Upgrading to Functional Encryption

Saikrishna Badrinarayanan * Dakshita Khurana † Amit Sahai ‡ Brent Waters §

Abstract

The notion of Functional Encryption (FE) has recently emerged as a strong primitive with several exciting applications. In this work, we initiate the study of the following question: Can existing public key encryption schemes be "upgraded" to Functional Encryption schemes without changing their public keys or the encryption algorithm? We call a public-key encryption with this property to be *FE-compatible*.

Indeed, assuming ideal obfuscation, it is easy to see that every CCA-secure public-key encryption scheme is FE-compatible. Despite the recent success in using indistinguishability obfuscation to replace ideal obfuscation for many applications, we show that this phenomenon most likely will not apply here. We show that assuming fully homomorphic encryption and the learning with errors (LWE) assumption, there exists a CCA-secure encryption scheme that is provably *not FE-compatible*. We also show that a large class of natural CCA-secure encryption schemes proven secure in the random oracle model are not FE-compatible in the random oracle model.

Nevertheless, we identify a key structure that, if present, is sufficient to provide FE-compatibility. Specifically, we show that assuming sub-exponentially secure iO and sub-exponentially secure one way functions, there exists a class of public key encryption schemes which we call *Special-CCA* secure encryption schemes that are in fact, FE-compatible.

In particular, each of the following popular CCA secure encryption schemes (some of which existed even before the notion of FE was introduced) fall into the class of *Special-CCA* secure encryption schemes and are thus FE-compatible:

1. [CHK04] when instantiated with the IBE scheme of [BB04].
2. [CHK04] when instantiated with any Hierarchical IBE scheme.
3. [PW08] when instantiated with any Lossy Trapdoor Function.

*UCLA. Email: saikrishna@cs.ucla.edu
†UCLA. Email: dakshita@cs.ucla.edu
‡UCLA. Email: sahai@cs.ucla.edu.
§UT Austin. Email: bwaters@cs.utexas.edu.

Contents

1 Introduction **4**

2 Technical Overview **7**
2.1 Organization 11

3 Preliminaries **11**

4 Defining Functional Encryption Compatibility **11**
4.1 Public Key Encryption 11
4.2 Functional Encryption 12
4.2.1 Security 12
4.3 FE-Compatibility 13

5 An Impossibility Result **13**
5.1 An Attack 15
5.2 IND-CCA Security 16
5.3 Proof of Claim 1 19

6 On the Difficulty of Proving Functional Encryption Compatibility for Fujisaki-Okamoto and RSA-OAEP **22**
6.1 Construction 22
6.2 Attack 23

7 Building FE-Compatible Encryption Schemes **24**
7.1 Puncturable Tag Based Encryption 24
7.2 Special-CCA secure encryption scheme 26
7.3 Instantiating Special-CCA encryption 27
7.4 Building selectively secure FE 27
7.4.1 Security Proof 29

8 Acknowledgements **35**

A Security Notions for Public Key Encryption **38**
A.1 CCA Security 38
A.2 CPA Security 38

B Further Preliminaries **38**
B.1 Indistinguishability Obfuscation 38
B.2 Differing Inputs Obfuscation 39
B.3 Lockable Obfuscation 40
B.3.1 Correctness 40
B.3.2 Security 40

C Proof of Security for Special-CCA **41**

D Examples of Special-CCA Secure Encryption Schemes **43**
D.1 Scheme in [CHK04] 43
D.1.1 [CHK04] using IBE scheme in [BB04] 44
D.1.2 [CHK04] using any HIBE scheme 46
D.2 Scheme in [PW08] 47

E Key Only FE-Compatibility **49**
E.1 Construction from iO 50

1 Introduction

Functional Encryption (FE) [SW05, SW08] is a powerful framework that significantly expands the scope of public-key encryption. In an ordinary public-key encryption scheme, a user Alice first chooses a public key PK and a corresponding secret key SK using a (master) setup algorithm Setup. Then, any other user Bob can use Alice's public key to encrypt a message m to obtain a ciphertext $c = \mathsf{Enc}(\mathsf{PK}, m)$. Alice can decrypt this ciphertext using her secret key, yielding $m = \mathsf{Dec}(\mathsf{SK}, c)$.

In a functional encryption scheme, we give Alice key delegation capabilities: Alice can use a new key generation algorithm KeyGen to generate a *functional key* $\mathsf{SK}_f = \mathsf{FE.KeyGen}(\mathsf{SK}, f)$ for a function f that is, say, described by a circuit. Then Alice can hand this functional key SK_f to an associate Charlie, and Charlie can use this functional key together with a new decryption algorithm to only learn $f(m) = \mathsf{FE.Dec}(\mathsf{SK}_f, c)$ when given the ciphertext c. Intuitively speaking, nothing[1] beyond $f(m)$ should be learned by Charlie when given SK_f and c. This notion was fully formalized by [BSW11] in the setting where many functional keys and ciphertexts may be given to an adversary. The first work achieving functional encryption for general functions was [GGH+13], using the power of indistinguishability obfuscation.

The work of [BSW11] gave several compelling applications of functional encryption. For instance, Alice may want to store her e-mail in encrypted form, but she wants her cloud provider to be able to execute a phishing-detection circuit C on her email prior to sending it to her for decryption. She could accomplish this goal by providing her cloud provider with a functional key for SK_C, and the only thing the cloud provider would learn is whether any email received by Alice satisfies the phishing-detection circuit.

Applications of functional encryption become even more compelling when we think of Alice as representing a large organization or company. In such a scenario, the threat that functional encryption helps to address cryptographically is the insider threat. For example, consider an organization like a government tax authority, that regularly handles extremely sensitive information, but where individuals within the organization should only have access to limited digests or snippets of this sensitive information. For example, an analyst Dave at the tax authority may need only to compute statistical summaries of tax returns filed by a large set of people. Functional encryption would allow Dave to obtain a functional key SK_T, where T is the description of a function that produces statistical summaries of tax returns. The security of functional encryption would guarantee that even if Dave goes rogue, Dave's functional key would only allow him to learn and exfiltrate statistical summaries, and not any more personal information about individual tax returns beyond what could be deduced from the statistical summary.

Contrast this to the case where only ordinary public-key encryption is used to encrypt tax information. In this case, Dave would need the (master) secret key SK in order to decrypt tax information before processing it to obtain statistical summaries. And therefore a rogue Dave could exfiltrate the personal details of any person's tax return that was an input to the statistical summary he was supposed to compute. This is just one example, illustrative of many such scenarios where functional encryption could be beneficial for security.

Upgrading to Functional Encryption. Suppose that some time in the future, an organization, upon hearing about the advantages of functional encryption, wishes to "upgrade" to use functional encryption. Such an organization may face many challenges. First, the organization may already have infrastructure in place where partners and clients use an existing public-key encryption scheme

[1] Slightly more formally, functional encryption requires that encryptions of two messages m_0 and m_1 should be indistinguishable when given functional keys corresponding to any functions f that satisfy $f(m_0) = f(m_1)$. See Section 4 for more details.

to communicate with the organization. As such, the organization may have already amassed large amounts of encrypted data using a legacy public-key encryption system. Second, the organization may face regulatory burdens like HIPAA or other future regulations, that require the organization to use a particular encryption algorithm. Third, it could be that, even in this future time, existing key generation algorithms for general-purpose functional encryption (which typically currently use indistinguishability obfuscation) are too slow, but the organization wants to be ready for the day when such algorithms become practical.

In light of these concerns, what public-key encryption algorithm should the organization use now? While these are mostly societal challenges, security must exist in the context of human societies with traditions, rules, and regulations. And in this case, these concerns give rise to an intriguing theoretical question:

What (existing) public-key encryption algorithms can be "upgraded" to become functional encryption schemes, without changing the encryption algorithm or the public keys?

Our paper initiates the systematic study of this question. To formalize this, we say that a public-key encryption scheme E is *FE-compatible* if there exist new key generation and decryption algorithms that, when combined with the original setup and encryption algorithms of E, yield a (selectively) secure functional encryption scheme. (See Section 4 for details.)

Necessary Conditions. The technical starting point for our work is the folklore observation that any functional encryption scheme must satisfy a certain level of non-malleability. To see why, consider a functional encryption scheme for encrypting $(n+1)$-bit messages m, and consider the function f_1 that on input m simply outputs the first n bits of m. Suppose that we obtain a functional key SK_{f_1} for this function. Then functional encryption guarantees that encryptions of any two messages with identical n-bit prefixes should still be indistinguishable from each other.

But suppose there was a way for an adversary to modify any encryption $\mathsf{FE.Enc}(m)$ to obtain $\mathsf{FE.Enc}(m')$ where m' swapped the first and last bits of m. This would, for example, easily be possible if one tried to encrypt the message bit-by-bit. Then, by applying the functional key SK_{f_1} to $\mathsf{FE.Enc}(m')$, the adversary would learn the last bit of m, and break the security that is supposed to be guaranteed by functional encryption.

Indeed, it is not hard to see that the above argument generalizes to guarantee a type of security against chosen-ciphertext attacks. Thus, (a form of) CCA-security is a necessary requirement for an encryption scheme to be FE-compatible.

Universal Functional Encryption? At this point, it might be tempting to consider the possibility that CCA-security is also a *sufficient* condition for being FE-compatible. Indeed, this would be true if we had ideal obfuscation[2] [Had00] – that is, obfuscation that creates the equivalent of a virtual black box. It is not difficult to see why: To create a functional key SK_f, simply obfuscate the function that uses SK as a hardwired constant to decrypt the input ciphertext c to obtain the message m, and then simply output $f(m)$. If the obfuscation is ideal, then this functional key can easily be simulated as a black box just by using the CCA-decryption oracle for decryption. Thus, given ideal obfuscation, every CCA-secure public-key encryption scheme is FE-compatible. In this sense, we could hope to have a kind of universal functional encryption (in the sense of universal deniable encryption [SW14] or universal signature aggregators [HKW15]), where the key generation construction above could be applied to any CCA-secure encryption scheme.

[2]Note that ideal obfuscation is impossible to build.

Recently our field has had remarkable success in achieving results using indistinguishability obfuscation that were previously known to be possible only using ideal obfuscation, especially using the punctured programming paradigm of [SW14]. Is this just a matter of applying enough "$i\mathcal{O}$ gymnastics" to make this work?

Our Results. In our first result, somewhat surprisingly, we show that in this case, the intuition based on ideal obfuscation is wrong. Specifically, we show the following:

Informal Theorem 1. *Assuming CCA-secure public-key encryption, fully homomorphic encryption (FHE) and LWE, there exists a CCA-secure public-key encryption scheme that is provably not FE-compatible.*

The construction we give in the impossibility result above is quite contrived, like most impossibility results of this type. Could it be that all "natural" CCA-secure public-key encryption schemes are FE-compatible? Sadly, we do not know how to answer, or even formally define, this question. Nevertheless, one natural setting in which to consider this question is the well-studied random oracle model; this model allows for very simple and intuitive proofs of CCA-security, via the popular Fujisaki-Okamoto [FO99] transformation. In the random oracle model, however, we show an even stronger negative result: Every public-key encryption scheme, when converted into a CCA-secure encryption scheme in the random oracle model via the Fujisaki-Okamoto transformation, is provably not FE-compatible in the random oracle model. Thus, in the random oracle model, we obtain a large family of natural CCA-secure schemes [3] that are not FE-compatible. [4]

In light of the impossibility results above, we believe that a systematic study of FE-compatibility will need to proceed in a "bottom-up" manner, by looking at existing classes of CCA-secure encryption schemes and seeing if they can indeed be FE-compatible. We initiate this line of study by identifying a key structure that, if present, is sufficient to provide FE-compatibility. Specifically, we show the following:

Informal Theorem 2. *Assuming sub-exponentially secure iO and sub-exponentially secure one way functions, there exists a class of public key encryption schemes which we call* Special-CCA *secure encryption schemes that are FE-compatible.*

We then note that several existing CCA-secure encryption schemes fall into the class of *Special-CCA* secure encryption schemes. As a result, we get the following theorem:

Informal Theorem 3. *Assuming sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure one way functions, each of the following existing CCA-secure encryption schemes are FE-compatible:*

- [*CHK04*] *when instantiated with the IBE scheme of* [*BB04*].
- [*CHK04*] *when instantiated with any Hierarchical IBE scheme.*
- [*PW08*] *when instantiated with any Lossy Trapdoor Function.*

[3] We believe similarly structured transformation such as RSA-OAEP [BR94] will have the same issues.

[4] Interestingly, if the scheme is instantiated with a particular hash function family it might actually be FE-compatible. This is somewhat the opposite of a typical RO infeasibility results where one usually finds a scheme is provably secure in the RO model, but is insecure under any concrete instantiation. Unfortunately, it is unclear how to argue positive security of any such concrete FO instantiations as the usual RO heuristic is now off limits.

It is interesting to note that the above CCA-secure encryption schemes are each at least 9 years old, and yet they can be used to build functional encryption schemes without changing the encryption mechanism. Contrast this to existing functional encryption schemes before our work, most of which have specifically designed encryption methods using "$i\mathcal{O}$-friendly" tools.

Finally, we also consider a weaker notion called *key-only FE-compatibility* where we retain only the public key and secret key of the public key encryption scheme and design new encryption, function secret key generation and decryption algorithms to "upgrade" it to a FE scheme. In the common random string model, we show that assuming polynomially hard iO, every public key encryption scheme is key-only FE compatible - that is, it can be upgraded to a selectively secure FE scheme for any function family.

Open problems and future work. It would be interesting to understand if there exists other classes of encryption schemes that are FE-compatible. More generally, an interesting open problem would be to study what is the exact type of CCA-security needed for an encryption scheme to be FE-compatible.

While it is known that general purpose functional encryption implies indistinguishability obfuscation, another interesting direction would be to weaken the security requirement of functional encryption (for example, bounded-key secure FE) and understand what class of encryption schemes can be upgraded without the use of indistinguishability obfuscation. A solution in this setting might also be practical in today's world. Going in the other direction, an interesting feasibility question is whether we can upgrade existing encryption schemes to achieve general purpose multi-input functional encryption [GGG+14, BGJS15].

Finally, we observe that in our positive result, on upgrading the CCA secure encryption schemes into an FE scheme, it may potentially lose the CCA property. It is an interesting open problem to define and achieve FE-CCA compatibility[5].

2 Technical Overview

The question at the core of this paper is: what kinds of public-key encryption schemes can be "upgraded" to yield functional encryption schemes? Informally speaking, we say that a public-key encryption scheme PKE is *FE-compatible* if a functional encryption scheme can be generated where the setup and encryption algorithms of the functional encryption scheme are the same as the public-key encryption scheme. Namely, we have $\mathsf{FE.Setup} = \mathsf{PKE.Setup}$ and $\mathsf{FE.Enc} = \mathsf{PKE.Enc}$. Thus, only the functional encryption key generation and decryption algorithms are allowed to be newly specified.

As already noted, the technical starting point for our work is the folklore observation that any functional encryption scheme must satisfy a certain level of non-malleability. To remind ourselves why, consider a functional encryption scheme for encrypting $(n+1)$-bit messages m, and consider the function f_1 that on input m simply outputs the first n bits of m. Suppose that we obtain a functional key SK_{f_1} for this function. Then functional encryption guarantees that encryptions of any two messages with identical n-bit prefixes should still be indistinguishable from each other.

But suppose there was a way for an adversary to modify any encryption $\mathsf{FE.Enc}(m)$ to obtain $\mathsf{FE.Enc}(m')$ where m' swapped the first and last bits of m. This would, for example, easily be possible if one tried to encrypt the message bit-by-bit. Then, by applying the functional key SK_{f_1} to $\mathsf{FE.Enc}(m')$, the adversary would learn the last bit of m, and break the security that is supposed to be guaranteed by functional encryption.

[5]Note that our negative result would still hold in this stronger model of FE-CCA compatibility.

An impossibility result. The most natural question to ask, then, is whether CCA-security is also a sufficient condition for FE-compatibility. In our first result, we prove that this is indeed not the case: we construct a counterexample public-key encryption scheme that satisfies CCA-security, but provably is not FE-compatible.

Let us build some intuition for how our impossibility result will proceed. The main difference between the CCA security game and the FE security game is that in the CCA security game, there is a decryption *oracle*, whereas in the FE security game, the adversary can actually obtain a circuit that will (at least partially) decrypt ciphertexts. This is reminiscent of the situation underlying the impossibility result of Barak et al. [BGI+01] for virtual black-box obfuscation: There, the ideal model gave oracle access to the function to be obfuscated, whereas the real model gave the adversary an actual circuit implementing that function. Indeed, we draw inspiration from [BGI+01] in devising our negative result, although we differ from it in almost every technical respect.

The idea behind our negative result will be to take an arbitrary CCA-secure encryption scheme $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ and somehow "damage" it to make it FE-incompatible, without disturbing its CCA security. This "damaged scheme" must somehow make use of the fact that an FE-adversary will be able to ask for and obtain a functional key SK_{f_1}, let us say for the same prefix-revealing function f_1 that we defined above. This functional key SK_{f_1} enables the FE-adversary to compute a prefix-decryption circuit D that outputs the first n bits of the message corresponding to any ciphertext.

Our first idea (which conceptually dates back to [BGI+01]) is to use fully homomorphic encryption (FHE) to help us take advantage of this situation. We first choose a random n-bit string α, and encrypt it $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha||0)$ using the CCA-secure encryption scheme. But then we re-encrypt this $c' = \mathsf{FHE}(c)$ using the fully homomorphic encryption scheme. We reveal c' as part of the public key of the "damaged scheme," but crucially both α and c are kept hidden.

Why does this help? Because now an FE-adversary that obtains the prefix-decryption circuit D can compute $\mathsf{FHE.Eval}(D, c') = \mathsf{FHE}(\alpha)$. While it is not yet clear that this is useful for any attack, we observe that, at least intuitively, a CCA-attacker has no obvious way to obtaining $\mathsf{FHE}(\alpha)$ from the public key and the decryption oracle (though formally proving this will be the main technical challenge of our impossibility result, as we will discuss shortly). This is because the only information that the CCA-attacker has about α is contained in c', but c' is an encryption under FHE and the decryption oracle only decrypts ciphertexts validly encrypted using $\mathsf{Enc}_{\mathsf{CCA}}$.

To enable a real attack, then, we also add to the public key an obfuscation of a program P that takes as input an FHE ciphertext e, decrypts it, and checks whether this decryption is equal to α. If so, it outputs the secret key needed for executing $\mathsf{Dec}_{\mathsf{CCA}}$, otherwise it outputs $\perp$. Because the FE-attacker can obtain $\mathsf{FHE}(\alpha)$ as noted above, it can then use the obfuscated program to obtain the full secret key for executing $\mathsf{Dec}_{\mathsf{CCA}}$, breaking the security of the FE scheme.

Why these changes preserve CCA security. The changes above – adding the FHE ciphertext c' and the obfuscated program P to the public key – only provide an impossibility result if CCA security is preserved even after these two objects are added to the public key. While it is not obvious how a CCA-attacker could use these objects to break security, in order to prove CCA security, intuitively we will need to remove the dependence of c' on α. But $c' = \mathsf{FHE}(\mathsf{Enc}_{\mathsf{CCA}}(\alpha||0))$, and the obfuscated program P contains the secret key for FHE. But in order to remove these secret keys from P, intuitively we need to remove the "trigger" point $\mathsf{FHE}(\alpha)$ from the code of P, for which we first need to remove the dependence of c' on α. This chicken-and-egg situation is the primary technical obstacle that we need to overcome to finish the proof.

To deal with this problem, we draw inspiration from the work of Myers and Shelat [MS09] and

Hohenberger, Lewko and Waters [HLW12] that considered the seemingly very different problem of converting any CCA-secure encryption scheme for single-bit messages into a CCA-secure encryption scheme for multi-bit messages. However, to implement our inspiration, we will need to make a technical change to the encryption system. Instead of using $\mathsf{Enc}_{\mathsf{CCA}}$ to encrypt the entire $n+1$-bit message, we will use the CCA-secure encryption schemes to encrypt the first n bits of the message, and use a separate encryption scheme $\mathsf{Enc}_{\mathsf{CPA}}$ to encrypt the last bit of the message. (In fact, we will use $\mathsf{Enc}_{\mathsf{CCA}}$ to jointly encrypt the first n bits of the message and the ciphertext produced by $\mathsf{Enc}_{\mathsf{CPA}}$. But we will ignore this detail for the purpose of this overview.) Finally, we will change our obfuscated program P to output just the secret key for executing $\mathsf{Dec}_{\mathsf{CPA}}$ to decrypt the last bit. This way, the secret key for executing $\mathsf{Dec}_{\mathsf{CCA}}$ is independent of the program P. Now, we will define a Bad Event to be when a CCA-attacker queries its decryption oracle on the ciphertext $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha)$. Looking ahead, we will first consider the situation when this Bad Event does not happen. Then, we will show that indeed the Bad Event can only occur with negligible probability.

Suppose that we know that the Bad Event cannot happen. Then, the decryption oracle given to the CCA-attacker is equivalent to a decryption oracle that would be given to a CCA-attacker if $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha)$ was the "challenge" ciphertext on which the attacker is not allowed to query. Note that in this case, the CCA security of $\mathsf{Enc}_{\mathsf{CCA}}$ already guarantees that $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha)$ is indistinguishable from $c = \mathsf{Enc}_{\mathsf{CCA}}(0^n)$, even to an adversary that is given the obfuscated program P as auxiliary information about α. Thus, we can already remove the dependence of c' on α.

Now, the only part of the public key that depends on α is the obfuscated program P, and we just need to get rid of it. This could be accomplished via $i\mathcal{O}$ using the fact that α is a uniformly random string, but in fact our job is made even easier due to the recent works on "lockable obfuscation" of Goyal et al. [GKW17] and Wichs and Zirdelis [WZ17]. These works consider obfuscating programs $C(x)$ whose structure is exactly such that, for some circuit Test if $\mathsf{Test}(x) = \alpha$, then some secret β is revealed, and otherwise the output is $\perp$. Lockable obfuscation states that if α is chosen uniformly (and, for our setting, no auxiliary information about α is revealed), then such obfuscated programs are indistinguishable from obfuscated programs that always output $\perp$ and have no secrets within them whatsoever. Furthermore, such lockable obfuscation is possible to construct just assuming LWE for suitable parameters. Thus, applying the security of lockable obfuscation, we are able to replace the obfuscated program P with a program that always outputs $\perp$, thereby completely removing any information about the secret keys of any encryption scheme and about α. This shows that the new scheme is CCA-secure, under the assumption that the Bad Event does not occur.

All that remains to be done is to prove that the Bad Event does not occur. Counterintuitively, we first observe that the lockable obfuscation argument above already shows that the Bad Event cannot occur if the ciphertext c had been $c = \mathsf{Enc}_{\mathsf{CCA}}(0^n)$ instead of $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha)$. In other words, if $c = \mathsf{Enc}_{\mathsf{CCA}}(0^n)$, then the adversary never queries the decryption oracle with c. Now, suppose for sake of contradiction, that the adversary does query c with noticeable probability if $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha)$. Then, we can use this to break CCA-security of $\mathsf{Enc}_{\mathsf{CCA}}$; take as a challenge ciphertext c that is either $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha)$ or $c = \mathsf{Enc}_{\mathsf{CCA}}(0^n)$. Then run the adversary until it attempts to query the oracle on c. If it ever does this, we can conclude that $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha)$. If it doesn't, then we can output a random guess. This will give us an nontrivial advantage in determining whether $c = \mathsf{Enc}_{\mathsf{CCA}}(\alpha)$ or $c = \mathsf{Enc}_{\mathsf{CCA}}(0^n)$.

This completes the impossibility proof. Full details can be found in Section 5.

For the impossibility result that applies to CCA secure encryption schemes built using the Fujisaki-Okamoto transformation in the random oracle model, we refer the reader to Section 6 for the techniques used.

Positive Results for FE-compatibility. Our impossibility result shows that CCA security is not a sufficient condition for an encryption scheme to be FE-compatible. On the other hand, unfortunately positive results on FE in the literature (e.g. [GGH+13, Wat14]) typically construct special-purpose encryption methods that are atypical for achieving CCA security. For instance, even though the original general-purpose FE scheme of [GGH+13] follows the Naor-Yung paradigm [NY90, Sah99], instead of using a simulation-sound NIZK in the encryption, it uses a special object introduced in [GGH+13] called a statistically simulation-sound NIZK. Recall that our goal is to find existing CCA-secure encryption schemes that are already FE-compatible, rather than design special-purpose (sometimes called "$i\mathcal{O}$ friendly") primitives that would enable FE.

How can we go about this? Let us try to see if there are encryption mechanisms that were useful in achieving CCA-security that can also be sufficient for achieving FE.

Our key observation is that the notion of a *punctured* decryption key, which has implicitly been used for building CCA-security for over a decade, since (at least) the work of [CHK04], can also be useful for building FE functional keys. Roughly speaking, we consider the notion of a tag-based encryption, where every ciphertext is associated with a tag. Then, a punctured decryption key $\mathsf{SK}_{\mathsf{tag}^*}$ should allow a user to decrypt every ciphertext with $\mathsf{tag} \neq \mathsf{tag}^*$, but messages encrypted under tag tag^* should still be semantically secure. Intuitively, such punctured keys have been useful for building CCA-secure encryption because a punctured decryption key would allow the implementation of a decryption oracle that would still not be able to decrypt a challenge message that was encrypted under tag tag^*. In the literature, such schemes are combined with one-time signature schemes, where the tag is set to be the verification key of such a one-time signature scheme, and then the ciphertext is signed in a way that verifies with this key.

How can we use this idea for building FE functional keys? At a high level, we start with the most basic idea for building a functional key for a function f. We can simply obfuscate a program that has the decryption key built in, uses this decryption key to decrypt the message m, and then outputs $f(m)$. Now, we need to argue that the encryption of m_0 and the encryption of m_1 should be indistinguishable as long as $f(m_0) = f(m_1) = y$. The first idea is to fix the verification key VK^* in advance that will be used as the tag for the challenge ciphertext c^*. Now, we can reformulate the obfuscated program to first check whether the input ciphertext is equal to c^*, in which case the program should output y, but otherwise it should just use the decryption key to decrypt the message m, and then output $f(m)$ as before. This program is functionally equivalent to the previous one, and therefore indistinguishability of obfuscated programs follows from $i\mathcal{O}$.

Now, our goal will be to replace the decryption key within the program with the punctured decryption key $\mathsf{SK}_{\mathsf{VK}^*}$. However, note that we cannot do that immediately, because there are many valid ciphertexts for various messages m that could be signed under verification key VK^*, on which the program is supposed to output $f(m)$. However, we know that it should be hard for the adversary to actually find such valid ciphertexts, because of the security of the one-time signature scheme. Here, we can use sub-exponentially secure $i\mathcal{O}$ to complete the argument: Roughly speaking, the work of [BCP14] shows that if an $i\mathcal{O}$ scheme is secure against time $T \cdot \mathsf{poly}(\mathsf{n})$ adversaries, then $i\mathcal{O}(P_1)$ and $i\mathcal{O}(P_2)$ are indistinguishable as long as: (1) they only differ on at most T inputs, and (2) these inputs are hard to find even if given the code of both P_1 and P_2, even for machines whose running time far exceed T. By using this, assuming also sub-exponentially secure one-time signatures (which follow from sub-exponentially strong one-way functions), we can replace the program with one that first checks whether the input ciphertext is equal to c^*, in which case the program outputs y, but otherwise it uses the *punctured* decryption key $\mathsf{SK}_{\mathsf{VK}^*}$ to decrypt the message m, and then output $f(m)$ as before.

Now, since only this punctured decryption key $\mathsf{SK}_{\mathsf{VK}^*}$ is used, we can argue that an encryption of m_0 under tag VK^* is indistinguishable from an encryption of m_1 under tag VK^*. Thus, we show how

to bootstrap punctured decryption keys as an existing method for building CCA-secure encryption, into a method for constructing functional keys without needing to change the underlying encryption scheme. Interestingly, the security of the encryption given a punctured decryption key needs to hold only against polynomial-time adversaries, as in standard proofs of CCA-security.

We observe that at least three different existing CCA-secure schemes from the literature, some dating back over a decade, already follow the punctured key approach to building CCA-secure encryption, and therefore are FE-compatible. Full details can be found in Section 7.

2.1 Organization

In Section 3, we define some preliminaries. This is followed by the definition of FE-compatibility in Section 4. In Section 5 we show the impossibility result. This is followed by the impossibility result in the random oracle model in Section 6. Finally, in Section 7, we show the constructions of FE-compatible CCA secure encryption schemes. In Appendix E, we discuss the weaker notion of Key Only FE-Compatibility.

3 Preliminaries

We define the following primitives in Appendix B : indistinguishability obfuscation, differing inputs obfuscation, lockable obfuscation. We refer the reader to [Gen09, Gen10] for a definition of fully homomorphic encryption.

4 Defining Functional Encryption Compatibility

Throughout, let the security parameter be denoted by n. Let $\mathcal{X} = \{\mathcal{X}_n\}_{n\in\mathbf{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_n\}_{n\in\mathbf{N}}$ denote ensembles where each $\mathcal{X}_n$ and $\mathcal{Y}_n$ is a finite set. Let $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbf{N}}$ denote an ensemble where each $\mathcal{F}_n$ is a finite collection of functions, and each function $f \in \mathcal{F}_n$ takes as input a string $x \in \mathcal{X}_n$ and outputs $f(x) \in \mathcal{Y}_n$.

We first define public key encryption in the next subsection. This is followed by the definition of functional encryption(FE) in the subsequent subsection and then finally, we define what it means for a public key encryption scheme to be FE-Compatible.

4.1 Public Key Encryption

A public key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ for a family of message spaces $\{\mathcal{X}_n\}$ consists of the following algorithms:

- $\mathsf{PKE.Setup}(1^n)$:
  Given the security parameter n, it generates a public key PK and a secret key SK.
- $\mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m})$:
  Given a message $\mathsf{m} \in \mathcal{X}_n$ and the public key PK as input, the encryption algorithm outputs a ciphertext CT.
- $\mathsf{PKE.Dec}(\mathsf{MSK}, \mathsf{CT})$:
  Given a ciphertext CT and the secret key SK as input, the decryption algorithm outputs a string $y \in \mathcal{X}_n$ or $\perp$.

Correctness: A public key encryption scheme PKE is correct if for all messages $\mathsf{m} \in \mathcal{X}_n$

$$\Pr\left[\begin{array}{c}(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{PKE.Setup}(1^{\mathsf{n}}) \\ \mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m})) = \mathsf{m}\end{array}\right] = 1$$

The probability is over the randomness used in the setup, encryption and decryption algorithms above.

Security Notions: We consider two notions of security namely IND-CPA security and IND-CCA security. We define them formally in Appendix A. It is well known that any IND-CCA secure public key encryption scheme is also IND-CPA secure.

4.2 Functional Encryption

A functional encryption scheme $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Keygen}, \mathsf{FE.Dec})$ for a family of message spaces $\{\mathcal{X}_n\}$, a family of output spaces $\{\mathcal{Y}_n\}$ and a family of functions $\mathcal{F}$ consists of the following polynomial time algorithms:

- $\mathsf{FE.Setup}(1^n)$. The setup algorithm takes as input the security parameter n and outputs a master public key-secret key pair $(\mathsf{MPK}, \mathsf{MSK})$.
- $\mathsf{FE.Enc}(\mathsf{MPK}, x) \rightarrow \mathsf{CT}$. The encryption algorithm takes as input a message $x \in \mathcal{X}_n$ and the master public key MPK. It outputs a ciphertext CT.
- $\mathsf{FE.Keygen}(\mathsf{MSK}, f) \rightarrow \mathsf{SK}_f$. The key generation algorithm takes as input a function $f \in \mathcal{F}_n$ and the master secret key MSK. It outputs a function secret key SK_f.
- $\mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{CT}) \rightarrow y$. The decryption algorithm takes as input a secret key SK_f and a ciphertext CT. It outputs a string $y \in \mathcal{Y}_n$ or $\perp$.

Definition 1. *(Correctness) A functional encryption scheme* FE *for* $\mathcal{F}$ *is correct if for all* $f \in \mathcal{F}_n$ *and all* $x \in \mathcal{X}_n$

$$\Pr\left[\begin{array}{c}(\mathsf{MPK},\mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^{\mathsf{n}}) \\ \mathsf{SK}_f \leftarrow \mathsf{FE.Keygen}(\mathsf{MSK}, f) \\ \mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{FE.Enc}(\mathsf{MPK}, x)) = f(x)\end{array}\right] = 1$$

where the probability is over the random coins of $\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Keygen}$ *and* $\mathsf{FE.Dec}$.

4.2.1 Security

We define the security notion for a functional encryption scheme using the following game $(\mathsf{Adaptive} - \mathsf{IND})$ between a challenger and an adversary.

Setup Phase: The challenger generates $(\mathsf{MPK}, \mathsf{MSK}) \leftarrow \mathsf{FE.Setup}(1^{\mathsf{n}})$ and then hands over the master public key MPK to the adversary.
Key Query Phase 1: The adversary makes function secret key queries by submitting functions $\mathsf{f} \in \mathcal{F}_n$. The challenger responds by giving the adversary the corresponding function secret key $\mathsf{SK}_\mathsf{f} \leftarrow \mathsf{FE.KeyGen}(\mathsf{MSK}, \mathsf{f})$.
Challenge Phase: The adversary chooses two messages $(\mathsf{m}_0, \mathsf{m}_1)$ of the same size (each in $\mathcal{X}_n$) such that for all queried functions f in the key query phase, it holds that $\mathsf{f}(\mathsf{m}_0) = \mathsf{f}(\mathsf{m}_1)$. The challenger selects a random bit $\mathsf{b} \in \{0, 1\}$ and sends a ciphertext $\mathsf{CT} \leftarrow \mathsf{FE.Enc}(\mathsf{MPK}, \mathsf{m}_\mathsf{b})$ to the

adversary.
Key Query Phase 2: The adversary may submit additional key queries $\mathsf{f} \in \mathcal{F}_n$ as long as they do not violate the constraint described above. That is, for all queries f, it must hold that $\mathsf{f}(\mathsf{m}_0) = \mathsf{f}(\mathsf{m}_1)$.
Guess: The adversary submits a guess b' and wins if $b' = b$. The adversary's advantage in this game is defined to be $2 * |\mathsf{Pr}[\mathsf{b} = \mathsf{b}'] - 1/2|$.

We also define the *selective* security game, which we call $(\mathsf{Selective} - \mathsf{IND})$ where the adversary outputs the challenge message pair even before seeing the master public key.

Definition 2. *A functional encryption scheme* FE *is selective/adaptive secure if all PPT adversaries have at most a negligible advantage in the* $\mathsf{Selective} - \mathsf{IND}/\mathsf{Adaptive} - \mathsf{IND}$ *security game.*

We can also parameterize by the number of function secret key queries the adversary can make in the security game.
Compactness[AJ15] : A functional encryption scheme is said to be compact if the size of the ciphertext does not depend on the size of the functions that the scheme can handle. That is, let $p(\cdot)$ be a polynomial. Now, any functional encryption scheme FE for a class of functions $\mathcal{F}$ is said to be compact if $|\mathsf{FE.Enc}(\mathsf{MPK}, x)| = p(n, |x|)$ where n is the security parameter.

4.3 FE-Compatibility

In this section, we define a property called FE-Compatibility for any public key encryption scheme.

Definition 3. *A public key encryption scheme* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is said to be selective/adaptive FE-Compatible relative to a family of functions* $\mathcal{F}$ *if there exists two algorithms* $(\mathsf{FE.Keygen}, \mathsf{FE.Dec})$ *such that* $(\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Keygen}, \mathsf{FE.Dec})$ *is a selectively/adaptively secure functional encryption scheme for the family* $\mathcal{F}$ *where:*

- $\mathsf{FE.Setup}(n) = \mathsf{PKE.Setup}(n)$. *In particular, if* $\mathsf{PKE.Setup}(n)$ *outputs* $(\mathsf{PK}, \mathsf{SK})$, *the output of* $\mathsf{FE.Setup}(n)$ *is* $(\mathsf{MPK} = \mathsf{PK}, \mathsf{MSK} = \mathsf{SK})$.
- $\mathsf{FE.Enc}(\mathsf{MPK}, \mathsf{m}) = \mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m})$.

Moreover, any such FE scheme is also compact because the size of the ciphertext is determined by the scheme PKE *and doesn't depend on the size of the functions being queried.*

5 An Impossibility Result

In this section, we will construct an IND-CCA secure encryption scheme that is not FE-Compatible according to Definition 11. Consider a function f_1 that on any input x of length $(n+1)$ bits, outputs the first n bits of x. Formally, we prove the following theorem:

Theorem 1. *Assuming the existence of lockable obfuscation, fully homomorphic encryption and IND-CCA secure public key encryption, the scheme* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *described below is an IND-CCA secure public key encryption scheme that is not selective FE-Compatible even for a single function secret key query for any function family* $\mathcal{F}$ *such that* $f_1 \in \mathcal{F}$.

We know how to construct lockable obfuscation with perfect correctness from the learning with errors (LWE) assumption[GKW17, WZ17]. As a result, we get the following corollary:

Corollary 2. *Assuming LWE, fully homomorphic encryption and the existence of IND-CCA secure public key encryption, the scheme* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *described below is an IND-CCA secure public key encryption scheme that is not selective FE-Compatible even for a single function secret key query for any function family* $\mathcal{F}$ *such that* $f_1 \in \mathcal{F}$.

Notation: Let the security parameter be n. Let $(\mathsf{Setup}_{\mathsf{CPA}}, \mathsf{Enc}_{\mathsf{CPA}}, \mathsf{Dec}_{\mathsf{CPA}})$ be an IND-CPA secure encryption scheme that encrypts 1 bit messages and produces ciphertexts of length $l_1(n)$, $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ be a CCA secure encryption scheme that encrypts messages of length $(n+1+l_1(n))$ and produces ciphertexts of size $l_2(n)$. Let $\mathsf{FHE} = (\mathsf{FHE.Setup}, \mathsf{FHE.Enc}, \mathsf{FHE.DecFHE.Eval})$ be a fully homomorphic encryption scheme that encrypts messages of length $(l_1(n) + l_2(n))$ and can evaluate any $\mathsf{Poly}(n)$-sized circuit. Let $(\mathcal{O}, \mathsf{Eval})$ be a secure lockable obfuscator for all $\mathsf{Poly}(n)$-sized circuits that take inputs of size $l_2(n)$ and produce outputs of size n. Our scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ that encrypts messages of length $(n+1)$ is as follows:

- $\mathsf{PKE.Setup}(1^n)$**:**
  1. Compute $(\mathsf{PK}_{\mathsf{CPA}}, \mathsf{SK}_{\mathsf{CPA}}) \leftarrow \mathsf{Setup}_{\mathsf{CPA}}(1^n)$, $(\mathsf{PK}_{\mathsf{CCA}}, \mathsf{SK}_{\mathsf{CCA}}) \leftarrow \mathsf{Setup}_{\mathsf{CCA}}(1^n)$ and $(\mathsf{PK}_{\mathsf{FHE}}, \mathsf{SK}_{\mathsf{FHE}}) \leftarrow \mathsf{FHE.Setup}(1^n)$.
  2. Choose a random string $\alpha \in \{0,1\}^n$.
  3. Compute $\mathsf{CT}'_{\mathsf{CPA}} = \mathsf{Enc}_{\mathsf{CPA}}(\mathsf{PK}_{\mathsf{CPA}}, 0)$ and $\mathsf{CT}'_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CCA}}, \alpha||0||\mathsf{CT}'_{\mathsf{CPA}})$. Let $\mathsf{CT}' = (\mathsf{CT}'_{\mathsf{CCA}}, \mathsf{CT}'_{\mathsf{CPA}})$. (In fact, CT' is an encryption of $(\alpha||0)$ using the encryption algorithm $\mathsf{PKE.Enc}$ described next).
  4. Compute $\mathsf{CT}'_{\mathsf{FHE}} = \mathsf{FHE.Enc}(\mathsf{PK}_{\mathsf{FHE}}, \mathsf{CT}')$.
  5. Generate $\tilde{P} = \mathcal{O}(n, P, \mathsf{SK}_{\mathsf{CPA}}, \alpha)$ using the tester program P described in Figure 1 where n is the security parameter, P is the program, $\mathsf{SK}_{\mathsf{CPA}}$ is the message and α is the lock value. In particular, the functionality of the obfuscated program $\tilde{P}$ is described in Figure 2. Note that Figure 2 is just for intuition and does not correspond to a formal specification.
  6. Output the public key as $\mathsf{PK} = (\mathsf{PK}_{\mathsf{CPA}}, \mathsf{PK}_{\mathsf{CCA}}, \mathsf{PK}_{\mathsf{FHE}}, \mathsf{CT}'_{\mathsf{FHE}}, \tilde{P})$. The secret key of the scheme is $\mathsf{SK} = \mathsf{SK}_{\mathsf{CCA}}$.
- $\mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m})$**:**
  1. Given an $(n+1)$ bit message m, let p be the last bit of m.
  2. Compute $\mathsf{CT}_{\mathsf{CPA}} = \mathsf{Enc}_{\mathsf{CPA}}(\mathsf{PK}_{\mathsf{CPA}}, p)$.
  3. Compute $\mathsf{CT}_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CCA}}, \mathsf{m}||\mathsf{CT}_{\mathsf{CPA}})$.
  4. Output the ciphertext $\mathsf{CT} = (\mathsf{CT}_{\mathsf{CCA}}, \mathsf{CT}_{\mathsf{CPA}})$.
- $\mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT})$**:**
  1. Parse $\mathsf{CT} = (\mathsf{CT}_{\mathsf{CCA}}, \mathsf{CT}_{\mathsf{CPA}})$. Recall that $\mathsf{SK} = \mathsf{SK}_{\mathsf{CCA}}$.
  2. Let $(\mathsf{m}||\mathsf{y}) = \mathsf{Dec}_{\mathsf{CCA}}(\mathsf{SK}_{\mathsf{CCA}}, \mathsf{CT}_{\mathsf{CCA}})$.
  3. If the above decryption outputs $\perp$ or if $y \neq \mathsf{CT}_{\mathsf{CPA}}$, output $\perp$.
  4. Else, output the message m.

Program P

Input : FHE ciphertext $\mathsf{CT}_{\mathsf{FHE}}$
Constants : $\mathsf{SK}_{\mathsf{FHE}}$

1. Output $\mathsf{FHE.Dec}(\mathsf{SK}_{\mathsf{FHE}}, \mathsf{CT}_{\mathsf{FHE}})$.

Figure 1: Tester Program (as in lockable obfuscation notation - see Appendix B)

Program $\tilde{P}$

Input : FHE ciphertext $\mathsf{CT}_{\mathsf{FHE}}$
Constants : $\mathsf{SK}_{\mathsf{FHE}}, \alpha, \mathsf{SK}_{\mathsf{CPA}}$

1. Compute $\mathsf{y} \leftarrow \mathsf{FHE.Dec}(\mathsf{SK}_{\mathsf{FHE}}, \mathsf{CT}_{\mathsf{FHE}})$.
2. If $\mathsf{y} = \alpha$, output $\mathsf{SK}_{\mathsf{CPA}}$. Else, output $\perp$.

Figure 2: Functionality of lockable obfuscated tester program

We now prove Theorem 1.

Correctness: It can be easily observed that if the schemes $(\mathsf{Setup}_{\mathsf{CPA}}, \mathsf{Enc}_{\mathsf{CPA}}, \mathsf{Dec}_{\mathsf{CPA}})$ and $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}$ are correct except with negligible probability, then PKE is correct except with negligible probability. That is, $\mathsf{PKE.Dec}(\mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m}), \mathsf{SK}) = \mathsf{m}$ for any message $\mathsf{m} \in \{0,1\}^{(n+1)}$.

To prove our theorem we need to show two things. First, we will show that any candidate functional encryption scheme that includes a "all but last bit reveal" functionality which shares the setup and encrypt algorithms with the above public key encryption scheme must be insecure. Second, we show that the scheme PKE actually does have IND-CCA security under certain assumptions. Putting these together will yield our theorem.

5.1 An Attack

In this section, assuming the correctness of the encryption schemes used and correctness of the obfuscator, we show that the above scheme PKE is not FE-Compatible. Suppose it is indeed FE-Compatible. We will arrive at a contradiction. Formally, we prove the following lemma.

Lemma 1. *Any scheme* $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Keygen}, \mathsf{FE.Dec})$ *where* $\mathsf{FE.Setup}(.) = \mathsf{PKE.Setup}(.)$ *and* $\mathsf{FE.Enc}(.) = \mathsf{PKE.Enc}(.)$ *is not selectively secure even for just 1 function secret key query for any function family* $\mathcal{F}$ *such that* $f_1 \in \mathcal{F}$.

Proof. Consider a FE adversary $\mathcal{A}$ who interacts with a FE challenger in the selective IND-security game as follows:

1. In the first round, $\mathcal{A}$ submits two messages $\mathsf{m}_0 = (0^n||0)$ and $\mathsf{m}_1 = (0^n||1)$.

2. $\mathcal{A}$ asks for a function secret key corresponding to the following function f_1: on input x of length $(n+1)$ bits, $f_1(x)$ outputs the first n bits of x. Note that since the first n bits of m_0 and m_1 are equal, this is a valid function secret key query.

3. The challenger runs the setup algorithm and generates PK, SK. He gives PK to the adversary along with the function secret key SK_{f_1}. Also, the challenger picks a bit b at random and sends $\mathsf{CT}^* = \mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m}_b)$.

4. Let the challenge ciphertext be $\mathsf{CT}^* = (\mathsf{CT}^*_{\mathsf{CCA}}, \mathsf{CT}^*_{\mathsf{CPA}})$. The adversary computes a FHE ciphertext $\mathsf{CT}_{\mathsf{FHE}} = \mathsf{FHE.Eval}(\mathsf{FE.Dec}(\mathsf{SK}_f, \cdot), \mathsf{CT}'_{\mathsf{FHE}})$ using the ciphertext $\mathsf{CT}'_{\mathsf{FHE}}$ in the public key and the function secret key SK_f. $\mathcal{A}$ then runs the obfuscated program $\tilde{P}$ on input $\mathsf{CT}_{\mathsf{FHE}}$. That is, run $\mathsf{Eval}(\tilde{P}, \mathsf{CT}_{\mathsf{FHE}})$ to receive output $\mathsf{SK}'_{\mathsf{CPA}}$. It then computes $b' = \mathsf{Dec}_{\mathsf{CPA}}(\mathsf{SK}'_{\mathsf{CPA}}, \mathsf{CT}^*_{\mathsf{CPA}})$ and outputs b' to the challenger.

Analysis: We now show why the adversary's guess b' is equal to the challenger's random bit b except with negligible probability. From the correctness of the FE scheme, SK_{f_1} must be a correct function secret key for the function f_1. First, from the correctness of the FHE scheme, observe that $\mathsf{CT}_{\mathsf{FHE}} = \mathsf{FHE.Eval}(\mathsf{SK}_f, \mathsf{CT}'_{\mathsf{FHE}})$ is an encryption of the random string α using the algorithm $\mathsf{FHE.Enc}$. Now, notice that when this ciphertext $\mathsf{CT}_{\mathsf{FHE}}$ is a correct encryption of α. So, when it is fed as input to the program $\tilde{P}$, from the correctness of lockable obfuscation, the program outputs the secret key of the IND-CPA secure encryption scheme - $\mathsf{SK}_{\mathsf{CPA}}$ (which we denoted as $\mathsf{SK}'_{\mathsf{CPA}}$). Therefore, now the adversary's strategy easily follows. $\mathcal{A}$ uses $\mathsf{SK}_{\mathsf{CPA}}$ to decrypt $\mathsf{CT}^*_{\mathsf{CPA}}$ and from the correctness of the IND-CPA secure encryption scheme, this decrypts to give the value b correctly, which is the adversary's output.

Hence, the adversary can break the selective IND-security of the FE scheme which is a contradiction. Note that the negligible error comes from the fact that the IND-CPA secure encryption scheme, the FE scheme, the lockable obfuscation scheme and the FHE scheme are all correct except with negligible probability.

□

5.2 IND-CCA Security

We now prove that the scheme is IND-CCA secure. Our proof strategy is organized along the lines around detecting a bad query event which follows the work of Myers and Shelat[MS09] and Hohenberger, Lewko and Waters[HLW12] who proved multibit CCA security from the existence of 1-bit CCA security. Formally, we prove the following lemma:

Lemma 2. *Assuming the hardness of learning with errors (LWE),* $(\mathsf{Setup}_{\mathsf{CPA}}, \mathsf{Enc}_{\mathsf{CPA}}, \mathsf{Dec}_{\mathsf{CPA}})$ *is an IND-CPA secure public key encryption scheme and* $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ *is an IND-CCA secure public key encryption scheme,* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is an IND-CCA secure public key encryption scheme.*

Proof. We begin our proof by defining a "Bad-Query" event that is defined within the context of the attacker playing the IND-CCA security game on the encryption scheme PKE.

Definition 4. ***(Bad Query Event):*** *Let* PK *be the public key of the scheme* PKE *that is given to the adversary. We say that a bad query event has occurred during an execution of the IND-CCA security game between the adversary* $\mathcal{A}$ *and the challenger if* $\mathcal{A}$ *makes a decryption query of the form* $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ *such that* $\mathsf{CT}_1 = \mathsf{CT}'_{\mathsf{CCA}}$*, where* $\mathsf{CT}'_{\mathsf{CCA}}$ *was created by the setup algorithm* $\mathsf{PKE.Setup}$*.*

In order to prove IND-CCA security of our scheme, we will rely on the following claim :

Claim 1. *A Bad Query Event does not take place except with negligible probability in n, where the probability is taken over the coins of the adversary and the challenger playing the IND-CCA security game.*

We defer the proof of this claim to the next section. Here, we show that our scheme is IND-CCA secure assuming the claim holds true. We will prove this via a series of hybrid experiments where we show that every successive pair of hybrids is computationally indistinguishable and the final hybrid is independent of the challenge bit b and hence the attacker's advantage will be 0 in the final hybrid.

- Hyb_1**:** This is the real world experiment with challenge bit b chosen randomly. The challenge ciphertext is $\mathsf{CT}^* = (\mathsf{CT}^*_{\mathsf{CCA}}, \mathsf{CT}^*_{\mathsf{CPA}})$.
- Hyb_2**:** This hybrid is identical to the previous hybrid except that now, the decryption oracle rejects[6] for any ciphertext query $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ if $\mathsf{CT}_1 = \mathsf{CT}'_{\mathsf{CCA}}$. Note that the oracle also continues to reject the challenge ciphertext as before.
- Hyb_3**:** This hybrid is identical to the previous hybrid except that during setup, $\mathsf{CT}'_{\mathsf{CCA}}$ is now computed as $\mathsf{CT}'_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CCA}}, 0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}})$.
- Hyb_4**:** This hybrid is identical to the previous hybrid except that in the public key, $\tilde{P}$ is replaced with the simulated obfuscated program - i.e $\mathsf{Sim}(n, 1^{|P|}, 1^{|\mathsf{SK}_{\mathsf{CPA}}|})$ where Sim is the simulator of the lockable obfuscation scheme.
- Hyb_5**:** This hybrid is identical to the previous hybrid except that in the challenge ciphertext $\mathsf{CT}^* = (\mathsf{CT}^*_{\mathsf{CCA}}, \mathsf{CT}^*_{\mathsf{CPA}})$, $\mathsf{CT}^*_{\mathsf{CPA}}$ is now computed independent of the bit b as follows: $\mathsf{CT}^*_{\mathsf{CPA}} = \mathsf{Enc}_{\mathsf{CPA}}(\mathsf{PK}_{\mathsf{CPA}}, 0)$.
- Hyb_6**:** This hybrid is identical to the previous hybrid except that now, the decryption oracle also rejects any ciphertext query $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ if $\mathsf{CT}_1 = \mathsf{CT}^*_{\mathsf{CCA}}$.
- Hyb_7**:** This hybrid is identical to the previous hybrid except that in the challenge ciphertext, $\mathsf{CT}^*_{\mathsf{CCA}}$ is now computed independent of the bit b as follows: $\mathsf{CT}^*_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CPA}}(\mathsf{PK}_{\mathsf{CCA}}, 0^{n+1}||\mathsf{CT}^*_{\mathsf{CPA}})$.

Observe that in this last hybrid, the challenge ciphertext is created independent of the bit b. Hence, the attacker's advantage in this hybrid is negligible.

We will now show the indistinguishability of every successive pair of hybrids.

Claim 2. *Assuming Claim 1 holds,* Hyb_1 *is computationally indistinguishable from* Hyb_2*.*

Proof. The only difference between the two hybrids is that in Hyb_2, the decryption oracle rejects queries of the form $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ where $\mathsf{CT}_1 = \mathsf{CT}'_{\mathsf{CCA}}$ while such queries are not rejected by the oracle in Hyb_1. However, Claim 1 essentially proves that such queries (which we have defined as the occurrence of a bad query event) are never made by the adversary except with negligible probability. Therefore, if Claim 1 holds, Hyb_1 is computationally indistinguishable from Hyb_2. □

Claim 3. *Assuming that* $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ *is an IND-CCA secure encryption scheme,* Hyb_2 *is computationally indistinguishable from* Hyb_3*.*

[6] Throughout the paper, we use rejecting an input and producing output $\perp$ for the input interchangeably.

Proof. The only difference is that in Hyb_2, $\mathsf{CT}'_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CCA}}, \alpha||0||\mathsf{CT}'_{\mathsf{CPA}})$ while in Hyb_3, $\mathsf{CT}'_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CCA}}, 0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}})$. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids. We will use this adversary to come up with an adversary $\mathcal{B}$ that can break the CCA security of the encryption scheme $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$. $\mathcal{B}$ receives the public key $\mathsf{PK}_{\mathsf{CCA}}$ of the CCA secure encryption scheme from its challenger. $\mathcal{B}$ runs the algorithm $\mathsf{PKE.Setup}$ (except the $\mathsf{Setup}_{\mathsf{CCA}}$ algorithm part) to create the public key PK. $\mathcal{B}$ sends the pair $(\alpha||0||\mathsf{CT}'_{\mathsf{CPA}}, 0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}}))$ to its challenger and sets $\mathsf{CT}'_{\mathsf{CCA}}$ as the response.

It then interacts with $\mathcal{A}$ acting as the challenger of the scheme PKE and sends the public key PK to the adversary $\mathcal{A}$. Now, for every decryption oracle query, $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ made by $\mathcal{A}$, $\mathcal{B}$ forwards CT_1 to the decryption oracle provided by its challenger and receives a response $(\mathsf{m}||y)$ or $\perp$. If the response was $\perp$ or if $y \neq \mathsf{CT}_2$, $\mathcal{B}$ responds with $\perp$ to $\mathcal{A}$. Else, it sends m as the output of the decryption. $\mathcal{B}$, on receiving challenge messages $\mathsf{m}_0, \mathsf{m}_1$, picks one of them at random and encrypts that as the challenge ciphertext.

We can observe that if $\mathcal{B}$ received an encryption of $(\alpha||0||\mathsf{CT}'_{\mathsf{CPA}})$ from its challenger, that corresponds to an execution of Hyb_2 while if it received an encryption of $(0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}})$, that corresponds to an execution of Hyb_3. Thus, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{B}$ can break the CCA security of the encryption scheme $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ which is a contradiction. □

Claim 4. *Assuming that* $(\mathcal{O}, \mathsf{Eval})$ *is a secure lockable obfuscator,* Hyb_3 *is computationally indistinguishable from* Hyb_4*.*

Proof. The only difference between the two hybrids is that in Hyb_3, the public key contains $\mathcal{O}(n, P, \mathsf{SK}_{\mathsf{CPA}}, \alpha)$ while in Hyb_4, it contains the simulated program - $\mathsf{Sim}(n, 1^{|P|}, 1^{|\mathsf{SK}_{\mathsf{CPA}}|})$. Since α is picked uniformly at random and is used only as the lock value in the obfuscated program and nowhere else, from the security of lockable obfuscation, the two hybrids will be computationally indistinguishable. We now describe the reduction.

Consider an adversary $\mathcal{A}$ that can distinguish between these two hybrids. We will now design a reduction $\mathcal{A}_{\mathsf{lock}}$ that uses $\mathcal{A}$ to break the security of the lockable obfuscation scheme. $\mathcal{A}_{\mathsf{lock}}$ interacts with $\mathcal{A}$ and runs the experiment exactly as in Hyb_3 except generating the obfuscated program. $\mathcal{A}_{\mathsf{lock}}$ interacts with a challenger $\mathcal{C}$ for the lockable obfuscation scheme. $\mathcal{A}_{\mathsf{lock}}$ sends the program P and the message $\mathsf{SK}_{\mathsf{CPA}}$ to the challenger $\mathcal{C}$. $\mathcal{C}$ sends back either $\mathcal{O}(n, P, \mathsf{SK}_{\mathsf{CPA}}, \alpha)$ where α is picked uniformly at random or a simulated obfuscated circuit $\mathsf{Sim}(n, 1^{|}P|, 1^{|}\mathsf{SK}_{\mathsf{CPA}}|)$. $\mathcal{A}_{\mathsf{lock}}$ sets this as the obfuscated circuit $\tilde{P}$ and continues with the experiment as in Hyb_3. Now, it easily follows that if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{A}_{\mathsf{lock}}$ can use the same distinguishing guess to break the security of the lockable obfuscation scheme which is a contradiction.

□

Claim 5. *Assuming that* $(\mathsf{Setup}_{\mathsf{CPA}}, \mathsf{Enc}_{\mathsf{CPA}}, \mathsf{Dec}_{\mathsf{CPA}})$ *is an IND-CPA secure encryption scheme,* Hyb_4 *is computationally indistinguishable from* Hyb_5*.*

Proof. The only difference between the two hybrids is in the challenge ciphertexts. In Hyb_4, $\mathsf{CT}^*_{\mathsf{CPA}} = \mathsf{Enc}_{\mathsf{CPA}}(\mathsf{PK}_{\mathsf{CPA}}, p_b)$ while in Hyb_5, $\mathsf{CT}^*_{\mathsf{CPA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CPA}}, 0)$. Here, p is the last bit of the message m_b. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids. We will use this adversary to come up with an adversary $\mathcal{B}$ that can break the CPA security of the encryption scheme $(\mathsf{Setup}_{\mathsf{CPA}}, \mathsf{Enc}_{\mathsf{CPA}}, \mathsf{Dec}_{\mathsf{CPA}})$. $\mathcal{B}$ receives the public key $\mathsf{PK}_{\mathsf{CPA}}$ of the CCA secure encryption scheme from its challenger. $\mathcal{B}$ runs the algorithm $\mathsf{PKE.Setup}$ (except the $\mathsf{Setup}_{\mathsf{CPA}}$ algorithm part) to create the public key PK. $\mathcal{B}$ sends the public key to the adversary $\mathcal{A}$ and answers all the decryption queries honestly as in Hyb_4. $\mathcal{B}$, on receiving challenge messages $\mathsf{m}_0, \mathsf{m}_1$, picks one of them at random by choosing a bit b. $\mathcal{B}$ sends the pair $(p_b, 0)$ to its challenger

and receives $\mathsf{CT}^*_{\mathsf{CPA}}$ as an encryption of one of them. It sets $\mathsf{CT}^*_{\mathsf{CCA}}$ as in Hyb_4 and continues with the rest of the experiment.

We can observe that if $\mathcal{B}$ received an encryption of (p_b) from its challenger, that corresponds to an execution of Hyb_4 while if it received an encryption of (0), that corresponds to an execution of Hyb_5. Thus, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{B}$ can use that to break the CPA security of the encryption scheme $(\mathsf{Setup}_{\mathsf{CPA}}, \mathsf{Enc}_{\mathsf{CPA}}, \mathsf{Dec}_{\mathsf{CPA}})$ which is a contradiction.

Note that here, CPA security of the encryption scheme is enough since the decryption algorithm $\mathsf{PKE.Dec}$ does not use its secret key at all. Also, the only place the secret key $\mathsf{SK}_{\mathsf{CPA}}$ was used was inside the obfuscation of program P which is also not the case anymore in the simulated obfuscated program. □

Claim 6. Hyb_5 *is identical to* Hyb_6.

Proof. The only difference between the two hybrids is that in Hyb_6, the decryption oracle rejects any ciphertext query $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ if $\mathsf{CT}_1 = \mathsf{CT}^*_{\mathsf{CCA}}$. First, observe that if $\mathsf{CT}_2 = \mathsf{CT}^*_{\mathsf{CPA}}$, then CT is in fact the challenge ciphertext CT^* itself and hence even Hyb_6 would reject the query. On the other hand, if $\mathsf{CT}_2 \neq \mathsf{CT}^*_{\mathsf{CPA}}$ but $\mathsf{CT}_1 = \mathsf{CT}^*_{\mathsf{CCA}}$, then, $\mathsf{Dec}_{\mathsf{CCA}}(\mathsf{SK}_{\mathsf{CCA}}, \mathsf{CT}_1)$ produces (m^*, y^*) such that $y^* \neq \mathsf{CT}_2$. This is because y^* would in fact be equal to $\mathsf{CT}^*_{\mathsf{CPA}}$. Hence, even Hyb_5 would reject these queries and so the two hybrids are identical. □

Claim 7. *Assuming that* $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ *is an IND-CCA secure encryption scheme,* Hyb_6 *is computationally indistinguishable from* Hyb_7.

Proof. The only difference between the two hybrids is in the challenge ciphertext $\mathsf{CT}^* = (\mathsf{CT}^*_{\mathsf{CCA}}, \mathsf{CT}^*_{\mathsf{CPA}})$. In Hyb_6, $\mathsf{CT}^*_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CCA}}, \mathsf{m}_b || \mathsf{CT}^*_{\mathsf{CPA}})$ while in Hyb_7, $\mathsf{CT}^*_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CCA}}, 0^{n+1} || \mathsf{CT}^*_{\mathsf{CPA}})$. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between these two hybrids. We will use this adversary to come up with an adversary $\mathcal{B}$ that can break the CCA security of the encryption scheme $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$. $\mathcal{B}$ receives the public key $\mathsf{PK}_{\mathsf{CCA}}$ of the CCA secure encryption scheme from its challenger. $\mathcal{B}$ runs the algorithm $\mathsf{PKE.Setup}$ (except the $\mathsf{Setup}_{\mathsf{CCA}}$ algorithm part) to create the public key PK. It then interacts with $\mathcal{A}$ acting as the challenger of the scheme PKE and sends the public key PK to the adversary $\mathcal{A}$. Now, for every decryption oracle query, $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ made by cA, $\mathcal{B}$ forwards CT_1 to the decryption oracle provided by its challenger and receives a response $(\mathsf{m}||y)$ or $\perp$. If the response was $\perp$ or if $y \neq \mathsf{CT}_2$, $\mathcal{B}$ responds with $\perp$ to $\mathcal{A}$. Else, it sends m as the output of the decryption. $\mathcal{B}$, on receiving challenge messages $\mathsf{m}_0, \mathsf{m}_1$, picks one of them at random by choosing a bit b. Compute $\mathsf{CT}^*_{\mathsf{CPA}} = \mathsf{Enc}_{\mathsf{CPA}}(\mathsf{PK}_{\mathsf{CPA}}, 0)$. $\mathcal{B}$ sends the pair $(\mathsf{m}_b || \mathsf{CT}^*_{\mathsf{CPA}}, 0^{n+1} || \mathsf{CT}^*_{\mathsf{CPA}}))$ to its challenger and sets $\mathsf{CT}^*_{\mathsf{CCA}}$ as the response. $\mathcal{B}$ sends the challenge ciphertext $\mathsf{CT}^* = (\mathsf{CT}^*_{\mathsf{CCA}}, \mathsf{CT}^*_{\mathsf{CPA}})$.

We can observe that if $\mathcal{B}$ received an encryption of $(\mathsf{m}_b || \mathsf{CT}^*_{\mathsf{CPA}})$ from its challenger, that corresponds to an execution of Hyb_6 while if it received an encryption of $(0^{n+1} || \mathsf{CT}^*_{\mathsf{CPA}})$, that corresponds to an execution of Hyb_7. Thus, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{B}$ can use that to break the CCA security of the encryption scheme $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ which is a contradiction. □

5.3 Proof of Claim 1

Instead of proving the claim directly, we first prove it for an alternate IND-CCA security game and then show how it holds even in the actual IND-CCA security game.

Alternate IND-CCA Game. This is same as the original game except that the Challenger now computes $\mathsf{CT}'_{\mathsf{CCA}}$ during setup as follows: $\mathsf{CT}'_{\mathsf{CCA}} = \mathsf{Enc}_{\mathsf{CCA}}(\mathsf{PK}_{\mathsf{CCA}}, 0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}})$. That is, α is no longer part of the message being encrypted. For this alternate IND-CCA game, the Bad Query Event remains the same: i.e, the event occurs if the adversary makes a query $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ to the decryption oracle where $\mathsf{CT}_1 = \mathsf{CT}'_{\mathsf{CCA}}$. Now, via a sequence of hybrids, we show that Claim 1 holds for this alternate IND-CCA game. That is, we show that the Bad Query Event happens with negligible probability.

- Hyb_1**:** This hybrid corresponds to the alternate IND-CCA game as described above.
- Hyb_2**:** This hybrid is identical to the previous hybrid except that in the public key, $\tilde{P}$ is replaced with the simulated obfuscated program - i.e $\mathsf{Sim}(n, 1^{|P|}, 1^{|\mathsf{SK}_{\mathsf{CPA}}|})$ where Sim is the simulator of the lockable obfuscation scheme.
- Hyb_3**:** This hybrid is identical to the previous hybrid except that the ciphertext $\mathsf{CT}'_{\mathsf{FHE}}$ is now computed as $\mathsf{CT}'_{\mathsf{FHE}} = \mathsf{FHE.Enc}(\mathsf{PK}_{\mathsf{FHE}}, 0^{l_1(n)+l_2(n)})$.

We now show that every successive pair of hybrids is computationally indistinguishable. This proves that the probability that the Bad Query Event occurs is the same for every pair of successive hybrids. Finally, we show that in the last hybrid Hyb_4, the probability that the Bad Query Event occurs is negligible.

Claim 8. *Assuming that* $(\mathcal{O}, \mathsf{Eval})$ *is a secure lockable obfuscator,* Hyb_1 *is computationally indistinguishable from* Hyb_2*.*

Proof. The proof is same as the proof of Claim 5. □

Claim 9. *Assuming that* $(\mathsf{FHE}.setup, \mathsf{FHE}.enc, \mathsf{FHE}.dec)$ *is an IND-CPA secure fully homomorphic encryption scheme,* Hyb_2 *is computationally indistinguishable from* Hyb_3*.*

Proof. The proof is same as the proof of Claim 6. □

Claim 10. *Pr[Bad Query Event occurs in* Hyb_3*] = negligible(n).*

Proof. This is because the ciphertext $\mathsf{CT}'_{\mathsf{CCA}}$ does not appear at all in the public key anymore! Even if the adversary knew the value of $(\mathsf{CT}'_{\mathsf{CPA}})$, the only information that the adversary has about $\mathsf{CT}'_{\mathsf{CCA}}$ is that it is an encryption of $(0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}})$ using public key $\mathsf{PK}_{\mathsf{CCA}}$.

First, observe that the number of possible ciphertexts for the message $(0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}})$ must be at least super-polynomial in n. This follows from the CPA security of the encryption scheme $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ because if this wasn't true, a polynomial time adversary can break the CPA security by generating all possible ciphertexts for $(0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}})$ and testing it with the challenge ciphertext.

Now, notice that to make the Bad Query Event occur, the adversary will just have to guess the value of $\mathsf{CT}'_{\mathsf{CCA}}$ (or the randomness that was used in the encryption to generate $\mathsf{CT}'_{\mathsf{CCA}}$) and this can be done only with negligible probability. □

Original IND-CCA Game. We show that the Bad Query Event happens only with negligible probability even in the original IND-CCA game. Formally, we prove the following lemma:

Lemma 3. *Assuming* $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$ *is a CCA secure encryption scheme and that the Bad Query Event does not occur in the Alternate CCA game described above except with negligible probability, the Bad Query Event does not occur in the original CCA security game for the encryption scheme* PKE *except with negligible probability.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that makes the Bad Query Event occur with non-negligible probability. We now construct an algorithm $\mathcal{B}$ that breaks the IND-CCA security of $(\mathsf{Setup}_{\mathsf{CCA}}, \mathsf{Enc}_{\mathsf{CCA}}, \mathsf{Dec}_{\mathsf{CCA}})$. $\mathcal{B}$ acts as the challenger of the IND-CCA security game for the scheme PKE in its interaction with $\mathcal{A}$. First, $\mathcal{B}$ interacts with its challenger and receives the public key $\mathsf{PK}_{\mathsf{CCA}}$. $\mathcal{B}$ then runs the setup algorithm $\mathsf{PKE.Setup}$,(except the $\mathsf{Setup}_{\mathsf{CCA}}$ part) to compute the public keys $\mathsf{PK}_{\mathsf{CPA}}, \mathsf{PK}_{\mathsf{FHE}}$. It computes $\mathsf{CT}'_{\mathsf{CPA}}$ as done by the setup algorithm. $\mathcal{B}$ then sends the pair $(\alpha||0||\mathsf{CT}'_{\mathsf{CPA}}, 0^{n+1}||\mathsf{CT}'_{\mathsf{CPA}})$ as the two challenge messages to the challenger and sets the response as $\mathsf{CT}'_{\mathsf{CCA}}$. $\mathcal{B}$ continues with the rest of the game acting as the challenger to $\mathcal{A}$. Whenever $\mathcal{A}$ makes a decryption query $(\mathsf{CT}_1, \mathsf{CT}_2)$, if $\mathsf{CT}_1 \neq \mathsf{CT}'_{\mathsf{CCA}}$, it queries the decryption oracle of its challenger with CT_1 and uses this to respond to $\mathcal{A}$ as done in the original game. Similarly, $\mathcal{B}$ also creates the challenge ciphertext. If $\mathcal{B}$ ever receives a query $(\mathsf{CT}_1, \mathsf{CT}_2)$ from $\mathcal{A}$ to the decryption oracle such that $\mathsf{CT}_1 = \mathsf{CT}'_{\mathsf{CCA}}$, it immediately halts the game with $\mathcal{A}$ and outputs the guess 0 to its challenger. If such a query never happens, it outputs 1 to the challenger after completing the game with $\mathcal{A}$.

We now analyze why this works. The algorithm $\mathcal{B}$ knows that if its challenger gave an encryption of 0^{n+1}, then its interaction with $\mathcal{A}$ corresponds to the alternate IND-CCA game described earlier. Here, we know that the adversary $\mathcal{A}$ can not make the Bad Query Event occur. Therefore, if the adversary $\mathcal{A}$ makes the Bad Query Event occur, then it must occur in the case that $\mathsf{CT}'_{\mathsf{CCA}}$ is an encryption of $(\alpha||0||\mathsf{CT}'_{\mathsf{CPA}})$. Hence, $\mathcal{B}$ guesses 0 in that case. On the other hand, if the adversary $\mathcal{A}$ does not make the Bad Query Event occur, then it must be the case that 0^{n+1} was encrypted. This is because we assumed that $\mathcal{A}$ can make the Bad Query Event occur with non-negligible probability in the original IND-CCA security game. This completes the proof.

Note that the reduction is actually not interested in completing the game with $\mathcal{A}$ in the event that $\mathcal{B}$ halts. That is, $\mathcal{B}$ does not care whether $\mathcal{A}$ wins the IND-CCA game but is rather more interested in whether $\mathcal{A}$ makes a Bad Query Event occur. □

Remark: At first glance, there seems to be a circularity issue in trying to prove IND-CCA security of our scheme. That is, in order to prove indistinguishability of the main hybrids, we require to first erase α which depends on no queries being made to the decryption oracle that contain $\mathsf{CT}'_{\mathsf{CCA}}$. On the other hand, it seems difficult to directly argue that no such queries are made because of the presence of α in $\mathsf{CT}'_{\mathsf{CCA}}$. This causes a circularity. We get around this issue using the alternate IND-CCA game where α is erased. In this game, we show that the bad query event can't occur and then using a reduction to the underlying encryption scheme's security, we can eventually show that the bad query event does not occur even in the original CCA security game.

This technique is very similar to [MS09, HLW12]. In these works, they construct CCA secure encryption and in the process, they run into a similar circularity issue. The analog of α was the randomness used for encryption and this randomness is in fact encrypted by an inner encryption scheme.

This completes the proof of Theorem 1. □

6 On the Difficulty of Proving Functional Encryption Compatibility for Fujisaki-Okamoto and RSA-OAEP

In exploring functional encryption compatibility an important class of schemes to consider is schemes that have proofs of CCA security based on the random oracle heuristic. Examples of these include RSA-OAEP[BR94] and the Fujisaki-Okamoto [FO99] transformation. While standard model proofs might be desirable, studying random oracle-based schemes is important as they largely represent what encryption is deployed and standardized. For example, RSA-OAEP is standardized as PKCS #1 v2.2 [JK03].

In this section we give a negative result and show that any functional encryption system that uses the setup and encryption algorithms from the Fujisaki-Okamoto transformation is insecure *in the random oracle model*. We believe a very similar analysis will work for the RSA-OAEP scheme, but chose to focus our formal analysis on Fujisaki-Okamoto.

However, this finding must be taken with a grain of salt as it might the case that such a candidate FE system is actually secure when the random oracles are instantiated with a concrete hash function such as SHA-256. The above statement is quite peculiar in that almost all random oracle uninstantiability results [CGH04] are of the flavor that one shows a scheme that is secure in the random oracle model, but insecure when one uses any concrete instantiation of it. Our result is somewhat the opposite where the scheme in question is insecure when using the random oracle, but may or may not be secure for a particular hash function.

Our takeaway interpretation is that if one attempted to derive a functional encryption scheme from the Fujisaki-Okamoto transformation (or a similar scheme) it might be secure, but it would be difficult to arrive at a proof of security. The reason is that any argument for FE compatibility would seem to need to be rooted in the encryption scheme being CCA secure to begin with. However, the only way we currently know how to argue security of such systems is via the random oracle heuristic and this pathway is cutoff by our result.

6.1 Construction

We first describe the Fujisaki-Okamoto transformation that combines any public key encryption scheme and any private key encryption scheme into a CCA secure encryption scheme in the random oracle model. Then, we show why such a CCA secure encryption scheme can not be FE-Compatible.

Notation: Let n be the security parameter. Let $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be any public key encryption scheme with message space $\{0,1\}^{p(n)}$ and randomness space $\{0,1\}^{l(n)}$ for some polynomials p, l. Let $(\mathsf{SKE.Enc}, \mathsf{SKE.Dec})$ be a private key encryption scheme with message space $\{0,1\}^{q(n)}$ and key space $\{0,1\}^{k(n)}$ for some polynomials q, k. Let $\mathsf{G} : \{0,1\}^* \to \{0,1\}^{k(n)}$ and $\mathsf{H} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{l(n)}$ be two random oracles.

The CCA secure encryption scheme $\mathsf{FO} = (\mathsf{FO.Setup}, \mathsf{FO.Enc}, \mathsf{FO.Dec})$ on applying the Fujisaki-Okamoto transformation has message space $\{0,1\}^{q(n)}$. The scheme is as follows:

- $\mathsf{FO.Setup}(1^n)$**:**
  1. Compute $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{PKE.Setup}(1^n)$.
  2. Set PK as the public key and SK as the secret key of the scheme.
- $\mathsf{FO.Enc}(\mathsf{PK}, \mathsf{m})$**:**
  1. Choose $\mathsf{r} \in \{0,1\}^{p(n)}$ uniformly at random. Compute $\mathsf{a} = \mathsf{G}(\mathsf{r})$.

2. Compute $\mathsf{CT}_1 = \mathsf{SKE.Enc}(\mathsf{a}, \mathsf{m})$.
3. Compute $\mathsf{h} = \mathsf{H}(\mathsf{r}, \mathsf{CT}_1)$.
4. Compute $\mathsf{CT}_2 = \mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{r}; \mathsf{h})$ using randomness h.
5. Output $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$.

- $\mathsf{FO.Dec}(\mathsf{SK}, \mathsf{CT})$**:**

1. Parse $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$.
2. Compute $\tilde{\mathsf{r}} = \mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT}_2)$.
3. Compute $\tilde{\mathsf{a}} = \mathsf{G}(\tilde{\mathsf{r}})$ and $\tilde{\mathsf{h}} = \mathsf{H}(\tilde{\mathsf{r}}, \mathsf{CT}_1)$.
4. Check that $\mathsf{CT}_2 = \mathsf{PKE.Enc}(\mathsf{PK}, \tilde{\mathsf{r}}; \tilde{\mathsf{h}})$. Else, Abort.
5. Output $\mathsf{m} = \mathsf{SKE.Dec}(\tilde{\mathsf{a}}, \mathsf{CT}_1)$.

6.2 Attack

Consider the scheme FO as described above with message space consisting of all strings of length $(n+1)$. That is, let $q(n) = n+1$. Assuming the correctness and security of the underlying encryption schemes PKE and SKE, we will show that the above scheme FO is not FE-Compatible in the random oracle model. We will use the same function f_1 as in the last section. That is, $f_1(\mathsf{m})$ outputs the first n bits of the $(n+1)$ bit string m. Formally, we prove the following lemma.

Lemma 4. *In the random oracle model, assuming that the underlying encryption scheme* SKE *is semantically secure, any scheme* $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Keygen}, \mathsf{FE.Dec})$ *where* $\mathsf{FE.Setup}(.) = \mathsf{FO.Setup}(.)$ *and* $\mathsf{FE.Enc}(.) = \mathsf{FO.Enc}(.)$ *is not a selectively secure functional encryption scheme even for just 1 function secret key query for any function family* $\mathcal{F}$ *such that* $f_1 \in \mathcal{F}$.

Proof. Consider any ciphertext $\mathsf{CT} = \mathsf{FO.Enc}(\mathsf{PK}, \mathsf{m}; \mathsf{r})$. That is, the ciphertext was generated using randomness r. Recall that, given ciphertext $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$, the algorithm $\mathsf{FO.Dec}$ first decrypts CT_2 to generate r and then queries the two random oracles G and H on this randomness r. We will now consider two cases and show that the scheme FE is not a secure FE scheme in both.

Case 1: In this case, let's assume that, given any ciphertext $\mathsf{CT} = \mathsf{FO.Enc}(\mathsf{PK}, \mathsf{m}; \mathsf{r})$, the $\mathsf{FE.Dec}$ algorithm, except with negligible probability, either queries the random oracle G on the randomness r or queries the random oracle H on (r, c) for some c.

Now, consider a FE adversary $\mathcal{A}$ who interacts with a FE challenger in the selective IND-security game as follows:

1. In the first round, $\mathcal{A}$ submits two messages $\mathsf{m}_0^* = (0^n||0)$ and $\mathsf{m}_1^* = (0^n||1)$.
2. $\mathcal{A}$ asks for a function secret key corresponding to the function f_1. Note that since the first n bits of m_0 and m_1 are equal, this is a valid function secret key query.
3. The challenger runs the setup algorithm and generates PK, SK. He gives PK to the adversary along with the function secret key SK_{f_1}. Also, the challenger picks a bit b at random and sends $\mathsf{CT}^* = \mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m}_b^*; \mathsf{r}^*)$.

4. $\mathcal{A}$ computes $\mathsf{FE.Dec}(\mathsf{SK}_{f_1}, \mathsf{CT}^*)$. Recall that this algorithm, except with negligible probability, would either query G on the randomness r^* or query H on (r, c) for some c. Let's say the algorithm makes q queries to both oracles combined. $\mathcal{A}$ observes all these queries and for each of them, tries to check whether it was the randomness r^*. That is, it can re-encrypt both m_0^* and m_1^* using the algorithm $\mathsf{FO.Enc}$ and check if it generates CT^*. Therefore, the adversary can easily determine whether CT^* encrypts m_0^* or m_1^* and this would break FE security.

Case 2: Now, we assume that given any ciphertext $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ that is computed as $\mathsf{FO.Enc}(\mathsf{PK}, \mathsf{m}; \mathsf{r})$, the $\mathsf{FE.Dec}$ algorithm, except with negligible probability, neither queries the random oracle G on the randomness r nor queries the random oracle H on (r, c) for some string c.

We now describe why it is impossible in this case too. In any ciphertext $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2)$ computed as $\mathsf{FO.Enc}(\mathsf{PK}, \mathsf{m}; \mathsf{r})$, notice that $\mathsf{CT}_1 = \mathsf{SKE.Enc}(\mathsf{a}, \mathsf{m})$ where $\mathsf{a} = \mathsf{G}(\mathsf{r})$. That is, the secret key used for the private key encryption scheme is the output of the random oracle G on the randomness r and this is the only component of the ciphertext that uses the underlying message m. However, if the decryption algorithm $\mathsf{FE.Dec}$ never queries the random oracle G on r, then by the security of the random oracle, the output $\mathsf{G}(\mathsf{r})$ is statistically indistinguishable from random. Therefore, the algorithm $\mathsf{FE.Dec}$ has no information about the secret key a. Yet, by the correctness of the FE scheme, we know that $\mathsf{FE.Dec}$ is able to partially decrypt the underlying message inside the ciphertext $\mathsf{CT}_1 = \mathsf{SKE.Enc}(\mathsf{a}, \mathsf{m})$ and evaluate the function output. Therefore, we can use this to break the semantic security of the private key encryption scheme which will be a contradiction.

Thus, in this case, the resulting functional encryption scheme can not be correct. Note that, in fact, in this case, we don't even need to use the second oracle H in the proof.

Further, recall that the decryption algorithm for any functional encryption scheme is deterministic (we are only considering deterministic functionalities). Hence, these 2 cases are exhaustive because given any ciphertext, the decryption algorithm either queries one of the random oracles or it doesn't. This completes the proof. □

7 Building FE-Compatible Encryption Schemes

We first define a new notion called puncturable tag based encryption[7]. In the next subsection, we show how to construct a selective IND-CCA secure public key encryption scheme from any puncturable tag based encryption scheme. We call such a selective IND-CCA secure public key encryption scheme as "Special-CCA". In the following subsection, we show how to instantiate a "Special-CCA" secure encryption scheme with several existing popular encryption schemes in literature. Finally, we show that this "Special-CCA" secure public key encryption scheme is FE-Compatible.

7.1 Puncturable Tag Based Encryption

In this section, we define a new primitive called puncturable tag based encryption (PTBE) that is a modification of tag based encryption schemes [Kil06] but with two more algorithms. We then show how several well known encryption schemes in literature (based on various assumptions) do in fact fit into the framework of puncturable tag based encryption.

Let n denote the security parameter and $\mathcal{X} = \{\mathcal{X}_n\}_{n\in\mathbf{N}}$, $\mathcal{T} = \{\mathcal{T}_n\}_{n\in\mathbf{N}}$ denote ensembles where each $\mathcal{X}_n$ and $\mathcal{T}_n$ is a finite set. Formally, a puncturable tag based encryption scheme PTBE =

[7] Previously, [MH14] also introduced a primitive called puncturable tag based encryption which is completely different from the one we define here.

$(\mathsf{PTBE.Setup}, \mathsf{PTBE.Enc}, \mathsf{PTBE.Dec}, \mathsf{PTBE.Setup\text{-}Alt}, \mathsf{PTBE.Setup\text{-}Alt\text{-}1}, \mathsf{PTBE.Dec\text{-}Alt})$ consists of the following algorithms:

- $\mathsf{PTBE.Setup}(1^n)$:
  Given the security parameter n, it generates a public key PK and a secret key SK.
- $\mathsf{PTBE.Enc}(\mathsf{PK}, \mathsf{t}, \mathsf{m})$:
  Given a message $\mathsf{m} \in \mathcal{X}_n$, a tag $\mathsf{t} \in \mathcal{T}_n$ and the public key PK as input, the encryption algorithm outputs a ciphertext CT.
- $\mathsf{PTBE.Dec}(\mathsf{SK}, \mathsf{t}, \mathsf{CT})$:
  Given a ciphertext CT, a tag $\mathsf{t} \in \mathcal{T}_n$ and the secret key SK as input, the decryption algorithm outputs a string $y \in \mathcal{X}_n$ or $\perp$.
- $\mathsf{PTBE.Setup\text{-}Alt}(1^n, \mathsf{t}^*, \mathsf{m}^*)$:
  Given the security parameter n, a tag $\mathsf{t}^* \in \mathcal{T}_n$ and a message $\mathsf{m}^* \in \mathcal{X}_n$, it generates a public key PK, a secret key SK, an alternate secret key $\mathsf{SK\text{-}Alt}$ and a ciphertext CT^*.
- $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}(1^n, \mathsf{t}^*, \mathsf{m}^*)$:
  Given the security parameter n, a tag $\mathsf{t}^* \in \mathcal{T}_n$ and a message $\mathsf{m}^* \in \mathcal{X}_n$, it generates a public key PK, a secret key SK, an alternate secret key $\mathsf{SK\text{-}Alt}$ and a ciphertext CT^*.
- $\mathsf{PTBE.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{t}, \mathsf{CT})$:
  Given a ciphertext CT, a tag $\mathsf{t} \in \mathcal{T}_n$ and an alternate secret key $\mathsf{SK\text{-}Alt}$ as input, the alternate decryption algorithm outputs a string $y \in \mathcal{X}_n$ or $\perp$.

Remark: For technical reasons, to make our proofs simpler while instantiating our "Special-CCA" secure encryption schemes, we use two setup-alt algorithms (that albeit perform a very similar role). We provide more details about this in a remark at the end of Section D.2. Alternatively, we could just use one setup-alt algorithm in the abstraction and make the proof a bit more complicated. We choose the former option in this writeup.

Correctness: A puncturable tag based encryption scheme PTBE is correct if for all messages $\mathsf{m} \in \mathcal{X}_n$ and all tags $\mathsf{t} \in \mathcal{T}_n$

$$\Pr\left[\begin{array}{c}(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{PTBE.Setup}(1^n) \\ \mathsf{PTBE.Dec}(\mathsf{SK}, \mathsf{t}, \mathsf{PTBE.Enc}(\mathsf{PK}, \mathsf{t}, \mathsf{m})) = \mathsf{m}\end{array}\right] = 1$$

The probability is over the randomness used in the setup, encryption and decryption algorithms.

For security, we require the following four properties:

1. **Equivalent on all but challenge tag:** For any message $\mathsf{m}^* \in \mathcal{X}_n$, any tag $\mathsf{t}^* \in \mathcal{T}_n$, for all ciphertexts CT and all tags $\mathsf{t} \in \mathcal{T}_n$ such that $\mathsf{t} \neq \mathsf{t}^*$, we require that:

$$\Pr\left[\begin{array}{c}(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*) \leftarrow \mathsf{PTBE.Setup\text{-}Alt}(1^n, \mathsf{t}^*, \mathsf{m}^*) \\ \mathsf{PTBE.Dec}(\mathsf{SK}, \mathsf{t}, \mathsf{CT}) = \mathsf{PTBE.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{t}, \mathsf{CT})\end{array}\right] = 1$$

   The probability is over the randomness used in all the above algorithms.

2. **Indistinguishability of parameters:** The output of the following two experiments must be computationally indistinguishable for all messages m^* and tags t^*:

(a) **Experiment 1:**
Run $\mathsf{PTBE.Setup}(1^n)$ to generate $(\mathsf{PK}, \mathsf{SK})$. Compute $\mathsf{CT}^* = \mathsf{PTBE.Enc}(\mathsf{PK}, \mathsf{t}^*, \mathsf{m}^*)$ and output $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}^*)$.

(b) **Experiment 2:**
Run $\mathsf{PTBE.Setup\text{-}Alt}(1^n, \mathsf{t}^*, \mathsf{m}^*)$ to generate $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$ and output $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}^*)$.

3. **Indistinguishability of alternate setups:** The output of the following two experiments must be indistinguishable for all messages m^* and tags t^*:

   (a) **Experiment 1:**
   Run $\mathsf{PTBE.Setup\text{-}Alt}(1^n, \mathsf{t}^*, \mathsf{m}^*)$ to generate $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$ and output $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$.

   (b) **Experiment 2:**
   Run $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}(1^n, \mathsf{t}^*, \mathsf{m}^*)$ to generate $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$ and output $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$.

4. **Indistinguishability of messages:** For this property to hold, we require the adversary's advantage to be negligible in the following game between an adversary $\mathcal{A}$ and a challenger Ch:

   (a) $\mathcal{A}$ sends $(\mathsf{t}^*, \mathsf{m}_0^*, \mathsf{m}_1^*)$ to the challenger.

   (b) Ch chooses a random bit b and runs $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}(1^n, \mathsf{t}^*, \mathsf{m}_b^*)$ to generate $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$ and gives the adversary $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$.

   (c) $\mathcal{A}$ submits a guess b' and wins if $b' = b$. The adversary's advantage in this game is defined to be $2 * |\mathsf{Pr}[\mathsf{b} = \mathsf{b}'] - 1/2|$.

7.2 Special-CCA secure encryption scheme

In this section, we show how to build a selective CCA secure encryption scheme from any PTBE with the addition of one time signatures. Recall that we define selective CCA secure encryption schemes in Appendix A. We call such a CCA secure encryption scheme as "Special-CCA". Formally, we prove the following theorem:

Theorem 3. *Given a puncturable tag based encryption scheme* $\mathsf{PTBE} = (\mathsf{PTBE.Setup}, \mathsf{PTBE.Enc}, \mathsf{PTBE.Dec}, \mathsf{PTBE.Setup\text{-}Alt}, \mathsf{PTBE.Setup\text{-}Alt\text{-}1}, \mathsf{PTBE.Dec\text{-}Alt})$ *and a strongly secure one time signature scheme* $\mathsf{OTS} = (\mathsf{OTS.Setup}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$*, the scheme* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.D}$ *described below is a selective CCA secure encryption scheme.*

According to our notation, scheme PKE is a Special-CCA secure encryption scheme.

Notation: Let $\mathsf{PTBE} = (\mathsf{PTBE.Setup}, \mathsf{PTBE.Enc}, \mathsf{PTBE.Dec}, \mathsf{PTBE.Setup\text{-}Alt}, \mathsf{PTBE.Setup\text{-}Alt\text{-}1}, \mathsf{PTBE.Dec\text{-}Alt})$ be a puncturable tag based encryption scheme with message space $\mathcal{X}_n$, tag space $\mathcal{T}_n$ that outputs ciphertexts of size $l(n)$. Let $\mathsf{OTS} = (\mathsf{OTS.Setup}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$ be a one time signature scheme that signs messages of length $l(n)$ and the space of verification keys is $\mathcal{T}_n$. Our new scheme PKE has message space $\mathcal{X}_n$.

We now describe the template for building Special-CCA secure encryption schemes from any puncturable tag based encryption. This template can be instantiated by several existing CCA secure encryption schemes in the literature [CHK04, Kil06, PW08].

Construction:

- $\mathsf{PKE.Setup}(1^n)$:
  1. Generate the public key and secret key as $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{PTBE.Setup}(1^n)$.
- $\mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m})$:
  1. Generate $(\mathsf{VK}, \mathsf{SigK}) \leftarrow \mathsf{OTS.Setup}(1^n)$.
  2. Compute $\mathsf{CT}_1 = \mathsf{PTBE.Enc}(\mathsf{PK}, \mathsf{VK}, \mathsf{m})$ and $\sigma = \mathsf{OTS.Sign}(\mathsf{CT}_1, \mathsf{SigK})$.
  3. Output $\mathsf{CT} = (\mathsf{VK}, \mathsf{CT}_1, \sigma)$ as the ciphertext.
- $\mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT})$:
  1. Parse $\mathsf{CT} = (\mathsf{VK}, \mathsf{CT}_1, \sigma)$.
  2. Output $\perp$ if $\mathsf{OTS.Verify}(\mathsf{VK}, \mathsf{CT}_1, \sigma) = 0$.
  3. Output $\mathsf{m} = \mathsf{PTBE.Dec}(\mathsf{SK}, \mathsf{VK}, \mathsf{CT}_1)$.

We prove that the above scheme is CCA-secure in *Appendix C*.

7.3 Instantiating Special-CCA encryption

We show that several popular and well-studied CCA-secure encryption schemes are in fact Special-CCA. That is, they satisfy this property that they can be constructed using PTBE and one-time signatures as shown in the above construction. We now list the encryption schemes below and prove in Appendix D that they satisfy the necessary conditions. Formally,

Theorem 4. *The selective CCA-secure encryption schemes in the following popular works are in fact Special-CCA secure encryption schemes:*

- [*CHK04*] *when instantiated with the IBE scheme of* [*BB04*].
- [*CHK04*] *when instantiated with any Hierarchical IBE scheme.*
- [*PW08*] *when instantiated with any Lossy Trapdoor Function.*

7.4 Building selectively secure FE

In this section, we show that the "Special-CCA" secure encryption scheme $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ from the previous section is FE-Compatible. We prove the security of our construction in two different ways - the first is based on the assumption of sub-exponentially secure indistinguishability obfuscation. Additionally, it requires the one time signature scheme used in the construction of PKE to be a sub-exponentially secure unique signature scheme. On the other hand, the second proof is based on the existence of polynomially secure differing inputs obfuscation and just polynomially secure one time signatures.

Formally, we prove the following two theorems:

Theorem 5. *Any "Special-CCA" secure encryption scheme is selective FE-Compatible for any function family $\mathcal{F}_n$ and poly(n) function key queries assuming:*

- *Sub-exponentially secure indistinguishability obfuscation. (AND)*

- *Sub-exponentially secure unique one time signatures.*

Moreover, the resulting FE scheme is also compact.

Theorem 6. *Any "Special-CCA" secure encryption scheme is selective FE-Compatible for any function family $\mathcal{F}_n$ and poly(n) function key queries assuming:*

- *Polynomially secure differing inputs obfuscation. (AND)*
- *Polynomially secure strong one time signatures.*

Moreover, the resulting FE scheme is also compact.

One example of a one time signature scheme is the Lamport signature scheme[Lam79]. Observe that it is in fact a unique one time signature scheme if we rely on injective one way functions. Instantiating the Special-CCA scheme with the various schemes in Section 7.3, we get the following two corollaries:

Corollary 7. *Let* X *denote the CCA secure encryption scheme in any of the following popular works :*

- [*CHK04*] *when instantiated with the IBE scheme of* [*BB04*].
- [*CHK04*] *when instantiated with any Hierarchical IBE scheme.*
- [*PW08*] *when instantiated with any Lossy Trapdoor Function.*

Assuming sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure injective one way functions, scheme X *is selective FE-Compatible for any function family $\mathcal{F}_n$ and poly(n) function key queries. Moreover, the resulting FE scheme is also compact.*

Corollary 8. *Let* X *denote the CCA secure encryption scheme in any of the following popular works :*

- [*CHK04*] *when instantiated with the IBE scheme of* [*BB04*].
- [*CHK04*] *when instantiated with any Hierarchical IBE scheme.*
- [*PW08*] *when instantiated with any Lossy Trapdoor Function.*

Assuming polynomially secure differing inputs obfuscation and polynomially secure one way functions, scheme X *is selective FE-Compatible for any function family $\mathcal{F}_n$ and poly(n) function key queries. Moreover, the resulting FE scheme is also compact.*

Construction: Let $(\mathcal{O}, \mathsf{Eval})$ be a secure obfuscator (note that we will use indistinguishability obfuscation in one proof and differing inputs obfuscation in the other). The functional encryption $\mathsf{FE} = (\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Keygen}, \mathsf{FE.Dec})$ built from the Special-CCA scheme PKE is as follows. Recall that from the definition of FE-Compatibility, $\mathsf{FE.Setup}(\cdot) = \mathsf{PKE.Setup}(\cdot)$ and $\mathsf{FE.Enc}(\cdot) = \mathsf{PKE.Enc}(\cdot)$.

- $\mathsf{FE.Setup}(1^n)$: Run $\mathsf{PKE.Setup}(1^n)$ to generate $(\mathsf{PK}, \mathsf{SK})$.
- $\mathsf{FE.Enc}(\mathsf{PK}, \mathsf{m})$: Run $\mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m})$ to generate the ciphertext $\mathsf{CT} = (\mathsf{VK}, \mathsf{CT}_1, \sigma)$.
- $\mathsf{FE.Keygen}(\mathsf{SK}, f)$: Output $\mathsf{SK}_f = \mathcal{O}(\mathsf{G}_f)$ where the program G_f is described below.

- $\mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{CT})$ Run the program SK_f on input CT to output a string y.

Program G_f

Input : ciphertext CT
Constants : SK

1. Compute $\mathsf{m} = \mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT})$.
2. Output $\perp$ if the decryption aborts.
3. Else, output $f(m)$.

Figure 3: Program for generating function secret key

7.4.1 Security Proof

We will prove this via a series of hybrid experiments where we show that every successive pair of hybrids is computationally indistinguishable and the final hybrid is independent of the challenge bit b and hence the attacker's advantage will be 0 in the final hybrid. We will show the indistinguishability of the hybrids using two different proofs in some cases to prove both Theorem 5 and Theorem 6.

- Hyb_1**:** This is the real world experiment with challenge bit b chosen randomly. The challenge ciphertext as $\mathsf{CT}^* = (\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$.
- Hyb_2**:** This hybrid is identical to the previous hybrid except that now, $\mathsf{FE.Setup}(1^n)$ and the challenge ciphertext are computed differently. Instead of running the setup algorithm $\mathsf{PTBE.Setup}(1^n)$, we now run $\mathsf{PTBE.Setup\text{-}Alt}(1^n, \mathsf{VK}^*, \mathsf{m}_b^*)$ to generate $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$. The FE scheme's public key is PK, secret key is SK. Now, the challenge ciphertext is computed as follows: generate $(\mathsf{SigK}^*, \mathsf{VK}^*) \leftarrow \mathsf{OTS.Setup}(1^n)$ and compute $\sigma^* = \mathsf{OTS.Sign}(\mathsf{SigK}^*, \mathsf{CT}_1^*)$. The challenge ciphertext is $(\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$. Note that the alternate secret key $\mathsf{SK\text{-}Alt}$ is not used at all.
- **For each** i **in** $\{0, 1, \ldots, q\}$**,** $\mathsf{Hyb}_{3,i}$**:** This hybrid is identical to the previous hybrid except that now, the function secret key SK_f for the i^{th} function key query f is computed as $\mathcal{O}(\mathsf{G}_f^1)$ for the following program G_f^1.

Program G_f^1

Input : ciphertext CT
Constants : $\mathsf{SK}, \mathsf{CT}^* = (\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$

1. If $\mathsf{CT} = \mathsf{CT}^*$, output y where $y = f(\mathsf{m}_0^*) = f(\mathsf{m}_1^*)$.
2. Compute $\mathsf{m} = \mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT})$.
3. Output $\perp$ if the decryption aborts.
4. Else, output $f(m)$.

Figure 4: Program for generating function secret key

Note that $\mathsf{Hyb}_{3,0}$ corresponds to Hyb_2.

- **For each** i **in** $\{0, 1, \ldots, q\}$**,** $\mathsf{Hyb}_{4,i}$**:** This hybrid is identical to the previous hybrid except that now, the function secret key SK_f for the i^{th} function key query f is computed as $\mathcal{O}(\mathsf{G}_f^2)$ for the following program G_f^2.

Program G_f^2

Input : ciphertext $\mathsf{CT} = (\mathsf{VK}, \mathsf{CT}_1, \sigma)$
Constants : $\mathsf{SK}, \mathsf{CT}^* = (\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$

1. If $\mathsf{CT} = \mathsf{CT}^*$, output y where $y = f(\mathsf{m}_0^*) = f(\mathsf{m}_1^*)$.
2. If $\mathsf{VK} = \mathsf{VK}^*$, output $\perp$.
3. Compute $\mathsf{m} = \mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT})$.
4. Output $\perp$ if the decryption aborts.
5. Else, output $f(m)$.

Figure 5: Program for generating function secret key

Note that $\mathsf{Hyb}_{4,0}$ corresponds to $\mathsf{Hyb}_{3,q}$.

- **For each** i **in** $\{0, 1, \ldots, q\}$**,** $\mathsf{Hyb}_{5,i}$**:** This hybrid is identical to the previous hybrid except that now, the function secret key SK_f for the i^{th} function key query f is computed as $\mathcal{O}(\mathsf{G}_f^3)$ for the following program G_f^3.

Program G_f^3

Input : ciphertext $\mathsf{CT} = (\mathsf{VK}, \mathsf{CT}_1, \sigma)$
Constants : $\mathsf{SK}, \mathsf{CT}^* = (\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$

1. If $\mathsf{CT} = \mathsf{CT}^*$, output y where $y = f(\mathsf{m}_0^*) = f(\mathsf{m}_1^*)$.
2. If $\mathsf{VK} = \mathsf{VK}^*$, output $\perp$.
3. Check if $\mathsf{OTS.Verify}(\mathsf{VK}, \mathsf{CT}_1, \sigma) = 1$.
4. Compute $\mathsf{m} = \mathsf{PTBE.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{VK}, \mathsf{CT}_1)$.
5. Output $\perp$ if the decryption aborts or if the signature doesn't verify.
6. Else, output $f(m)$.

Figure 6: Program for generating function secret key

Note that $\mathsf{Hyb}_{5,0}$ corresponds to $\mathsf{Hyb}_{4,q}$.

- Hyb_6**:** This hybrid is identical to the previous hybrid except that we now run $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$ $(1^n, \mathsf{VK}^*, \mathsf{m}_\mathsf{b}^*)$ to generate $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$.
- Hyb_7**:** This hybrid is identical to the previous hybrid except that we now run $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}(1^n, \mathsf{VK}^*, \mathsf{m}_0^*)$ to generate $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$

Observe that in this last hybrid, the challenge ciphertext is created independent of the bit b. Hence, the attacker's advantage in this hybrid is 0.

We will now show the indistinguishability of every successive pair of hybrids.

Lemma 5. *Assuming that the* ***indistinguishability of parameters property*** *holds for the scheme* PTBE*,* Hyb_1 *is computationally indistinguishable from* Hyb_2*.*

Proof. The only difference between the two hybrids is the way in which the values $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}_1^*)$ are generated. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids. We will design an adversary $\mathcal{A}_{\mathsf{PTBE}}$ that interacts with a challenger $\mathcal{C}$ and breaks the indistinguishability of parameters property of the scheme PTBE.

First, $\mathcal{A}_{\mathsf{PTBE}}$ generates $(\mathsf{SigK}^*, \mathsf{VK}^*) \leftarrow \mathsf{OTS.Setup}(1^n)$ and sends $(\mathsf{VK}^*, \mathsf{m}_\mathsf{b}^*)$ to the challenger $\mathcal{C}$. $\mathcal{A}_{\mathsf{PTBE}}$ gets back $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}_1^*)$ from the challenger $\mathcal{C}$. Then, $\mathcal{A}_{\mathsf{PTBE}}$ sets $(\mathsf{PK}, \mathsf{SK})$ as the setup parameters of the FE scheme and interacts with $\mathcal{A}$ exactly as in Hyb_1 except for generating the challenge ciphertext. It computes the challenge ciphertext as $(\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$ where $\sigma^* = \mathsf{OTS.Sign}(\mathsf{SigK}^*, \mathsf{CT}_1^*)$. Recall that CT_1^* was given by the challenger $\mathcal{C}$. Now observe that if $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}_1^*)$ were generated by running the algorithms $\mathsf{PTBE.Setup}$ and $\mathsf{PTBE.Enc}$, then this exactly corresponds to Hyb_1 while if they were generated using $\mathsf{PTBE.Setup\text{-}Alt}$, then this corresponds to Hyb_2. Therefore, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{A}_{\mathsf{PTBE}}$ can use the same distinguishing guess to break the indistinguishability of parameters property. □

Lemma 6. *Assuming* $(\mathcal{O}, \mathsf{Eval})$ *is a secure* $\{$*indistinguishability/differing inputs*$\}$ *obfuscator,* $\mathsf{Hyb}_{3,i-1}$ *is computationally indistinguishable from* $\mathsf{Hyb}_{3,i}$ *for all* $i \in \{1, \ldots, q\}$*. In particular, this also implies indistinguishability of* Hyb_2 *and* $\mathsf{Hyb}_{3,1}$*.*

Proof. The only difference between the two hybrids is in the programs that are obfuscated to generate the i^{th} function secret key f. In $\mathsf{Hyb}_{3,i-1}$, the program is generated as in Figure 3 while in $\mathsf{Hyb}_{3,i}$, the program is generated as in Figure 4. The two programs only differ on input the challenge ciphertext CT^*. However, observe that since $f(m_0^*) = f(m_1^*)$ and the decryption algorithm is correct, the two programs are functionally equivalent. Therefore, from the security of the obfuscator, the two hybrids are computationally indistinguishable. □

Lemma 7. $\mathsf{Hyb}_{4,i-1}$ *is computationally indistinguishable from* $\mathsf{Hyb}_{4,i}$ *for all* $i \in \{1, \ldots, q\}$ *assuming either:*

1. $(\mathcal{O}, \mathsf{Eval})$ *is a polynomially secure differing inputs obfuscator and* $\mathsf{OTS} = (\mathsf{OTS.Setup}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$ *is a polynomially secure strong one time signature scheme. (OR)*
2. $(\mathcal{O}, \mathsf{Eval})$ *is a sub-exponentially secure indistinguishability obfuscator and* $\mathsf{OTS} = (\mathsf{OTS.Setup}, \mathsf{OTS.Sign}, \mathsf{OTS.V}$ *is a sub-exponentially secure unique one time signature scheme.*

Proof. We will prove the lemma in two ways.

Proof 1:

We first describe the proof for the first part of the lemma. That is, assuming polynomially secure differing inputs obfuscation and a polynomially secure one time signature scheme, the two hybrids are indistinguishable.

The only difference between the two hybrids is in the programs that are obfuscated to generate the i^{th} function secret key f. In $\mathsf{Hyb}_{4,i-1}$, the program is generated as in Figure 4 while in $\mathsf{Hyb}_{4,i}$, the program is generated as in Figure 5. The two programs only differ on input ciphertexts of the form $(\mathsf{VK}^*, \mathsf{CT}_1, \sigma)$ such that $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1, \sigma) = 1$. That is, in the first case, the program continues to decrypt the ciphertext while in the second case, it always rejects on such inputs.

Sub-Lemma 1. *Assuming* $\mathsf{OTS} = (\mathsf{OTS.Setup}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$ *is a polynomially secure strong one time signature scheme, it is hard for any PPT adversary to find a differing input for the two programs in Figure 4 and Figure 5 respectively.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that can find a differing input given the obfuscation of one of the two programs. We will design an adversary $\mathcal{A}_{\mathsf{OTS}}$ that uses $\mathcal{A}$ to break the security of the one time signature scheme OTS. $\mathcal{A}_{\mathsf{OTS}}$ interacts with a challenger $\mathcal{C}$ and sends CT_1^* (computed as in $\mathsf{Hyb}_{4,i-1}$. It receives a verification key VK^* and a signature $\sigma^* = \mathsf{OTS.Sign}(\mathsf{SigK}^*, \mathsf{CT}_1^*)$. It runs the setup algorithm to generate $(\mathsf{PK}, \mathsf{SK})$. Then, computes the two programs in Figure 4 and Figure 5 and sends them to $\mathcal{A}$ along with the public key PK. Now if, $\mathcal{A}$ finds a differing input $(\mathsf{VK}^*, \mathsf{CT}_1, \sigma)$ such that $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1, \sigma) = 1$, $\mathcal{A}_{\mathsf{OTS}}$ outputs (CT_1, σ) as the forged signature to the challenger $\mathcal{C}$. This breaks the unforgeability of the one time signature scheme and this is a contradiction. □

Sub-Lemma 2. *Assuming the previous sub-lemma holds and* $(\mathcal{O}, \mathsf{Eval})$ *is a polynomially secure differing inputs obfuscator,* $\mathsf{Hyb}_{4,i-1}$ *is indistinguishable from* $\mathsf{Hyb}_{4,i}$.

Proof. From the previous claim, we know that it is computationally hard for any PPT adversary to find a differing input for the two programs. Therefore, from the security of the obfuscator, the two programs are computationally indistinguishable. We know that this is the only difference between the two hybrids. Therefore, if there exists an adversary who can distinguish between the two hybrids, then we can use that adversary to break the security of the differing inputs obfuscator. □

Proof 2:
We now describe the proof for the second part of the lemma. That is, assuming sub-exponentially secure indistinguishability obfuscation and sub-exponentially secure unique one time signature schemes, the two hybrids are indistinguishable.

As mentioned earlier, the only difference between the two hybrids is in the programs that are obfuscated to generate the i^{th} function secret key f. In $\mathsf{Hyb}_{4,i-1}$, the program is generated as in Figure 4 while in $\mathsf{Hyb}_{4,i}$, the program is generated as in Figure 5. The two programs only differ on input ciphertexts of the form $(\mathsf{VK}^*, \mathsf{CT}_1, \sigma)$ such that $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1, \sigma) = 1$. That is, in the first case, the program continues to decrypt the ciphertext while in the second case, it always rejects on such inputs.

Recall that for each potential differing input, CT_1 is a ciphertext generated using the underlying PTBE scheme using tag VK^*. Without loss of generality, let's assume that the set of all possible ciphertexts that can be generated using the underlying PTBE scheme with tag VK^* corresponds to all strings of length $|\mathsf{CT}_1^*|$. Let's say $l = |\mathsf{CT}_1^*|$ and $L = 2^l$. Now, let's go one by one over all these values. That is, we define one intermediate hybrid $\mathsf{Hyb}^j_{4,i-1}$ corresponding to each $j \in L$ such that $\mathsf{Hyb}^0_{4,i-1}$ is the same as $\mathsf{Hyb}_{4,i-1}$ and $\mathsf{Hyb}^L_{4,i-1}$ is the same as $\mathsf{Hyb}_{4,i}$. We then prove that every pair of intermediate hybrids is computationally indistinguishable. In more detail, here are the intermediate hybrids:

- $\mathsf{Hyb}^0_{4,i-1}$**:** This is the same as $\mathsf{Hyb}_{4,i-1}$.
- **For each j in** $[L]$**,** $\mathsf{Hyb}^j_{4,i-1}$: This hybrid is same as the previous hybrid, except that the i^{th} function secret key SK_f is now computed as an obfuscation of the following program $\mathsf{G}^4_{f,j}$. Here, we use $\leq$ based on the lexicographic ordering of all binary strings.

Program $\mathsf{G}^4_{f,j}$

Input : ciphertext $\mathsf{CT} = (\mathsf{VK}, \mathsf{CT}_1, \sigma)$
Constants : $\mathsf{SK}, \mathsf{CT}^* = (\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$

1. If $\mathsf{CT} = \mathsf{CT}^*$, output y where $y = f(\mathsf{m}_0^*) = f(\mathsf{m}_1^*)$.
2. Let CT_1^j denote the j^{th} string in $\{0,1\}^l$.
3. If $\mathsf{VK} = \mathsf{VK}^*$ and $\mathsf{CT}_1 \leq \mathsf{CT}_1^j$, output $\perp$.
4. Compute $\mathsf{m} = \mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT})$.
5. Output $\perp$ if the decryption aborts.
6. Else, output $f(m)$.

Figure 7: Program for generating function secret key

Note that $\mathsf{Hyb}^L_{4,i-1}$ corresponds to $\mathsf{Hyb}_{4,i}$.

Claim 11. *Assuming* $(\mathcal{O}, \mathsf{Eval})$ *is a sub-exponentially secure indistinguishability obfuscator and* $\mathsf{OTS} = (\mathsf{OTS.Setup}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$ *is a sub-exponentially secure unique one time signature scheme,* $\mathsf{Hyb}^{j-1}_{4,i-1}$ *is computationally indistinguishable from* $\mathsf{Hyb}^j_{4,i-1}$ *for all* $j \in [L]$.

Proof. The only difference between the two hybrids is in the program used to obfuscate the i^{th} function secret key. The two programs differ on inputs of the form $(\mathsf{VK}^*, \mathsf{CT}_1^j, \sigma)$, (where CT_1^j is the j^{th} string in $\{0,1\}^l$) if $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1^j, \sigma) = 1$. On such inputs, the program in $\mathsf{Hyb}_{4,i-1}^j$ aborts whereas the program in $\mathsf{Hyb}_{4,i-1}^{j-1}$ doesn't abort and instead continues decrypting the ciphertext.

First, observe that OTS is a unique signature scheme, there exists only value of σ such that $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1^j, \sigma) = 1$. Let's call this value σ^j. Therefore, the two programs differ on exactly one input. Now, similar to the proof of *Sub − Lemma* 1, it can be seen that no PPT adversary can find a differing input between the two programs.

Now, we invoke a result of [BCP14] that states that if two programs differ on only a constant number of points, then for such programs, diO is implied by iO. Therefore, now, since we know that the two programs in $\mathsf{Hyb}_{4,i-1}^{j-1}$ and $\mathsf{Hyb}_{4,i-1}^{j}$ respectively, differ on only 1 input and it is computationally hard to find that input, by the security of the indistinguishability obfuscator, these two obfuscated programs are computationally indistinguishable. This completes the proof of indistinguishability between these two hybrids.

□

Observe that since we have an exponential number of hybrids, (i.e one for each PTBE ciphertext with tag VK^*), we require that the adversary's advantage in distinguishing each pair of consecutive hybrids be sub-exponentially small. For this reason, we use sub-exponentially secure iO and sub-exponentially secure unique one time signatures rather than polynomially secure ones. A similar idea was previously explored in the work of Brakerski et al. [BCG+17]. □

Lemma 8. *Assuming* $(\mathcal{O}, \mathsf{Eval})$ *is a secure* {*indistinguishability/differing inputs*} *obfuscator and that the* ***equivalent on all but challenge tag property*** *holds for the scheme* PTBE, $\mathsf{Hyb}_{5,i-1}$ *is computationally indistinguishable from* $\mathsf{Hyb}_{5,i}$ *for all* $i \in \{1, \ldots, q\}$.

Proof. The only difference between the two hybrids is in the programs that are obfuscated to generate the i^{th} function secret key f. In $\mathsf{Hyb}_{5,i-1}$, the program is generated as in Figure 5 while in $\mathsf{Hyb}_{5,i}$, the program is generated as in Figure 6. The two programs only differ in the way the input ciphertext is decrypted. However, from the equivalent on all but okay property, we know that for all $\mathsf{m}_b^*, \mathsf{VK}^*, \mathsf{CT}_1, \mathsf{VK}$ with $\mathsf{VK} \neq \mathsf{VK}^*$, $\mathsf{PTBE.Dec}(\mathsf{SK}, \mathsf{VK}, \mathsf{CT}_1) = \mathsf{PTBE.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{VK}, \mathsf{CT}_1)$. Therefore, the two programs are functionally equivalent, and so, from the security of the obfuscator, the two hybrids are computationally indistinguishable.

□

Lemma 9. *Assuming that the* ***indistinguishability of alternate setups property*** *holds for the scheme* PTBE, $\mathsf{Hyb}_{5,q}$ *is computationally indistinguishable from* Hyb_6.

Proof. Now, the only difference between the two hybrids is the way in which the values $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ are generated. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids. We will design an adversary $\mathcal{A}_{\mathsf{PTBE}}$ that interacts with a challenger $\mathcal{C}$ and breaks the indistinguishability of messages property of the scheme PTBE.

First, $\mathcal{A}_{\mathsf{PTBE}}$ receives the challenge messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$ from $\mathcal{A}$. Then, it generates $(\mathsf{SigK}^*, \mathsf{VK}^*) \leftarrow \mathsf{OTS.Setup}(1^n)$ and submits $(\mathsf{VK}^*, \mathsf{m}_0^*, \mathsf{m}_1^*)$ to the challenger $\mathcal{C}$. $\mathcal{A}_{\mathsf{PTBE}}$ receives $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ from $\mathcal{C}$. Then, $\mathcal{A}_{\mathsf{PTBE}}$ sets $(\mathsf{PK}, \mathsf{SK\text{-}Alt})$ as the setup parameters of the FE scheme and interacts with $\mathcal{A}$ exactly as in $\mathsf{Hyb}_{5,q}$. Observe that if $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ were generated by running the algorithms $\mathsf{PTBE.Setup\text{-}Alt}$ on input $(\mathsf{VK}^*, \mathsf{m}_1^*)$, then this exactly corresponds to $\mathsf{Hyb}_{5,q}$ while if they were generated using $\mathsf{PTBE.Setup\text{-}Alt}$ on input $(\mathsf{VK}^*, \mathsf{m}_0^*)$, then this corresponds to Hyb_6. Therefore, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{A}_{\mathsf{PTBE}}$ can use the same distinguishing guess to break the indistinguishability of alternate setups property. □

Lemma 10. *Assuming that the* ***indistinguishability of messages property*** *holds for the scheme* PTBE*,* Hyb_6 *is computationally indistinguishable from* Hyb_7*.*

Proof. First, observe that in Hyb_6, if the challenger sets $\mathsf{b} = 0$, then the two hybrids are identical. Therefore, they can potentially be distinguished only if $\mathsf{b} = 1$ in Hyb_6.

Now, the only difference between the two hybrids is the way in which the values $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ are generated. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids. We will design an adversary $\mathcal{A}_{\mathsf{PTBE}}$ that interacts with a challenger $\mathcal{C}$ and breaks the indistinguishability of messages property of the scheme PTBE.

First, $\mathcal{A}_{\mathsf{PTBE}}$ receives the challenge messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$ from $\mathcal{A}$. Then, it generates $(\mathsf{SigK}^*, \mathsf{VK}^*) \leftarrow \mathsf{OTS.Setup}(1^n)$ and submits $(\mathsf{VK}^*, \mathsf{m}_0^*, \mathsf{m}_1^*)$ to the challenger $\mathcal{C}$. $\mathcal{A}_{\mathsf{PTBE}}$ receives $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ from $\mathcal{C}$. Then, $\mathcal{A}_{\mathsf{PTBE}}$ sets $(\mathsf{PK}, \mathsf{SK\text{-}Alt})$ as the setup parameters of the FE scheme and interacts with $\mathcal{A}$ exactly as in Hyb_6. Observe that if $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ were generated by running the algorithms $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$ on input $(\mathsf{VK}^*, \mathsf{m}_1^*)$, then this exactly corresponds to Hyb_6 while if they were generated using $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$ on input $(\mathsf{VK}^*, \mathsf{m}_0^*)$, then this corresponds to Hyb_7. Therefore, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{A}_{\mathsf{PTBE}}$ can use the same distinguishing guess to break the indistinguishability of messages property. □

8 Acknowledgements

The third author's research is supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1619348, 1228984, 1136174, and 1065276, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

The fourth author's research is supported by NSF CNS-1228599 and CNS-1414082, DARPA SafeWare, Microsoft Faculty Fellowship, and Packard Foundation Fellowship. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of Defense or the U.S. Government.

References

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2015.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *Eurocrypt*, 2004.

[BCG+17] Zvika Brakerski, Nishanth Chandran, Vipul Goyal, Aayush Jain, Amit Sahai, and Gil Segev. Hierarchical functional encryption. In *ITCS*, 2017.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*, pages 52–73, 2014.

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

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BGJS15] Saikrishna Badrinarayanan, Divya Gupta, Abhishek Jain, and Amit Sahai. Multi-input functional encryption for unbounded arity functions. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, volume 9452 of *Lecture Notes in Computer Science*, pages 27–51. Springer, 2015.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, pages 92–111, 1994.

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

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *Eurocrypt*, 2004.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, 1999.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[Gen10] Craig Gentry. Toward basing fully homomorphic encryption on worst-case hardness. In *CRYPTO*, 2010.

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 578–602. Springer, 2014.

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

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. *IACR Cryptology ePrint Archive*, 2017.

[GS02] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *Asiacrypt 2002*, 2002.

[Had00] Satoshi Hada. Zero-knowledge and code obfuscation. In *ASIACRYPT*, 2000.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In *EUROCRYPT*, 2015.

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *Eurocrypt*, 2002.

[HLW12] Susan Hohenberger, Allison B. Lewko, and Brent Waters. Detecting dangerous queries: A new approach for chosen ciphertext security. In *EUROCRYPT*, 2012.

[JK03] Jakob Jonsson and Burt Kaliski. Public-key cryptography standards (pkcs)# 1: Rsa cryptography specifications version 2.1. 2003.

[Kil06] Eike Kiltz. Chosen-ciphertext security from tag-based encryption. In *TCC*, 2006.

[Lam79] Lamport. Constructing digital signatures from a one-way function. *Technical Report SRI-CSL-98, SRI International Computer Science Laboratory*, 1979.

[MH14] Takahiro Matsuda and Goichiro Hanaoka. Chosen ciphertext security via UCE. In Hugo Krawczyk, editor, *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*, pages 56–76. Springer, 2014.

[MS09] Steven Myers and Abhi Shelat. Bit encryption is complete. In *FOCS*, 2009.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, 1990.

[PW07] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. *IACR Cryptology ePrint Archive*, page 279, 2007.

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In *STOC*, 2008.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, 1999.

[SP92] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction (extended abstract). In *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, pages 427–436. IEEE Computer Society, 1992.

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

[SW08] Amit Sahai and Brent Waters. Slides on functional encryption, powerpoint presentation. 2008.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 2014.

[Wat14] Brent Waters. A punctured programming approach to adaptively secure functional encryption. *IACR Cryptology ePrint Archive*, 2014:588, 2014.

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. *IACR Cryptology ePrint Archive*, 2017.

A Security Notions for Public Key Encryption

A.1 CCA Security

We define the security notion for a public key encryption scheme using the following game ($\mathsf{Adaptive} - \mathsf{IND} - \mathsf{CCA}$) between a challenger and an adversary.

Setup Phase: The challenger generates $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{PKE.Setup}(1^n)$ and then hands over the public key PK to the adversary.
Decryption Oracle Phase 1: The adversary adaptively queries the decryption oracle with any ciphertext CT of his choice. The challenger responds by giving the adversary the corresponding plaintext $\mathsf{m} \leftarrow \mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT})$.
Challenge Phase: The adversary chooses two messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$ of the same size (each in $\mathcal{X}_\setminus$)) and sends them to the challenger. The challenger selects a random bit $b \in \{0, 1\}$ and sends a ciphertext $\mathsf{CT}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m}_b^*)$ to the adversary.

Decryption Oracle Phase 2: The adversary can continue querying the decryption oracle adaptively on all ciphertext except the challenge ciphertext CT^*.
Guess: The adversary submits a guess $b^{'}$ and wins if $b^{'} = b$. The adversary's advantage in this game is defined to be $2 * |\mathsf{Pr}[\mathsf{b} = \mathsf{b}^{'}] - 1/2|$.

We also define the *selective* security game, which we call ($\mathsf{Selective} - \mathsf{IND} - \mathsf{CCA}$) where the adversary outputs the challenge message pair even before seeing the master public key.

Definition 5. *An encryption scheme* FE *is selective/adaptive* $\mathsf{IND} - \mathsf{CCA}$ *secure if all PPT adversaries have at most a negligible advantage in the* $\mathsf{Selective/Adaptive} - \mathsf{IND} - \mathsf{CCA}$ *security game.*

A.2 CPA Security

The $\mathsf{IND} - \mathsf{CPA}$ security notion is same as $\mathsf{IND} - \mathsf{CCA}$ except that in the security game, the adversary does not get access to the decryption oracle.

B Further Preliminaries

B.1 Indistinguishability Obfuscation

We recall the notion of indistinguishability obfuscation for circuits[BGI+01, GGH+13].

Definition 6. *A pair of uniform PPT machines* $(\mathcal{O}, \mathsf{Eval})$ *is called an indistinguishability obfuscator for a circuit class* $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbf{N}}$ *if the following conditions are satisfied:*

- *Functionality :*
  For every $\lambda \in \mathbf{N}$*, every* $C \in \mathcal{C}_\lambda$ *, every input* x *to* C *:*

$$Pr[\mathsf{Eval}(\mathcal{O}(C), x) \neq C(x)] <= negl(|C|),$$

where the probability is over the coins of $\mathcal{O}$.

- *Polynomial Slowdown :*
  There exists a polynomial q such that for every $\lambda \in \mathbf{N}$ and every $C \in \mathcal{C}_\lambda$, we have that $|\mathcal{O}(C)| <= q(|C|)$.

- *Indistinguishability :*
  For all PPT distinguishers $\mathcal{D}$, there exists a negligible function α such that for every $\lambda \in \mathbf{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have that if $C_0(x) = C_1(x)$ for all inputs x, then

$$|Pr[D(\mathcal{O}(C_0))] - Pr[D(\mathcal{O}(C_1))]| \leq \alpha(\lambda)$$

.

B.2 Differing Inputs Obfuscation

In this section, we recall the notion of differing inputs obfuscation for circuits [BGI+01]. First, we describe the notion of a differing-inputs circuit family. Intuitively, we call a circuit family to be differing-inputs circuit family if there does not exist any PPT adversary who, given two circuits sampled from a distribution defined on this circuit family, can output a value such that both the circuits differ on this input.

Definition 7 (Differing-Inputs Sampler)**.** *A circuit family $\mathcal{C}$ associated with a PPT* Sampler *is said to be a differing-inputs circuit family if for every PPT adversary $\mathcal{A}$ there exists a negligible function α such that:*

$$Pr\Big[C_0(x) \neq C_1(x) : (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\lambda), x \leftarrow \mathcal{A}(1^\lambda, C_0, C_1, \mathsf{aux})\Big] \leq \alpha(\lambda)$$

Definition 8 (Differing-Inputs Obfuscator)**.** *A pair of uniform PPT machines $(\mathcal{O}, \mathsf{Eval})$ is called a differing-inputs obfuscator for a differing-inputs circuit class $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbf{N}}$ if the following conditions are satisfied:*

- *Functionality :*
  For every $\lambda \in \mathbf{N}$, every $C \in \mathcal{C}_\lambda$, every input x to C :

$$Pr[\mathsf{Eval}(\mathcal{O}(C), x) \neq C(x)] <= negl(|C|),$$

  where the probability is over the coins of $\mathcal{O}$.

- *Polynomial Slowdown :*
  There exists a polynomial q such that for every $\lambda \in \mathbf{N}$ and every $C \in \mathcal{C}_\lambda$, we have that $|\mathcal{O}(C)| <= q(|C|)$.

- *Indistinguishability :*
  For all PPT distinguishers $\mathcal{D}$, there exists a negligible function α such that the following holds: for every $\lambda \in \mathbf{N}$, for all $(C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Sampler}(1^\lambda)$, we have that

$$|Pr[D(\mathcal{O}(C_0), \mathsf{aux})] - Pr[D(\mathcal{O}(C_1), \mathsf{aux})]| \leq \alpha(\lambda)$$

.

B.3 Lockable Obfuscation

We recall the recently introduced notion of lockable obfuscation[GKW17, WZ17]. We take this section verbatim from [GKW17].

A lockable obfuscator takes as input a program P, a message m, and a 'lock' α. It outputs an obfuscated program $\tilde{P}$ which has the same domain as the program P. The program $\tilde{P}$, on input x, outputs the message m if $P(x) = \alpha$. If not, then with overwhelming probability, it will output $\perp$. For security, we require that for all programs P and messages m, $\tilde{P}$ for a uniformly random α can be efficiently simulated given only the sizes of P and m. We will now present the syntax, correctness and security definition.

Let n, m, d be polynomials, and $\mathcal{C}_{n,m,d}(\lambda)$ be the class of depth $d(\lambda)$ circuits with $n(\lambda)$ bit input and $m(\lambda)$ bit output. A lockable obfuscator for $\mathcal{C}_{n,m,d}$ consists of algorithms $\mathcal{O}$ and Eval with the following syntax. Let $\mathcal{M}$ be the message space.

- $\mathcal{O}(1^\lambda, P, \mathsf{m}, \alpha) \rightarrow \tilde{P}$. The obfuscation algorithm is a randomized algorithm that takes as input the security parameter λ, a program $P \in \mathcal{C}_{n,m,d}$, message $\mathsf{m} \in \mathcal{M}$ and 'lock string' $\alpha \in \{0,1\}^{m(\lambda)}$. It outputs a program $\tilde{P}$.
- $\mathsf{Eval}(\tilde{P}, x) \rightarrow y \in \mathcal{M} \cup \{\perp\}$. The evaluator is a deterministic algorithm that takes as input a program $\tilde{P}$ and a string $x \in \{0,1\}^{n(\lambda)}$. It outputs $y \in \mathcal{M} \cup \{\perp\}$.

B.3.1 Correctness

For correctness, we informally require that if $P(x) = \alpha$, then the obfuscated program $\tilde{P} \leftarrow \mathcal{O}(1^\lambda, P, \mathsf{m}, \alpha)$, evaluated on input x, outputs m, and if $P(x) \neq \alpha$, then $\tilde{P}$ outputs $\perp$ on input x. definitions, which differ in the case where $P(x) \neq \alpha$.

Definition 9 (Perfect Correctness)**.** *A lockable obfuscation scheme is said to be perfectly correct if it satisfies the following properties:*

1. *For all security parameters λ, inputs $x \in \{0,1\}^{n(\lambda)}$, programs $P \in \mathcal{C}_{n,m,d}$ and messages $\mathsf{m} \in \mathcal{M}$, if $P(x) = \alpha$, then*
$$\mathsf{Eval}(\mathcal{O}(1^\lambda, P, \mathsf{m}, \alpha)), x) = \mathsf{m}.$$
2. *For all security parameters λ, inputs $x \in \{0,1\}^{n(\lambda)}$, programs $P \in \mathcal{C}_{n,m,d}$ and messages $\mathsf{m} \in \mathcal{M}$, if $P(x) \neq \alpha$, then*
$$\mathsf{Eval}(\mathcal{O}(1^\lambda, P, \mathsf{m}, \alpha)), x) = \perp .$$

B.3.2 Security

Definition 10. *Let n, m, d be polynomials and λ be the security parameter. A lockable obfuscation scheme $(\mathcal{O}, \mathsf{Eval})$ for $\mathcal{C}_{n,m,d}$ and message space $\mathcal{M}$ is said to be secure if there exists a PPT simulator Sim such that for all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, there exists a negligible function $\mathsf{negligible}(\cdot)$ such that the following function is bounded by $\mathsf{negligible}(\cdot)$:*

$$\left| Pr\left[\mathcal{A}_1(\tilde{P}_b, \mathsf{st}) = 1 : \begin{array}{c} (P, \mathsf{m}, \mathsf{st}) \leftarrow \mathcal{A}_0(1^\lambda), b \leftarrow \{0,1\}, \alpha \leftarrow \{0,1\}^{m(\lambda)}, \\ \tilde{P}_0 \leftarrow \mathcal{O}(1^\lambda, P, \mathsf{m}, \alpha), \tilde{P}_1 \leftarrow \mathsf{Sim}(1^\lambda, 1^{|P|}, 1^{|\mathsf{m}|}) \end{array} \right] - \frac{1}{2} \right|$$

C Proof of Security for Special-CCA

We now prove that the above scheme is selective IND-CCA secure by proving the indistinguishability of the following series of hybrid arguments. Consider a challenger $\mathcal{C}$ who interacts with an adversary $\mathcal{A}$.

- Hyb_1 : The adversary sends two messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$ to $\mathcal{C}$. The challenger then runs $\mathsf{PKE.Setup}(1^n)$ to generate $(\mathsf{PK}, \mathsf{SK})$. $\mathcal{C}$ computes $\mathsf{CT}^* = \mathsf{PKE.Enc}(\mathsf{PK}, \mathsf{m}_\mathsf{b}^*)$ where b is picked randomly. Then, $\mathcal{C}$ gives $(\mathsf{PK}, \mathsf{CT}^*)$ to the adversary $\mathcal{A}$. $\mathcal{C}$ answers the adversary's decryption queries by running the algorithm $\mathsf{PKE.Dec}$.

- Hyb_2 : In this hybrid, the challenger doesn't run the algorithm $\mathsf{PKE.Setup}$ directly. Instead, he does the following: generate $(\mathsf{VK}^*, \mathsf{SigK}^*) \leftarrow \mathsf{OTS.Setup}(1^n)$ and computes $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*) \leftarrow \mathsf{PTBE.Setup\text{-}Alt}(1^n, \mathsf{VK}^*, \mathsf{m}_\mathsf{b}^*)$ where b is picked at random. $\mathcal{C}$ sets the public key of the scheme to be PK and the secret key to be SK. Then, computes $\sigma^* = \mathsf{OTS.Sign}(\mathsf{SigK}^*, \mathsf{CT}_1^*)$ and sets $\mathsf{CT}^* = (\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$. $\mathcal{C}$ gives $(\mathsf{PK}, \mathsf{CT}^*)$ to the adversary and answers the decryption queries as before by running the algorithm $\mathsf{PKE.Dec}$. Notice that $\mathsf{SK\text{-}Alt}$ is not used at all in this hybrid.

- Hyb_3 : This hybrid is same as the previous hybrid except that now, the decryption oracle, in addition to the challenge ciphertext, outputs $\perp$ also to queries of the following form: $\mathsf{CT} = (\mathsf{VK}^*, \mathsf{CT}_1, \sigma)$ where $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1, \sigma) = 1$ (this form also captures the challenge ciphertext).

- Hyb_4 : This hybrid is same as the previous hybrid except that the decryption oracle on input a ciphertext CT now does the following:

  1. Parse $\mathsf{CT} = (\mathsf{VK}, \mathsf{CT}_1, \sigma)$.
  2. If $\mathsf{VK} = \mathsf{VK}^*$ and $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1, \sigma) = 1$, output $\perp$.
  3. Output $\perp$ if $\mathsf{OTS.Verify}(\mathsf{VK}, \mathsf{CT}_1, \sigma) = 0$.
  4. Output $\mathsf{m} = \mathsf{PTBE.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{VK}, \mathsf{CT}_1)$.

  Notice that we start using $\mathsf{SK\text{-}Alt}$ now and the secret key SK is no longer used. The red coloured text highlights the difference from the previous decryption oracle.

- Hyb_5 : This is same as the previous hybrid except that $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$ is computed using the algorithm $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$. That is, we run $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}(1^n, \mathsf{VK}^*, \mathsf{m}_\mathsf{b}^*)$.

- Hyb_6 : This hybrid is same as the previous hybrid except that now CT^* is computed as an encryption of m_0^*. That is, we run $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}(1^n, \mathsf{VK}^*, \mathsf{m}_0^*)$.

Observe that in this last hybrid, the challenge ciphertext is created independent of the bit b. Hence, the attacker's advantage in this hybrid is negligible.

We will now show the indistinguishability of every successive pair of hybrids.

Lemma 11. *Assuming that the* ***indistinguishability of parameters property*** *holds for the scheme* PTBE, Hyb_1 *is computationally indistinguishable from* Hyb_2.

Proof. The only difference between the two hybrids is the way in which the values $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}_1^*)$ are generated. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids. We

will design an adversary $\mathcal{A}_{\mathsf{PTBE}}$ that interacts with a challenger $\mathcal{C}$ and breaks the indistinguishability of parameters property of the scheme PTBE.

First, $\mathcal{A}_{\mathsf{PTBE}}$ generates $(\mathsf{SigK}^*, \mathsf{VK}^*) \leftarrow \mathsf{OTS.Setup}(1^n)$ and sends $(\mathsf{VK}^*, \mathsf{m}_\mathsf{b}^*)$ to the challenger $\mathcal{C}$. $\mathcal{A}_{\mathsf{PTBE}}$ gets back $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}_1^*)$ from the challenger $\mathcal{C}$. Then, $\mathcal{A}_{\mathsf{PTBE}}$ sets $(\mathsf{PK}, \mathsf{SK})$ as the public key and secret key and interacts with $\mathcal{A}$ exactly as in Hyb_1 except for generating the challenge ciphertext. It computes the challenge ciphertext as $(\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$ where $\sigma^* = \mathsf{OTS.Sign}(\mathsf{SigK}^*, \mathsf{CT}_1^*)$. Recall that CT_1^* was given by the challenger $\mathcal{C}$. Now observe that if $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}_1^*)$ were generated by running the algorithms $\mathsf{PTBE.Setup}$ and $\mathsf{PTBE.Enc}$, then this exactly corresponds to Hyb_1 while if they were generated using $\mathsf{PTBE.Setup\text{-}Alt}$, then this corresponds to Hyb_2. Therefore, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{A}_{\mathsf{PTBE}}$ can use the same distinguishing guess to break the indistinguishability of parameters property. □

Lemma 12. *Assuming* $\mathsf{OTS} = (\mathsf{OTS.Setup}, \mathsf{OTS.Sign}, \mathsf{OTS.Verify})$ *is a strongly secure one time signature scheme,* Hyb_2 *is computationally indistinguishable from* Hyb_3*.*

Proof. The only difference between the two hybrids is that in Hyb_3, the decryption oracle outputs $\perp$ for all queries of the form $(\mathsf{VK}^*, \mathsf{CT}_1, \sigma)$ such that $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1, \sigma) = 1$ whereas in Hyb_2, the decryption oracle doesn't reject such queries (except the challenge ciphertext). Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids. We will design an adversary $\mathcal{A}_{\mathsf{OTS}}$ that interacts with a challenger $\mathcal{C}$ and breaks the security of the one time signature scheme OTS.

$\mathcal{A}_{\mathsf{OTS}}$ interacts with a challenger $\mathcal{C}$ and receives a verification key VK^*. Also, $\mathcal{A}_{\mathsf{OTS}}$ interacts with $\mathcal{A}$ and receives two messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$. $\mathcal{A}_{\mathsf{OTS}}$ generates $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*) \leftarrow \mathsf{PTBE.Setup\text{-}Alt}(\mathsf{VK}^*, \mathsf{m}_\mathsf{b}^*)$ where b is picked randomly. Then, $\mathcal{A}_{\mathsf{OTS}}$ sends CT_1^* to $\mathcal{C}$ and receives back a signature σ^*. Then, $\mathcal{A}_{\mathsf{OTS}}$ sends $\mathsf{CT}^* = (\mathsf{VK}^*, \mathsf{CT}_1^*, \sigma^*)$ to $\mathcal{A}$ as the challenge ciphertext.

Now if $\mathcal{A}$ can distinguish between the two hybrids, it must make a decryption query of the form $\mathsf{CT} = (\mathsf{VK}^*, \mathsf{CT}_1, \sigma)$ such that $\mathsf{CT} \neq \mathsf{CT}^*$ and $\mathsf{OTS.Verify}(\mathsf{VK}^*, \mathsf{CT}_1, \sigma) = 1$. At this point, $\mathcal{A}_o ts$ aborts the experiment with $\mathcal{A}$ and outputs (CT_1, σ) as a forgery to the challenger $\mathcal{C}$ thus breaking the security of the one time signature scheme and this is a contradiction. □

Lemma 13. *Assuming that the* ***equivalent on all but challenge tag property*** *holds for the scheme* PTBE*,* Hyb_3 *is computationally indistinguishable from* Hyb_4*.*

Proof. The difference between the two hybrids is in the decryption oracle's response. That is, given a query of the form $\mathsf{CT} = (\mathsf{VK}, \mathsf{CT}_1, \sigma)$ with $\mathsf{VK} \neq \mathsf{VK}^*$ and $\mathsf{OTS.Verify}(\mathsf{VK}, \mathsf{CT}_1, \sigma) = 1$, in Hyb_4, the decryption oracle outputs $\mathsf{PTBE.Dec}(\mathsf{SK}, \mathsf{VK}, \mathsf{CT}_1)$ while the output in Hyb_5 is $\mathsf{PTBE.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{VK}, \mathsf{CT}_1)$.
However, from the equivalent on all but okay property, we know that for all $\mathsf{m}_b^*, \mathsf{VK}^*, \mathsf{CT}_1, \mathsf{VK}$ with $\mathsf{VK} \neq \mathsf{VK}^*$, $\mathsf{PTBE.Dec}(\mathsf{SK}, \mathsf{VK}, \mathsf{CT}_1) = \mathsf{PTBE.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{VK}, \mathsf{CT}_1)$. Therefore, the two hybrids are computationally indistinguishable.

□

Lemma 14. *Assuming that the* ***indistinguishability of alternate setups*** *holds for the scheme* PTBE*,* Hyb_4 *is computationally indistinguishable from* Hyb_5*.*

Proof. Now, the only difference between the two hybrids is the way in which the values $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ are generated. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids. We will design an adversary $\mathcal{A}_{\mathsf{PTBE}}$ that interacts with a challenger $\mathcal{C}$ and breaks the indistinguishability of alternate setups property of the scheme PTBE.

First, $\mathcal{A}_{\mathsf{PTBE}}$ receives the challenge messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$ from $\mathcal{A}$. Then, it generates $(\mathsf{SigK}^*, \mathsf{VK}^*) \leftarrow \mathsf{OTS.Setup}(1^n)$ and submits $(\mathsf{VK}^*, \mathsf{m}_0^*, \mathsf{m}_1^*)$ to the challenger $\mathcal{C}$. $\mathcal{A}_{\mathsf{PTBE}}$ receives $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ from $\mathcal{C}$. Then, $\mathcal{A}_{\mathsf{PTBE}}$ sets $(\mathsf{PK}, \mathsf{SK\text{-}Alt})$ as the setup parameters of the scheme and interacts with $\mathcal{A}$ exactly as in Hyb_4. Observe that if $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ were generated by running the algorithms $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$ on input $(\mathsf{VK}^*, \mathsf{m}_1^*)$, then this exactly corresponds to Hyb_4 while if they were generated using $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$ on input $(\mathsf{VK}^*, \mathsf{m}_0^*)$, then this corresponds to Hyb_5. Therefore, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{A}_{\mathsf{PTBE}}$ can use the same distinguishing guess to break the indistinguishability of alternate setups property. Further, note that since $\mathsf{SK\text{-}Alt}$ is equal in both cases, the decryption oracle answers exactly the same way. □

Lemma 15. *Assuming that the* ***indistinguishability of messages property*** *holds for the scheme* PTBE*,* Hyb_5 *is computationally indistinguishable from* Hyb_6*.*

Proof. First, observe that in Hyb_5, if the challenger sets $\mathsf{b} = 0$, then the two hybrids are identical. Therefore, they can potentially be distinguished only if $\mathsf{b} = 1$ in Hyb_5.

Now, the only difference between the two hybrids is the way in which the values $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ are generated. Suppose there exists an adversary $\mathcal{A}$ that can distinguish between the two hybrids. We will design an adversary $\mathcal{A}_{\mathsf{PTBE}}$ that interacts with a challenger $\mathcal{C}$ and breaks the indistinguishability of messages property of the scheme PTBE.

First, $\mathcal{A}_{\mathsf{PTBE}}$ receives the challenge messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$ from $\mathcal{A}$. Then, it generates $(\mathsf{SigK}^*, \mathsf{VK}^*) \leftarrow \mathsf{OTS.Setup}(1^n)$ and submits $(\mathsf{VK}^*, \mathsf{m}_0^*, \mathsf{m}_1^*)$ to the challenger $\mathcal{C}$. $\mathcal{A}_{\mathsf{PTBE}}$ receives $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ from $\mathcal{C}$. Then, $\mathcal{A}_{\mathsf{PTBE}}$ sets $(\mathsf{PK}, \mathsf{SK\text{-}Alt})$ as the setup parameters of the scheme and interacts with $\mathcal{A}$ exactly as in Hyb_5. Observe that if $(\mathsf{PK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}_1^*)$ were generated by running the algorithms $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$ on input $(\mathsf{VK}^*, \mathsf{m}_1^*)$, then this exactly corresponds to Hyb_5 while if they were generated using $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$ on input $(\mathsf{VK}^*, \mathsf{m}_0^*)$, then this corresponds to Hyb_6. Therefore, if $\mathcal{A}$ can distinguish between the two hybrids, $\mathcal{A}_{\mathsf{PTBE}}$ can use the same distinguishing guess to break the indistinguishability of messages property. □

D Examples of Special-CCA Secure Encryption Schemes

In this section, we show that the CCA secure encryption schemes in several popular works in literature are in fact "Special-CCA" as defined in Section 7.2.

D.1 Scheme in [CHK04]

Let X denote the CCA secure encryption scheme constructed by Canetti et al.[CHK04]. X is constructed by adding one time signatures to the below scheme Y (based on any selective-id secure identity based encryption[BF01]) exactly the way we convert any puncturable tag based encryption into a Special-CCA secure encryption scheme in Section 7.2. Therefore, in order to prove that X is a Special-CCA secure encryption scheme, it is enough to prove that Y is a puncturable tag based encryption scheme. However, note that the scheme Y implicitly described in the work of [CHK04] has only three algorithms - $\mathsf{Y.Setup}, \mathsf{Y.Enc}, \mathsf{Y.Dec}$. Therefore, in order to prove that it is a puncturable tag based encryption, we have to design three more algorithms $\mathsf{Y.Setup\text{-}Alt}, \mathsf{Y.Setup\text{-}Alt\text{-}1}, \mathsf{Y.Dec\text{-}Alt}$ that together satisfy the required properties.

For better understanding, we first describe the scheme Y (just with the 3 algorithms) that is implicit in [CHK04]. In the next sub-section, we show that if we use the selective-id secure IBE scheme of Boneh and Boyen[BB04], Y is a puncturable tag based encryption scheme. In the subsequent subsection, we show that if Y is instantiated using any selective-id secure Hierarchical IBE scheme [GS02, HL02], then Y is a puncturable tag based encryption scheme.

Notation: Let the security parameter be λ. Let $\mathsf{IBE} = (\mathsf{IBE.Setup}, \mathsf{IBE.Enc}, \mathsf{IBE.KeyGen}, \mathsf{IBE.Dec})$ be any selective-id secure identity-based encryption scheme. The scheme $\mathsf{Y} = (\mathsf{Y.Setup}, \mathsf{Y.Enc}, \mathsf{Y.Dec})$ is as follows:

Construction:

- $\mathsf{Y.Setup}(1^\lambda)$:
  1. Generate $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{IBE.Setup}(1^\lambda)$.
- $\mathsf{Y.Enc}(\mathsf{PK}, \mathsf{tag}, \mathsf{m})$:
  1. Output $\mathsf{CT} = \mathsf{IBE.Enc}(\mathsf{PK}, \mathsf{tag}, \mathsf{m})$. That is, set the identity as tag.
- $\mathsf{Y.Dec}(\mathsf{SK}, \mathsf{tag}, \mathsf{CT})$:
  1. First, compute $\mathsf{SK}_{\mathsf{tag}} = \mathsf{IBE.KeyGen}(\mathsf{SK}, \mathsf{tag})$.
  2. Output $\mathsf{m} = \mathsf{IBE.Dec}(\mathsf{SK}_{\mathsf{tag}}, \mathsf{CT})$.

D.1.1 [CHK04] using IBE scheme in [BB04]

In this section, we show that when instantiated with the selective-id secure IBE scheme of Boneh and Boyen[BB04], Y is a puncturable tag based encryption scheme.

Notation: Let G be a bilinear group of prime order p (the security parameter determines the size of G). Let $e : \mathsf{G} \times \mathsf{G} \to \mathsf{G}_1$ be the bilinear map. Let the space of all tags be Z_p. Let the message space be all elements in G_1. The scheme $\mathsf{Y} = (\mathsf{Y.Setup}, \mathsf{Y.Enc}, \mathsf{Y.Dec}, \mathsf{Y.Setup\text{-}Alt}, \mathsf{Y.Setup\text{-}Alt\text{-}1}, \mathsf{Y.Dec\text{-}Alt})$ is as follows:

Construction:

- $\mathsf{Y.Setup}(1^\lambda)$:
  1. Choose $(\mathsf{g}, \mathsf{h}) \in \mathsf{G}$ and $(\mathsf{a}, \mathsf{b}) \in Z_p$ randomly.
  2. Set $\mathsf{u} = \mathsf{g}^{\mathsf{a}}, \mathsf{v} = e(\mathsf{g}, \mathsf{g})^{\mathsf{ab}}$.
  3. Define a function $f : Z_p \to G$ such that for any value $\mathsf{t} \in Z_p$, $f(\mathsf{t}) = \mathsf{u}^{\mathsf{t}} \cdot \mathsf{h}$.
  4. The public key is $\mathsf{PK} = (\mathsf{g}, \mathsf{h}, \mathsf{u}, \mathsf{v}, f)$ and the secret key is $\mathsf{SK} = \mathsf{g}^{\mathsf{ab}}$.
- $\mathsf{Y.Enc}(\mathsf{PK}, \mathsf{tag}, \mathsf{m})$:
  1. Choose $\mathsf{s} \in Z_p$ randomly.
  2. Compute $\mathsf{CT}_0 = \mathsf{m} \cdot \mathsf{v}^{\mathsf{s}}$, $\mathsf{CT}_1 = \mathsf{g}^{\mathsf{s}}$, $\mathsf{CT}_2 = f(\mathsf{tag})^{\mathsf{s}}$.
  3. Output $\mathsf{CT} = (\mathsf{CT}_0, \mathsf{CT}_1, \mathsf{CT}_2)$.
- $\mathsf{Y.Dec}(\mathsf{SK}, \mathsf{tag}, \mathsf{CT})$:
  1. Choose $\mathsf{r} \in Z_p$ randomly.
  2. Compute $\mathsf{K}_1 = \mathsf{SK} \cdot f(\mathsf{tag})^{\mathsf{r}}$ and $\mathsf{K}_2 = \mathsf{g}^{\mathsf{r}}$.
  3. Output $\mathsf{m} = \mathsf{CT}_0 \cdot e(\mathsf{K}_2, \mathsf{CT}_2) \cdot e(\mathsf{K}_1, \mathsf{CT}_1)^{-1}$.

- $\mathsf{Y.Setup\text{-}Alt}(1^\lambda, \mathsf{tag}^*, \mathsf{m}^*)$:

  1. Choose $\mathsf{g} \in \mathsf{G}$ and $(\mathsf{a}, \mathsf{b}, \mathsf{y}) \in Z_p$ randomly.
  2. Set $\mathsf{u} = \mathsf{g}^{\mathsf{a}}, v = e(\mathsf{g}, \mathsf{g})^{\mathsf{ab}}$.
  3. Set $\mathsf{h} = \mathsf{u}^{-\mathsf{tag}^*} \cdot \mathsf{g}^{\mathsf{y}}$.
  4. Define a function $f : Z_p \to \mathsf{G}$ such that for any value $\mathsf{t} \in Z_p$, $f(\mathsf{t}) = \mathsf{u}^{\mathsf{t}} \cdot \mathsf{h}$.
  5. The public key is $\mathsf{PK} = (\mathsf{g}, \mathsf{h}, \mathsf{u}, \mathsf{v}, f)$, the secret key is $\mathsf{SK} = \mathsf{g}^{\mathsf{ab}}$ and the alternate secret key is $\mathsf{SK\text{-}Alt} = (\mathsf{g}^{\mathsf{b}}, \mathsf{y}, \mathsf{tag}^*)$.
  6. The challenge ciphertext CT^* is computed by running the algorithm $\mathsf{Y.Enc}$.

- $\mathsf{Y.Setup\text{-}Alt\text{-}1}(1^\lambda, \mathsf{tag}^*, \mathsf{m}^*)$:

  1. Choose $\mathsf{g} \in \mathsf{G}$ and $(\mathsf{a}, \mathsf{b}, \mathsf{y}) \in Z_p$ randomly.
  2. Set $\mathsf{u} = \mathsf{g}^{\mathsf{a}}, v = e(\mathsf{g}, \mathsf{g})^{\mathsf{ab}}$.
  3. Set $\mathsf{h} = \mathsf{u}^{-\mathsf{tag}^*} \cdot \mathsf{g}^{\mathsf{y}}$.
  4. Define a function $f : Z_p \to \mathsf{G}$ such that for any value $\mathsf{t} \in Z_p$, $f(\mathsf{t}) = \mathsf{u}^{\mathsf{t}} \cdot \mathsf{h}$.
  5. The public key is $\mathsf{PK} = (\mathsf{g}, \mathsf{h}, \mathsf{u}, \mathsf{v}, f)$, the secret key is $\mathsf{SK} = \mathsf{g}^{\mathsf{ab}}$ and the alternate secret key is $\mathsf{SK\text{-}Alt} = (\mathsf{g}^{\mathsf{b}}, \mathsf{y}, \mathsf{tag}^*)$.
  6. The challenge ciphertext CT^* is computed as follows: pick $\mathsf{s} \in Z_p$ randomly. Choose an element $\mathsf{R} \in \mathsf{G}_1$ randomly. Compute $\mathsf{CT}^*_0 = \mathsf{m}^* \cdot \mathsf{R}$, $\mathsf{CT}^*_1 = \mathsf{g}^{\mathsf{s}}, \mathsf{CT}^*_2 = \mathsf{g}^{\mathsf{cy}}$. The challenge ciphertext is $\mathsf{CT}^* = (\mathsf{CT}^*_0, \mathsf{CT}^*_1, \mathsf{CT}^*_2)$.

- $\mathsf{Y.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{tag}, \mathsf{CT})$:

  1. Choose $\mathsf{r}_1 \in Z_p$ randomly. Set $\mathsf{r}_2 = \frac{-\mathsf{y}}{\mathsf{tag}-\mathsf{tag}^*} + \mathsf{r}_1$.
  2. Compute $\mathsf{K}_1 = (\mathsf{g}^{\mathsf{b}})^{\mathsf{r}_2}$ and $\mathsf{K}_2 = g^{\mathsf{r}_2}$.
  3. Output $\mathsf{m} = \mathsf{CT}_0 \cdot e(\mathsf{K}_2, \mathsf{CT}_2) \cdot e(\mathsf{K}_1, \mathsf{CT}_1)^{-1}$.

We now prove that the scheme Y satisfies all the properties of a puncturable tag based encryption.

1. **Equivalent on all but challenge tag:**
   This property follows from the correctness of the challenger's responses in the IBE security game with the adversary, on all but the challenge identity. This is elaborated in the proof of security in [CHK04]. At a high level, for every key query made by the adversary in the IBE security game, the challenger generates the key using $\mathsf{SK\text{-}Alt}$ using which the adversary can decrypt any ciphertext. In order for IBE security to hold, all these decryptions must also be correct.

2. **Indistinguishability of parameters:**
   The only difference in the two distributions is the way in which the public parameter h is generated. In both cases, it is randomly chosen (notice that the random value g^{y} acts as a mask in the algorithm $\mathsf{Y.Setup\text{-}Alt}$). Therefore, the two distributions are identical and hence the property easily follows.

3. **Indistinguishability of alternate setups:**
   This property follows from the Bilinear DDH assumption. At a high level, given a BDDH tuple $(\mathsf{g}^{\mathsf{a}}, \mathsf{g}^{\mathsf{b}}, \mathsf{g}^{\mathsf{c}}, \mathsf{T})$ where either $\mathsf{T} = e(\mathsf{g}, \mathsf{g})^{\mathsf{abc}}$ or is uniformly random, the challenger in this

game can do the following: use $\mathsf{g}^{\mathsf{a}}, \mathsf{g}^{\mathsf{b}}$ from the tuple as is. In the challenge ciphertext, set $\mathsf{CT}_0^* = \mathsf{m}^*\mathsf{T}$, $\mathsf{CT}_1^* = \mathsf{g}^{\mathsf{c}}$ and $\mathsf{CT}_2^* = \mathsf{g}^{\mathsf{cy}}$. Now, if $\mathsf{T} = e(\mathsf{g},\mathsf{g})^{\mathsf{abc}}$, this corresponds to the distribution output by Y.Setup-Alt while if T is random, this corresponds to the distribution output by Y.Setup-Alt-1. In more detail, this is very similar to the proof of security in [CHK04] where they show that the adversary can't guess the bit used in the challenge ciphertext without breaking the BDDH assumption.

4. **Indistinguishability of messages:**
Since the message m is masked by a uniformly random value R in the ciphertext, even an unbounded adversary can't win this game (defined in this property) except with negligible probability. Hence, the property holds.

D.1.2 [CHK04] using any HIBE scheme

In this section, we show that when instantiated with any selective-id secure HIBE scheme, Y is a puncturable tag based encryption scheme.

Notation: Let the security parameter be n. Let HIBE = (HIBE.Setup, HIBE.Enc, HIBE.KeyGen, HIBE.Dec) be any selective-id seucre hierarchical identity-based encryption scheme. Let the identity space be the same as the space of all tags for the scheme Y. In particular, let's say each tag is of size n bits. Similarly, the messages spaces are also the same. The scheme Y = (Y.Setup, Y.Enc, Y.Dec, Y.Setup-Alt, Y.Setup is as follows:

Construction:

- $\mathsf{Y.Setup}(1^\lambda)$:
  1. Generate $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{HIBE.Setup}(1^n)$.
- $\mathsf{Y.Enc}(\mathsf{PK},\mathsf{tag},\mathsf{m})$:
  1. Output $\mathsf{CT} = \mathsf{HIBE.Enc}(\mathsf{PK},\mathsf{tag},\mathsf{m})$. That is, set the identity as tag.
- $\mathsf{Y.Dec}(\mathsf{SK},\mathsf{tag},\mathsf{CT})$:
  1. Let's denote tag by the n-bit string $(\mathsf{b}_1\mathsf{b}_2\ldots\mathsf{b}_n)$.
  2. Denote $\mathsf{tag}_i = (\mathsf{b}_1\ldots\mathsf{b}_i)$.
  3. Compute the private key for identity tag. That is, for $i = 1,\ldots,n$, compute $\mathsf{SK}_i = \mathsf{HIBE.KeyGen}(\mathsf{SK}_{i-1},\mathsf{tag}_i)$. Here $\mathsf{SK}_0 = \mathsf{SK}$.
  4. Set $\mathsf{SK}_{\mathsf{tag}} = \mathsf{SK}_n$.
  5. Output $\mathsf{m} = \mathsf{HIBE.Dec}(\mathsf{SK}_{\mathsf{tag}},\mathsf{CT})$.
- $\mathsf{Y.Setup\text{-}Alt}(1^\lambda,\mathsf{tag}^*,\mathsf{m}^*)$:
  1. Generate $(\mathsf{PK},\mathsf{SK}) \leftarrow \mathsf{HIBE.Setup}(1^n)$.
  2. Let's denote tag^* by the n-bit string $(\mathsf{b}_1^*\mathsf{b}_2^*\ldots\mathsf{b}_n^*)$.
  3. Let $\mathsf{SK}_{\mathsf{id}}$ denote the private key for identity id.
  4. Compute $\mathsf{SK\text{-}Alt} = \{\mathsf{tag}^*, \mathsf{SK}_{\mathsf{b}_1^*}, \mathsf{SK}_{\mathsf{b}_1^*\overline{\mathsf{b}_2^*}}, \mathsf{SK}_{\mathsf{b}_1^*\mathsf{b}_2^*\overline{\mathsf{b}_3^*}}, \ldots, \mathsf{SK}_{\mathsf{b}_1^*\mathsf{b}_2^*\ldots\mathsf{b}_{n-1}^*\overline{\mathsf{b}_n^*}}\}$.
  5. Compute the challenge ciphertext as $\mathsf{CT}^* = \mathsf{HIBE.Enc}(\mathsf{PK},\mathsf{tag}^*,\mathsf{m}^*)$.

- $\mathsf{Y.Setup\text{-}Alt\text{-}1}(1^\lambda, \mathsf{tag}^*, \mathsf{m}^*)$:
  1. This algorithm is same as the previous one - $\mathsf{Y.Setup\text{-}Alt}(1^\lambda, \mathsf{tag}^*, \mathsf{m}^*)$.
- $\mathsf{Y.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{tag}, \mathsf{CT})$:
  1. Recall that $\mathsf{SK\text{-}Alt} = \{\mathsf{tag}^*, \mathsf{SK}_{\mathsf{b}_1^*}, \mathsf{SK}_{\mathsf{b}_1^*\overline{\mathsf{b}_2^*}}, \mathsf{SK}_{\mathsf{b}_1^*\mathsf{b}_2^*\overline{\mathsf{b}_3^*}}, \ldots, \mathsf{SK}_{\mathsf{b}_1^*\mathsf{b}_2^*\ldots\mathsf{b}_{n-1}^*\overline{\mathsf{b}_n^*}}\}$.
  2. Let's denote tag by the n-bit string $(\mathsf{c}_1\mathsf{c}_2\ldots\mathsf{c}_n)$. We know that $\mathsf{tag} \neq \mathsf{tag}^*$.
  3. Also, denote $\mathsf{tag}_i = (\mathsf{c}_1\ldots\mathsf{c}_i)$.
  4. Find the largest j such that $(\mathsf{c}_1\ldots\mathsf{c}_j) = (\mathsf{b}_1^*\ldots\mathsf{b}_{j-1}^*\overline{\mathsf{b}_j^*})$.
  5. Compute the private key for identity tag as follows: That is, for $i = (j+1), \ldots, n$, compute $\mathsf{SK}_i = \mathsf{HIBE.KeyGen}(\mathsf{SK}_{i-1}, \mathsf{tag}_i)$. Here $\mathsf{SK}_j = \mathsf{SK}_{\mathsf{b}_1^*\mathsf{b}_2^*\ldots\mathsf{b}_{j-1}^*\overline{\mathsf{b}_j^*}}$.
  6. Set $\mathsf{SK}_{\mathsf{tag}} = \mathsf{SK}_n$.
  7. Output $\mathsf{m} = \mathsf{HIBE.Dec}(\mathsf{SK}_{\mathsf{tag}}, \mathsf{CT})$.

We now prove that the scheme Y satisfies all the properties of a puncturable tag based encryption.

1. **Equivalent on all but challenge tag:**
   This property follows from the correctness of the HIBE scheme. That is, for all identities except the challenge identity, the two algorithms $\mathsf{Y.Dec}$ and $\mathsf{Y.Dec\text{-}Alt}$ in fact do the exact same operations.
2. **Indistinguishability of parameters:**
   The values $(\mathsf{PK}, \mathsf{SK}, \mathsf{CT}^*)$ are generated the same way in both cases. Therefore, the property is trivially true.
3. **Indistinguishability of alternate setups:**
   This is trivially true as both alternate setup algorithms are the same.
4. **Indistinguishability of messages:**
   This property follows from the selective-id security of the underlying HIBE scheme. That is, the adversary is essentially getting the private key for all identities except the challenge identity and should not be able to distinguish between the two encryptions. Therefore, if the adversary can break this property, then we can break the security of the HIBE scheme which would be a contradiction.

D.2 Scheme in [PW08]

Let X denote the CCA secure encryption scheme constructed by Peikert and Waters [PW08]. X is constructed by adding one time signatures to the below scheme Y (based on lossy trapdoor functions) exactly the way we convert any puncturable tag based encryption into a Special-CCA secure encryption scheme in Section 7.2. Therefore, in order to prove that X is a Special-CCA secure encryption scheme, it is enough to prove that Y is a puncturable tag based encryption scheme. However, note that the scheme Y implicitly described in the work of [PW08] has only three algorithms - $\mathsf{Y.Setup}, \mathsf{Y.Enc}, \mathsf{Y.Dec}$. Therefore, in order to prove that it is a puncturable tag based encryption, we have to design three more algorithms $\mathsf{Y.Setup\text{-}Alt}, \mathsf{Y.Setup\text{-}Alt\text{-}1}, \mathsf{Y.Dec\text{-}Alt}$ that together satisfy the required properties.

Notation: Let the security parameter be λ. We refer the reader to [PW08] for the definition of lossy trapdoor functions and ABO trapdoor functions. Let $(\mathsf{S}_{\mathsf{ltdf}}, \mathsf{F}_{\mathsf{ltdf}}, \mathsf{F}^{-1}_{\mathsf{ltdf}})$ give a collection of (n, k)-lossy trapdoor functions. Recall from the definition that this implicitly gives us two algorithms $\mathsf{S}_{\mathsf{inj}}(\cdot) = \mathsf{S}_{\mathsf{ltdf}}(\cdot, 0)$ and $\mathsf{S}_{\mathsf{loss}}(\cdot) = \mathsf{S}_{\mathsf{ltdf}}(\cdot, 1)$. let $(\mathsf{S}_{\mathsf{abo}}, \mathsf{G}_{\mathsf{abo}}, \mathsf{G}^{-1}_{\mathsf{abo}})$ give a collection of $(\mathsf{n}, \mathsf{k}')$-ABO trapdoor functions having branches $\mathsf{B}_\lambda = \{0,1\}^{\mathsf{v}}$ (which contains the space of signature verification keys which is also the space of tags we use). We require that the total lossiness $\mathsf{k} + \mathsf{k}' \geq (\mathsf{n} + \kappa)$ for some $\kappa = \kappa(\mathsf{n}) = \omega(\log \mathsf{n})$. Let $\mathcal{H}$ be a family of pairwise independent hash functions from $\{0,1\}^{\mathsf{n}}$ to $\{0,1\}^l$, where $l \leq \kappa - 2\log(1/\epsilon)$ for some negligible $\epsilon = \mathsf{negligible}(\lambda)$. The message space is $\{0,1\}^l$.

The scheme $\mathsf{Y} = (\mathsf{Y.Setup}, \mathsf{Y.Enc}, \mathsf{Y.Dec}, \mathsf{Y.Setup\text{-}Alt}, \mathsf{Y.Setup\text{-}Alt\text{-}1}, \mathsf{Y.Dec\text{-}Alt})$ is as follows:

Construction:

- $\mathsf{Y.Setup}(1^\lambda)$:
  1. First, generate an injective trapdoor function: $(s, t) \leftarrow \mathsf{S}_{\mathsf{inj}}(1^\lambda)$.
  2. Then, generate an ABO trapdoor function having lossy branch 0^{v}: $(s', t') \leftarrow \mathsf{S}_{\mathsf{abo}}(1^\lambda, 0^{\mathsf{v}})$.
  3. Finally, choose a hash function $h \leftarrow \mathcal{H}$.
  4. The public key is $\mathsf{PK} = (s, s', h)$ and the secret key is $\mathsf{SK} = (t, \mathsf{PK})$.
- $\mathsf{Y.Enc}(\mathsf{PK}, \mathsf{tag}, \mathsf{m})$:
  1. Choose $x \in \{0,1\}^n$ uniformly at random.
  2. Compute $\mathsf{CT}_1 = \mathsf{F}_{\mathsf{ltdf}}(s, x)$, $\mathsf{CT}_2 = \mathsf{G}_{\mathsf{abo}}(s', \mathsf{tag}, x)$ and $\mathsf{CT}_3 = \mathsf{m} \oplus h(x)$.
  3. Output $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2, \mathsf{CT}_3)$.
- $\mathsf{Y.Dec}(\mathsf{SK}, \mathsf{tag}, \mathsf{CT})$:
  1. Compute $x = \mathsf{F}^{-1}_{\mathsf{ltdf}}(t, \mathsf{CT}_1)$.
  2. Check that $\mathsf{CT}_1 = \mathsf{F}_{\mathsf{ltdf}}(s, x)$ and $\mathsf{CT}_2 = \mathsf{G}_{\mathsf{abo}}(s, \mathsf{tag}, x)$. If not true, output $\perp$.
  3. Else, output $\mathsf{m} = \mathsf{CT}_3 \oplus h(x)$.
- $\mathsf{Y.Setup\text{-}Alt}(1^\lambda, \mathsf{tag}^*, \mathsf{m}^*)$:
  1. First, generate an injective trapdoor function: $(s, t) \leftarrow \mathsf{S}_{\mathsf{inj}}(1^\lambda)$.
  2. Then, generate an ABO trapdoor function having lossy branch tag^*: $(s', t') \leftarrow \mathsf{S}_{\mathsf{abo}}(1^\lambda, \mathsf{tag}^*)$.
  3. Finally, choose a hash function $h \leftarrow \mathcal{H}$.
  4. The public key is $\mathsf{PK} = (s, s', h)$, the secret key is $\mathsf{SK} = (t, \mathsf{PK})$, the alternate secret key is $\mathsf{SK\text{-}Alt} = (t', \mathsf{PK})$.
  5. Compute the challenge ciphertext CT^* by running the algorithm $\mathsf{Y.Enc}$.
  6. Output $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$.
- $\mathsf{Y.Setup\text{-}Alt\text{-}1}(1^\lambda, \mathsf{tag}^*, \mathsf{m}^*)$:
  1. First, generate a lossy trapdoor function: $(s, \perp) \leftarrow \mathsf{S}_{\mathsf{loss}}(1^\lambda)$.
  2. Then, generate an ABO trapdoor function having lossy branch tag^*: $(s, t) \leftarrow \mathsf{S}_{\mathsf{abo}}(1^\lambda, \mathsf{tag}^*)$.
  3. Finally, choose a hash function $h \leftarrow \mathcal{H}$.

4. The public key is $\mathsf{PK} = (s, s', h)$, the secret key is $\mathsf{SK} = (\perp, \mathsf{PK})$, the alternate secret key is $\mathsf{SK\text{-}Alt} = (t', \mathsf{PK})$.
5. Compute the challenge ciphertext CT^* by running the algorithm $\mathsf{Y.Enc}$.
6. Output $(\mathsf{PK}, \mathsf{SK}, \mathsf{SK\text{-}Alt}, \mathsf{CT}^*)$.

- $\mathsf{Y.Dec\text{-}Alt}(\mathsf{SK\text{-}Alt}, \mathsf{tag}, \mathsf{CT})$:
  1. Compute $x = \mathsf{G}_{\mathsf{abo}}^{-1}(t, \mathsf{tag}, \mathsf{CT}_2)$.
  2. Check that $\mathsf{CT}_1 = \mathsf{F}_{\mathsf{ltdf}}(s, x)$ and $\mathsf{CT}_2 = \mathsf{G}_{\mathsf{abo}}(s, \mathsf{tag}, x)$. If not true, output $\perp$.
  3. Else, output $\mathsf{m} = \mathsf{CT}_3 \oplus h(x)$.

Correctness of the scheme follows directly from the correctness of the underlying primitives. We refer the reader to [PW08] for a detailed description.

We now prove that the scheme Y satisfies all the properties of a puncturable tag based encryption.

1. **Equivalent on all but challenge tag:**
   This property follows from the proof of Claim 4.5 on page 21 of of [PW07]. [PW07] is the Eprint version of the paper and the scheme is the same. Briefly, the two decryptions are always equal if the lossy and ABO collections are both perfect.

2. **Indistinguishability of parameters:**
   This property follows from the proof of Claim 4.4 on page 21 of [PW07]. Briefly, it follows from the hidden lossy branch property that the ABO trapdoor function returns computationally indistinguishable outputs.

3. **Indistinguishability of alternate setups:**
   This property follows from the proof of Claim 4.6 on page 22 of [PW07]. Briefly, it follows from the indistinguishability of the lossy and injective functions of the lossy TDF collection.

4. **Indistinguishability of messages:**
   This property follows from the proof of Claim 4.7 on page 22 of [PW07]. In fact, the two distributions can't be distinguished even by an unbounded adversary. Briefly, this follows from an information theoretic argument due to the "lossiness" of the underlying functions.

Remark: Note that here we crucially use the fact that we have 2 alternate setup algorithms that allow us to switch from using the original secret key to an alternate punctured secret key - that is, in the first algorithm $\mathsf{PTBE.Setup\text{-}Alt}$, we can puncture the ABO trapdoor function on the branch tag^* while still maintaining the same secret key t generating using the lossy trapdoor function to guarantee the indistinguishability of parameters property. Then, in the next algorithm $\mathsf{PTBE.Setup\text{-}Alt\text{-}1}$, we can switch the lossy trapdoor function from being in the injective mode to lossy mode and here, we no longer need the original secret key to prove any property.

E Key Only FE-Compatibility

In this section, we define a weaker notion of FE-compatibility for a public key encryption scheme that we call Key Only FE-Compatibility. Informally, this states that given a public key encryption scheme, we can retain only the setup algorithm and design new encryption, decryption and function secret key generation algorithms for the FE scheme. Additionally, we will also work in the CRS model. That is, there exists a one time universal setup algorithm $\mathsf{CRS.Setup}$ that generates a CRS which will be made part of the public key. Formally:

Definition 11. *Given a one time universal setup algorithm* $\mathsf{CRS.Setup}(\cdot)$ *that, on input the security parameter outputs a common random string* CRS*, a public key encryption scheme* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *is said to be selective/adaptive Key Only FE-Compatible relative to a family of functions* $\mathcal{F}$ *if there exists three algorithms* $(\mathsf{FE.Enc}, \mathsf{FE.Keygen}, \mathsf{FE.Dec})$ *such that* $(\mathsf{FE.Setup}, \mathsf{FE.Enc}, \mathsf{FE.Keygen}, \mathsf{FE.Dec})$ *is a selectively/adaptively secure functional encryption scheme for the family* $\mathcal{F}$ *where:*

- *If* $\mathsf{PKE.Setup}(n)$ *outputs* $(\mathsf{PK}, \mathsf{SK})$*, the output of* $\mathsf{FE.Setup}(n)$ *is* $\mathsf{MPK} = (\mathsf{PK}, \mathsf{CRS})$ *and* $\mathsf{MSK} = (\mathsf{SK}, \mathsf{CRS})$*.*

Note that unlike the case of FE-Compatibility, here, the resulting FE scheme may not be compact.

E.1 Construction from iO

In this section, we show that every public key encryption scheme is Key Only FE-Compatible. That is, we construct a one time setup algorithm $\mathsf{CRS.Setup}$ relative to which, every public key encryption scheme can be upgraded into a functional encryption scheme. To achieve this, we use the construction of functional encryption from indistinguishability obfuscation given by Garg et al. [GGH+13]. Formally, we show that :

Theorem 9. *In the common random string model, every public key encryption scheme is selective Key Only FE-Compatible for any function family* $\mathcal{F}_n$ *and* $poly(n)$ *function key queries assuming the existence of polynomially secure versions of the following:*

1. *Indistinguishability obfuscation,*
2. *Public key encryption and*
3. *Statistically simulation sound non-interactive zero knowledge proofs (SSS-NIZKs)*

Notation: Let n be the security parameter. Let $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ be any public key encryption scheme that encrypts messages of length $p(n)$. Let $\mathsf{CPA} = (\mathsf{Setup_{CPA}}, \mathsf{Enc_{CPA}}, \mathsf{Dec_{CPA}})$ be a fixed public key encryption scheme that also encrypts messages of length $p(n)$. Let $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ be a SSS-NIZK system. The construction is as follows:

- $\mathsf{CRS.Setup}(1^n)$:
  1. Compute $(\mathsf{PK_{CPA}}, \mathsf{SK_{CPA}}) \leftarrow \mathsf{Setup_{CPA}}(1^n)$.
  2. Compute $\mathsf{CRS_{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^n)$.
  3. Output $\mathsf{CRS} = (\mathsf{PK_{CPA}}, \mathsf{CRS_{NIZK}})$.
- $\mathsf{FE.Setup}(1^n)$: Compute $(\mathsf{PK}, \mathsf{SK}) \leftarrow \mathsf{PKE.Setup}(1^n)$ and output $\mathsf{MPK} = (\mathsf{PK}, \mathsf{CRS})$ and $\mathsf{MSK} = (\mathsf{SK}, \mathsf{CRS})$.
- $\mathsf{FE.Enc}(\mathsf{m}, \mathsf{MPK})$:
  1. Parse $\mathsf{MPK} = (\mathsf{PK}, \mathsf{PK_{CPA}}, \mathsf{CRS_{NIZK}})$.
  2. Compute $\mathsf{CT}_1 = \mathsf{PKE.Enc}(\mathsf{m}, \mathsf{PK}; \mathsf{r}_1)$ and $\mathsf{CT}_2 = \mathsf{Enc_{CPA}}(\mathsf{m}, \mathsf{PK_{CPA}}; \mathsf{r}_2)$ using randomness r_1 and r_2 respectively.

3. Using $\mathsf{CRS}_{\mathsf{NIZK}}$ and the algorithm $\mathsf{NIZK.Prove}$, compute a proof π for the statement $(\mathsf{CT}_1, \mathsf{CT}_2) \in L$ using witness $(\mathsf{m}, \mathsf{r}_1, \mathsf{r}_2)$ where the language L is defined by the following relation R:
   Statement: $st = (\mathsf{CT}_1, \mathsf{CT}_2)$
   Witness: $w = (\mathsf{m}, \mathsf{r}_1, \mathsf{r}_2)$
   $R(st, w) = 1$ if $\mathsf{CT}_1 = \mathsf{PKE.Enc}(\mathsf{m}, \mathsf{PK}; \mathsf{r}_1)$ and $\mathsf{CT}_2 = \mathsf{Enc}_{\mathsf{CPA}}(\mathsf{m}, \mathsf{PK}_{\mathsf{CPA}}; \mathsf{r}_2)$.
4. Output $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2, \pi)$.

- $\mathsf{FE.Keygen}(\mathsf{MSK}, f)$: Output $\mathsf{SK}_f = \mathcal{O}(\mathsf{G}_f)$ where the program G_f is described below.
- $\mathsf{FE.Dec}(\mathsf{SK}_f, \mathsf{CT})$ Run the program SK_f on input CT to output a string y.

Program G_f

Input : ciphertext $\mathsf{CT} = (\mathsf{CT}_1, \mathsf{CT}_2, \pi)$
Constants : $\mathsf{SK}, \mathsf{CRS} = (\mathsf{PK}_{\mathsf{CPA}}, \mathsf{CRS}_{\mathsf{NIZK}})$

1. Check that π is a valid NIZK proof (using the algorithm $\mathsf{NIZK.Verify}$ and $\mathsf{CRS}_{\mathsf{NIZK}}$) for the statement $(\mathsf{CT}_1, \mathsf{CT}_2) \in L$.
2. Compute $\mathsf{m} = \mathsf{PKE.Dec}(\mathsf{SK}, \mathsf{CT}_1)$.
3. Output $\perp$ if the decryption aborts.
4. Else, output $f(\mathsf{m})$.

Figure 8: Program for generating function secret key

Security Proof: Notice that the scheme is identical to the one in [GGH+13] and the proof of security also follows directly.

Using just a Common Random String: Notice that in the above construction, the CRS we use is a common reference string. Let's see how to instead rely on just a common random string.

First, from the construction of SSS-NIZK in [GGH+13], observe that $\mathsf{CRS}_{\mathsf{NIZK}}$ consists of two parts - a random string that is the CRS of a NIZK system and a non-interactive commitment to 0. If we use a non-interactive commitment scheme that produces psuedo-random commitments, then $\mathsf{CRS}_{\mathsf{NIZK}}$ would be a uniformly random string.

The other component of the common reference string that we use is the public key $\mathsf{PK}_{\mathsf{CPA}}$. Assuming dense cryptosystems [SP92], the public key can be replaced by a uniformly random string. Thus, this gives us a common random string CRS.