf{S\text{-}RO}_{\mathrm{Dec}\text{-}i}$ for $i \in [n_{\mathsf{test}}]$: random oracle queries made during the decryption of ct_i^* using sk^*.
- $\mathsf{S\text{-}RO}_{\Gamma\text{-}b}$: random oracle queries recorded during 'RO Collection Phase b' for $b \in \{1, 2\}$. Let $\mathsf{S\text{-}RO}_{\Gamma} = \mathsf{S\text{-}RO}_{\Gamma\text{-}1} \bigcup \mathsf{S\text{-}RO}_{\Gamma\text{-}2}$.

Lemma B.1. *For any functional encryption scheme* FE *for the circuit family* $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$*, the adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *described in Section 5.1 outputs* 1 *in* $\mathsf{Real}_{\mathcal{A}}^{\mathsf{FE}}(1^\lambda)$ *with probability at least* $3/4 - \mathsf{negl}(\lambda)$.

Proof. Let Bad denote the event that the adversary outputs 0 at the end of the real world experiment. This event happens if at least $1/8$th fraction of the n_{chal} decryptions fail in the test phase. If $\mathsf{I\text{-}Dec}_i$ is an indicator variable that takes the value 1 in case the ith decryption *fails*, then Bad happens iff $\sum_{i\in[n_{\mathsf{chal}}]} \mathsf{I\text{-}Dec}_i > 1/8 \cdot n_{\mathsf{test}}$. Adapting Observation 5.1 to the present situation, we have

Observation B.1. *For every* $i \in [n_{\mathsf{chal}}]$*, if the decryption of* ct_i^* *using* sk^* *does not give* $C_i(x^*)$*, i.e.* $\mathsf{I\text{-}Dec}_i = 1$*, then* $\mathsf{S\text{-}RO}_{\mathrm{Dec}\text{-}i} \bigcap (\mathsf{S\text{-}RO}_{\mathsf{Setup}} \bigcup \mathsf{S\text{-}RO}_{\mathsf{s\text{-}key}} \bigcup \mathsf{S\text{-}RO}_{\mathsf{chal}}) \not\subseteq \mathsf{S\text{-}RO}_{\Gamma}$.

Let $\textsf{I-Dec-1}_i$ and $\textsf{I-Dec-2}_i$ be indicator variables that are 1 if and only if $\textsf{S-RO}_{\text{Dec-}i} \bigcap (\textsf{S-RO}_{\textsf{s-key}} \bigcup \textsf{S-RO}_{\textsf{Setup}}) \not\subseteq \textsf{S-RO}_{\Gamma}$ and $\textsf{S-RO}_{\text{Dec-}i} \bigcap \textsf{S-RO}_{\textsf{chal}} \not\subseteq \textsf{S-RO}_{\Gamma}$, respectively. Then, $\textsf{I-Dec}_i = 1$ iff either $\textsf{I-Dec-1}_i = 1$ or $\textsf{I-Dec-2}_i = 1$ (or both). Let Bad-1 and Bad-2 be events that happen iff $\sum_{i\in[n_{\textsf{chal}}]} \textsf{I-Dec-1}_i > 1/16 \cdot n_{\textsf{chal}}$ and $\sum_{i\in[n_{\textsf{chal}}]} \textsf{I-Dec-2}_i > 1/16 \cdot n_{\textsf{chal}}$, respectively. Below we show that $\Pr[\textsf{Bad-1}] \le \textsf{negl}(\lambda)$ and $\Pr[\textsf{Bad-2}] \le 1/4$. Since $\Pr[\textsf{Bad}] \le \Pr[\textsf{Bad-1}] + \Pr[\textsf{Bad-2}]$, the lemma follows. ■

Claim B.1. $\Pr[\textsf{Bad-1}] \le \textsf{negl}(\lambda)$.

Proof. Fix any random oracle $\mathcal{O}$, the randomness used in $\textsf{Setup}^{\mathcal{O}}(1^\lambda)$, the circuit $C_{n_{\textsf{key}}+1}$, and the randomness used in $\textsf{KeyGen}^{\mathcal{O}}(\textsf{msk}, \textsf{sk}^*)$. This also fixes the sets $\textsf{S-RO}_{\textsf{Setup}}$ and $\textsf{S-RO}_{\textsf{s-key}}$. Suppose an x is picked at random $\{0,1\}^{n(\lambda)}$, and a ciphertext, ct, is generated for it by running $\textsf{Encrypt}^{\mathcal{O}}(\textsf{pk}, x)$. For $z \in \textsf{S-RO}_{\textsf{Setup}} \cup \textsf{S-RO}_{\textsf{s-key}}$, let ρ_z be the probability that z is an RO query in the decryption of ct with $\textsf{sk}^*$, where the probability is over the choice of x and the randomness used in Encrypt.

Let $X_{j,z}$ and $X^*_{i,z}$ be indicator variables that are 1 if an RO query on z is made during the jth decryption in RO query collection 2 and ith decryption in test phase, respectively, for $j \in [n_{\textsf{enc}}]$, $i \in [n_{\textsf{chal}}]$. Note that the ciphertexts $\textsf{ct}_1, \ldots, \textsf{ct}_{n_{\textsf{enc}}}$ and $\textsf{ct}^*_1, \ldots, \textsf{ct}^*_{n_{\textsf{chal}}}$ are generated independently by choosing $x_1, \ldots, x_{n_{\textsf{enc}}}$ and $x^*_1, \ldots, x^*_{n_{\textsf{chal}}}$ uniformly at random. Thus for any z, the variables $X_{1,z}, \ldots, X_{n_{\textsf{enc}},z}$ and $X^*_{1,z}, \ldots, X^*_{n_{\textsf{chal}},z}$ are independent of each other, and the probability of any of them being 1 is ρ_z.

We are interested in the probability that $\sum_{i\in[n_{\textsf{chal}}]} \textsf{I-Dec-1}_i > n_{\textsf{chal}}/16$, i.e., in at least $1/16$th fraction of the decryptions in the test phase, an RO query q is made s.t. q was also queried during set-up or key generation for $C_{n_{\textsf{key}}+1}$, but it was not captured in either of the collection phases. Thus, there must exist a z s.t. $z \notin \textsf{S-RO}_{\Gamma}$ (in particular, $z \notin \textsf{S-RO}_{\Gamma\text{-}2}$) but an RO query on z is made in at least $n_{\textsf{chal}}/16|Q|$ of the decryptions, where $Q = \textsf{S-RO}_{\textsf{Setup}} \cup \textsf{S-RO}_{\textsf{s-key}}$. (If $Q = \phi$ then Bad-1 cannot happen, and we are done.). Therefore,

$$\Pr\left[\sum_{i\in[n_{\textsf{chal}}]} \textsf{I-Dec-1}_i > \frac{n_{\textsf{chal}}}{16}\right] \quad \le \quad \sum_{z\in Q} \Pr\left[z \notin \textsf{S-RO}_{\Gamma\text{-}2} \quad \wedge \quad \sum_{i\in[n_{\textsf{chal}}]} X^*_{i,z} > \frac{n_{\textsf{chal}}}{16|Q|}\right]$$

Based on the value of ρ_z, we can divide rest of the analysis into two parts:

- If $\rho_z \ge 1/32|Q|$ then

$$\begin{aligned}\Pr[z \notin \textsf{S-RO}_{\Gamma\text{-}2}] &= \Pr[X_{1,z} = 0 \wedge \ldots \wedge X_{n_{\textsf{enc}},z} = 0] \\ &= \prod_{i\in[n_{\textsf{enc}}]} \Pr[X_{i,z} = 0] \\ &= (1-\rho_z)^{n_{\textsf{enc}}} \le e^{-n_{\textsf{enc}}/32|Q|},\end{aligned}$$

  where the second equality follows from the independence of $X_{i,z}$. Recall that we set $n_{\textsf{enc}}$ to be at least $32\lambda(q_{\textsf{Setup}} + q_{\textsf{KeyGen}})$, where $q_{\textsf{Setup}}$ and $q_{\textsf{KeyGen}}$ are upper-bounds on the number of RO queries made during Setup and KeyGen, respectively. Thus, $e^{-n_{\textsf{enc}}/32|Q|}$ is at most $e^{-\lambda}$.

- If $\rho_z < 1/32|Q|$ then expected value of $\sum_{i\in[n_{\textsf{chal}}]} X^*_{i,z}$ is at most $n_{\textsf{chal}}/32|Q|$. Using Chernoff bounds we can argue that,

$$\Pr\left[\sum_{i\in[n_{\textsf{chal}}]} X^*_{i,z} > \frac{n_{\textsf{chal}}}{16|Q|}\right] < e^{-\frac{1}{3}\cdot\frac{n_{\textsf{chal}}}{32|Q|}}.$$

  We know that $n_{\textsf{chal}} \ge n_{\textsf{enc}}$. Thus, $e^{-\frac{1}{3}\cdot\frac{n_{\textsf{chal}}}{32|Q|}}$ is at most $e^{-\lambda}$ as well.

Since Q is polynomial in the security parameter, this proves that the probability of Bad-1 is negligible as well. ■

Claim B.2. $\Pr[\textsf{Bad-2}] \le 1/4$.

Proof. Fix any random oracle $\mathcal{O}$, the randomness used in $\mathsf{Setup}^{\mathcal{O}}(1^\lambda)$, challenge messages $x_1^*, \ldots, x_{n_{\mathsf{chal}}}^*$, and the randomness used in $\mathsf{Encrypt}^{\mathcal{O}}(\mathsf{pk}, x_i^*)$ for $i \in [n_{\mathsf{chal}}]$. This, in particular, fixes ciphertexts $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{n_{\mathsf{chal}}}^*$ and the set $\mathsf{S\text{-}RO}_{\mathsf{chal}}$. Consider the following experiment: $C \xleftarrow{R} \mathcal{C}_\lambda$, $\mathsf{sk} \leftarrow \mathsf{KeyGen}^{\mathcal{O}}(\mathsf{msk}, C)$, and decrypt $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{n_{\mathsf{chal}}}^*$ using sk. Let $\hat{\rho}_z$ be the probability that at least $n_{\mathsf{chal}}/16|\hat{Q}|$ of the decryptions make an RO query on z, where $\hat{Q} = \mathsf{S\text{-}RO}_{\mathsf{chal}}$.

Let $Y_{j,z}$ be an indicator variable that is 1 iff an RO query on z is made in at least $n_{\mathsf{chal}}/16|\hat{Q}|$ of the decryptions of $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{n_{\mathsf{chal}}}^*$ with sk_j in the first phase of RO query collection. Note that the keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_{n_{\mathsf{key}}}$ are generated independently by choosing $C_1, \ldots, C_{n_{\mathsf{key}}}$ uniformly at random. Thus for any z, the variables $Y_{1,z}, \ldots, Y_{n_{\mathsf{key}},z}$ are independent of each other, and $\Pr[Y_{j,z} = 1] = \hat{\rho}_z$ for every j. In a similar way, we can also define a random variable Y_z^* that indicates whether an RO query on z is made in at least $n_{\mathsf{chal}}/16|\hat{Q}|$ of the decryptions of $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{n_{\mathsf{chal}}}^*$ with sk^* in the test phase. Y_z^* is independent of $Y_{1,z}, \ldots, Y_{n_{\mathsf{key}},z}$ and $\Pr[Y_z^* = 1] = \hat{\rho}_z$.

In a manner similar to the previous claim, we can argue that

$$\Pr\left[\sum_{i \in [n_{\mathsf{chal}}]} \mathsf{I\text{-}Dec\text{-}2}_i > \frac{n_{\mathsf{chal}}}{16}\right] \quad \leq \quad \sum_{z \in \hat{Q}} \Pr\left[z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}1} \wedge Y_z^* = 1\right]$$

If $z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}1}$, then none of the decryptions in the first phase of RO collection make a query on z. In particular, the variables $Y_{1,z}, \ldots, Y_{n_{\mathsf{key}},z}$ are all zero in such a case. Therefore,

$$\begin{aligned}
\Pr\left[z \notin \mathsf{S\text{-}RO}_{\Gamma\text{-}1} \wedge Y_z^* = 1\right] &\leq \Pr\left[Y_{1,z} = 0 \wedge \ldots \wedge Y_{n_{\mathsf{key}},z} = 0 \wedge Y_z^* = 1\right] \\
&= \Pr\left[Y_z^* = 1\right] \quad \cdot \prod_{j \in [n_{\mathsf{key}}]} \Pr\left[Y_{j,z} = 0\right] \\
&= \hat{\rho}_z(1 - \hat{\rho}_z)^{n_{\mathsf{key}}}
\end{aligned}$$

Once again we have two cases. If $\hat{\rho}_z \leq 1/4|\hat{Q}|$, then $\hat{\rho}_z(1-\hat{\rho}_z)^{n_{\mathsf{key}}}$ is at most $1/4|\hat{Q}|$ as well. Otherwise, $(1-\hat{\rho}_z)^{n_{\mathsf{key}}} \leq e^{-n_{\mathsf{key}}/4|\hat{Q}|} \leq e^{-\lambda}$ because, recall that, n_{key} is at least $4\lambda \cdot n_{\mathsf{chal}} \cdot q_{\mathsf{Enc}}$, where q_{Enc} is an upper-bound on the number of RO queries made during $\mathsf{Encrypt}$. As a result, $\sum_{z \in \hat{Q}} \hat{\rho}_z(1-\hat{\rho}_z)^{n_{\mathsf{key}}}$ is at most $1/4$. ∎

B.2 Ideal world analysis

We now show that for any PPT simulator, our adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ outputs 1 in the ideal world with negligible probability. Let t be a polynomial in λ such that $t = \ell_{\mathrm{Key}} + n_{\mathsf{enc}} \cdot q_{\mathsf{Dec}} \cdot m$ (so that $n_{\mathsf{chal}} = 16t$) where, recall that, ℓ_{ct} is the maximum length of any key generated by KeyGen.

Approximate compressibility is defined w.r.t. the experiment where several circuits are chosen at random and then evaluated at a random point (Definition 4.1). We need a slightly different notion of compressibility here, with d as an additional parameter. Suppose circuits $C_1, \ldots, C_{d+1}$ are chosen at random from $\mathcal{C}_\lambda$ and points $s_1, \ldots, s_\ell$ are chosen at random from $\mathcal{D}_\lambda$. When Cmp is given $(\{C_i\}_{i \in [d+1]}, \{C_i(x_j)\}_{i \in [d+1], j \in [\ell]})$, it must produce an output z such that when DeCmp is given $(z, \{C_i\}_{i \in [d+1]}, \{C_i(x_j)\}_{i \in [d], j \in [\ell]})$, the hamming distance of its output from $(C_{d+1}(s_1), \ldots, C_{d+1}(s_\ell))$ is at most $\epsilon \cdot t$ with probability at least η. One can show that weak pseudo-random functions for many seeds with auxiliary information (Definition 2.3) can give a $(d, 16t, t, 1/8)$ approximately incompressible family for any polynomials d and t as long as t is at least λ. Below d is set to be n_{key}.

Lemma B.2. *If $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ is an $(d, 16t, t, 1/8)$ approximately incompressible circuit family, then for any* PPT *simulator $\mathcal{S}$, the adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ outputs 1 with probability at most* $\mathsf{negl}(\lambda)$.

Proof. Suppose there exists a simulator $\mathcal{S}$ such that our adversary $\mathcal{A}$ outputs 1 with a non-negligible probability η. Like in the proof of Lemma 5.2, we will use $\mathcal{S}$ to show that $\mathcal{C}$ is $(16t, t, 1/8)$ approximately compressible. Note that $\mathcal{A}_1$ picks $x_1^*, \ldots, x_{n_{\mathsf{chal}}}^*$ and $C_{n_{\mathsf{key}}+1}$ uniformly at random and independent of its other choices. Let $r_{\mathcal{S}}$ and $r_{\mathcal{A}}$ denote the randomness used by the simulator $\mathcal{S}$ and adversary $\mathcal{A}$ (in key query 1, RO collection 2, and test phases), respectively. The compression circuit takes as input $\{C_i\}_{i \in [d+1]}$ and $\{C_i(x_j)\}_{i \in [d+1], j \in [\ell]}$, has a randomly chosen string for $r_{\mathcal{S}}$ and $r_{\mathcal{A}}$ hardwired, and works as follows:

1. Use $\mathcal{S}$ to generate a public key pk and ciphertexts $\mathsf{ct}_1^*, \ldots, \mathsf{ct}_{n_{\mathsf{chal}}}^*$. Give both to $\mathcal{A}_1$.
2. Provide $\{C_i\}_{i \in [d]}$ and $\{C_i(x_j)\}_{i \in [d], j \in [\ell]}$ to $\mathcal{S}$. It generates secret keys $\mathsf{sk}_1, \ldots, \mathsf{sk}_d$, which are given to $\mathcal{A}_2$.
3. Run the first phase of RO query collection. When $\mathcal{A}_2$ makes an RO query in this phase, forward it to $\mathcal{S}$. Give $\mathcal{S}$'s response back to $\mathcal{A}_2$.
4. Run the encryption phase to get ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_{n_{\mathsf{enc}}}$. ($\mathcal{A}_2$'s RO queries are handled in the same way as before.)
5. Provide C_{d+1} and $\{C_{d+1}(x_j)\}_{j \in [\ell]}$ to $\mathcal{S}$. It generates a secret key sk_{d+1}, which is given to $\mathcal{A}_2$.
6. Run the second phase of RO query collection. Here $\mathsf{ct}_1, \ldots, \mathsf{ct}_{n_{\mathsf{enc}}}$ is decrypted with sk_{d+1}. Let $z_1, \ldots, z_v$ be the responses to RO queries in this phase, where $z_i \in \{0,1\}^m$.
7. Output sk_{d+1} and $z_1, \ldots, z_v$.

The decompression circuit takes $\{C_i\}_{i \in [d+1]}$ and $\{C_i(x_j)\}_{i \in [d], j \in [\ell]}$ together with the compressed string $\mathsf{str\text{-}cmp}$ as inputs, which can be parsed as $\mathsf{str\text{-}cmp} = (\mathsf{sk}_{d+1}, \{z_i\})$. It also has the random value chosen before for $r_{\mathcal{S}}$ and $r_{\mathcal{A}}$ hardwired, and works as follows:

1. The first four steps are same as that in the compression circuit. Let Γ be the list of RO queries and responses recorded in the third step.
2. Run the second phase of RO query collection. The RO responses required in this step are available as part of the input $(z_1, \ldots, z_v)$. They are also added to Γ.
3. Run the test phase with the help of Γ. Let y_i' denote the outcome of decrypting ct_i^* with sk_{d+1} for $i \in [n_{\mathsf{chal}}]$.
4. Output $y_1', \ldots, y_{16t}'$.

We need to show that the decompression property works with probability η. When $C_1, \ldots, C_{d+1}, x_1, \ldots, x_{16t}$ are chosen uniformly at random, then it is easy to see that the decompression circuit emulates the ideal world experiment perfectly. We know that $\mathcal{A}_2$ outputs 1 if and only if for at least $7/8$th of the decryptions, $y_i' = y_i$. Hence, if 1 is output with probability η, then the hamming distance is at most $1/8$ with probability at least η. ■