Functional Encryption with Secure Key Leasing

Fuyuki Kitagawa [1] Ryo Nishimaki [1]

[1] NTT Social Informatics Laboratories, Tokyo, Japan
{fuyuki.kitagawa.yh,ryo.nishimaki.zk}@hco.ntt.co.jp

Abstract

Secure software leasing is a quantum cryptographic primitive that enables us to lease software to a user by encoding it into a quantum state. Secure software leasing has a mechanism that verifies whether a returned software is valid or not. The security notion guarantees that once a user returns a software in a valid form, the user no longer uses the software.

In this work, we introduce the notion of secret-key functional encryption (SKFE) with secure key leasing, where a decryption key can be securely leased in the sense of secure software leasing. We also instantiate it with standard cryptographic assumptions. More specifically, our contribution is as follows.

- We define the syntax and security definitions for SKFE with secure key leasing.
- We achieve a transformation from standard SKFE into SKFE with secure key leasing *without using additional assumptions*. Especially, we obtain bounded collusion-resistant SKFE for $\mathsf{P/poly}$ with secure key leasing based on post-quantum one-way functions since we can instantiate bounded collusion-resistant SKFE for $\mathsf{P/poly}$ with the assumption.

Some previous secure software leasing schemes capture only pirate software that runs on an *honest* evaluation algorithm (on a legitimate platform). However, our secure key leasing notion captures arbitrary attack strategies and does not have such a limitation.

As an additional contribution, we introduce the notion of single-decryptor FE (SDFE), where each functional decryption key is copy-protected. Since copy-protection is a stronger primitive than secure software leasing, this notion can be seen as a stronger cryptographic primitive than FE with secure key leasing. More specifically, our additional contribution is as follows.

- We define the syntax and security definitions for SDFE.
- We achieve collusion-resistant single-decryptor PKFE for $\mathsf{P/poly}$ from post-quantum indistinguishability obfuscation and quantum hardness of the learning with errors problem.

Keywords: Functional encryption, secure leasing, copy-protection, quantum cryptography

Contents

1 Introduction

1.1 Background

Functional encryption (FE) [BSW11] is an advanced encryption system that enables us to compute on encrypted data. In FE, an authority generates a master secret key and an encryption key. An encryptor uses the encryption key to generate a ciphertext ct_x of a plaintext x. The authority generates a functional decryption key fsk from a function f and the master secret key. When a decryptor receives fsk and ct_x, it can compute $f(x)$ and obtains nothing beyond $f(x)$. In secret-key FE (SKFE), the encryption key is the same as the master secret key, while the encryption key is public in public-key FE (PKFE).

FE offers flexible accessibility to encrypted data since multiple users can obtain various processed data via functional decryption keys. Public-key encryption (PKE) and attribute-based encryption (ABE) [SW05] do not have this property since they recover an entire plaintext if decryption succeeds. This flexible feature is suitable for analyzing sensitive data and computing new data from personal data without compromising data privacy. For example, we can compute medical statistics from patients' data without directly accessing individual data. Some works present practical applications of FE (for limited functionalities): non-interactive protocol for hidden-weight coin flips [CS19], biometric authentication, nearest-neighbor search on encrypted data [KLM+18], private inference on encrypted data [RPB+19].

One issue is that once a user obtains fsk, it can compute $f(x)$ from a ciphertext of x forever. An authority may not want to provide users with the permanent right to compute on encrypted data. A motivative example is as follows. A research group member receives a functional decryption key fsk to compute some statistics from many encrypted data for their research. When the member leaves the group, an authority wants to prevent the member from doing the same computation on another encrypted data due to terms and conditions. However, the member might *keep a copy* of their functional decryption key and penetrate the database of the group to do the same computation. Another motivation is that the subscription business model is common for many services such as cloud storage services (ex. OneDrive, Dropbox), video on demand (ex. Netflix, Hulu), software applications (ex. Office 365, Adobe Photoshop). If we can keep a copy of functional decryption keys, we cannot use FE in the subscription business model (for example, FE can be used as broadcast encryption in a video on demand). We can also consider the following subscription service. A company provides encrypted data sets for machine learning and a functional decryption key. A researcher can perform some tasks using the encrypted data set and the key.

Achieving a revocation mechanism [NP01] is an option to solve the issue above. Some works propose revocation mechanisms for advanced encryption such as ABE [SSW12] and FE [NWZ16]. However, revocation is not a perfect solution since we need to update ciphertexts to embed information about revoked users. We want to avoid updating ciphertexts for several reasons. One is a practical reason. We possibly handle a vast amount of data, and updating ciphertexts incurs significant overhead. Another one is more fundamental. Even if we update ciphertexts, *there is no guarantee that all old ciphertexts are appropriately deleted.* If some user keeps copies of old ciphertexts, and a data breach happens after revocation, another functional decryption key holder whose key was revoked still can decrypt the old ciphertexts.

This problem is rooted in classical computation since we cannot prevent copying digital data. Ananth and La Placa introduce the notion of secure software leasing [AL21] to solve the copy problem by using the power of quantum computation. Secure software leasing enables us to encode software into a leased version. The leased version has the same functionality as the original one and must be a quantum state to prevent copying. *After a lessor verifies that the returned software from a lessee is valid (or that the lessee deleted the software)*, the lessee cannot execute the software anymore. Several works present secure software leasing for simple functionalities such as a sub-class of evasive functions (subEVS), PKE, signatures, pseudorandom functions (PRFs) [AL21, ALL+21, KNY21, BJL+21, CMP20]. If we can securely implement leasing and returning mechanisms for functional decryption keys, we can solve the problem above. Such mechanisms help us to use FE in real-world applications.

Thus, the main question in this work is as follows.

Can we achieve secure a leasing mechanism for functional decryption keys of FE?

We can also consider copy-protection, which is stronger security than secure leasing. Aaronson [Aar09] introduces the notion of quantum copy-protection. Copy-protection prevents users from creating a pirate copy. *It does not have a returning process*, and prevents copying software. If a user returns the original software, no copy is left behind on the user, and it cannot run the software. Coladangelo, Liu, Liu, and Zhandry [CLLZ21] achieve copy-protected PRFs and single-decryptor encryption (SDE)[1]. Our second question in this work is as follows.

Can we achieve copy-protection for functional decryption keys of FE?

We affirmatively answer those questions in this work.

1.2 Our Result

Secure key leasing. Our main contributions are introducing the notion of SKFE with secure key leasing and instantiating it with standard cryptographic assumptions. More specifically,

- We define the syntax and security definitions for SKFE with secure key leasing.
- We achieve a transformation from standard SKFE into SKFE with secure key leasing *without using additional assumptions*.

In SKFE with secure key leasing, a functional decryption key is a quantum state. More specifically, the key generation algorithm takes as input a master secret key, a function f, and an availability bound n (in terms of the number of ciphertexts), and outputs a quantum decryption key fsk tied to f. We can generate a *certificate for deleting* the decryption key fsk. If the user of this decryption key deletes fsk within the declared availability bound n and the generated certificate is valid, the user cannot compute $f(x)$ from a ciphertext of x anymore. We provide a high-level overview of the security definition in Section 1.3.

We can obtain bounded collusion-resistant SKFE for P/poly with secure key leasing from OWFs since we can instantiate bounded collusion-resistant SKFE for P/poly with OWFs.[2] Note that all building blocks in this work are post-quantum secure since we use quantum computation and we omit "post-quantum".

Our secure key leasing notion is similar to but different from secure software leasing [AL21] for FE because adversaries in secure software leasing (for FE) must run their pirate software by an *honest* evaluation algorithm (on a legitimate platform). This is a severe limitation. In our FE with secure key leasing setting, adversaries do *not* necessarily run their pirate software (for functional decryption) by an honest evaluation algorithm and can take arbitrary attack strategies.

We develop a transformation from standard SKFE into SKFE with secure key leasing by using quantum power. In particular, we use (reusable) secret-key encryption (SKE) with certified deletion [BI20, HMNY21], where we can securely delete *ciphertexts*, as a building block. We also develop a technique based on the security bound amplification for FE [AJL$^+$19, JKMS20] to amplify the availability bound, that is, the number of encryption queries before ct^* is given. This technique deviates from known multi-party-computation-based techniques for achieving bounded many-ciphertext security for SKFE [GVW12, AV19].[3] The security bound amplification-based technique is of independent interest

[1]SDE is PKE whose decryption keys are copy-protected.

[2]If we start with fully collusion-resistant SKFE, we can obtain fully collusion-resistant SKFE with secure key leasing.

[3]These techniques [GVW12, AV19] work as transformations from single-key FE into bounded collusion-resistant FE. However, they also work as transformations from single-ciphertext SKFE into bounded many-ciphertext SKFE. See Section 1.4 for the detail. Many-ciphertext means that SKFE is secure even if adversaries can send unbounded polynomially many queries to an encryption oracle.

since the security bound amplification is not directly related to the amplification of the number of queries. These are the main technical contributions of this work. See Section 1.3 and main sections for more details.

Copy-protected functional decryption keys. The other contributions are copy-protected functional decryption keys. We introduce the notion of single-decryptor FE (SDFE), where each functional decryption key is copy-protected. This notion can be seen as a stronger cryptographic primitive than FE with secure key leasing, as we argued in Section 1.1.

- We define the syntax and security definitions for SDFE.
- We achieve collusion-resistant public key SDFE for P/poly from sub-exponentially secure indistinguishability obfuscation (IO) and the sub-exponential hardness of the learning with errors problem (QLWE assumption).

First, we transform single-key PKFE for P/poly into *single-key* SDFE for P/poly by using SDE. Then, we transform single-key SDFE P/poly into collusion-resistant SDFE for P/poly by using an IO-based key bundling technique [KNT21, BNPW20]. We can instantiate SDE with IO and the QLWE assumption [CLLZ21, CV21] and single-key PKFE for P/poly with PKE [SS10, GVW12].

1.3 Technical Overview

We provide a high-level overview of our techniques. Below, standard math font stands for classical algorithms and classical variables, and calligraphic font stands for quantum algorithms and quantum states.

Syntax of SKFE with secure key leasing. We first recall a standard SKFE scheme. It consists of four algorithms $(\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$. Setup is given a security parameter 1^λ and a collusion bound 1^q and generates a master secret key msk. Enc is given msk and a plaintext x and outputs a ciphertext ct. KG is given msk and a function f and outputs a decryption key fsk tied to f. Dec is given fsk and ct and outputs $f(x)$. Then, the indistinguishability-security of SKFE roughly states that any QPT adversary cannot distinguish encryptions of x_0 and x_1 under the existence of the encryption oracle and the key generation oracle. Here, the adversary can access the key generation oracle at most q times and can query only a function f such that $f(x_0) = f(x_1)$.

An SKFE scheme with secure key leasing (SKFE-SKL) is a tuple of six algorithms $(\mathsf{Setup}, \mathcal{KG}, \mathsf{Enc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$, where the first four algorithms form a standard SKFE scheme except the following difference on $\mathcal{KG}$. In addition to a function f, $\mathcal{KG}$ is given an availability bound 1^n in terms of the number of ciphertexts. Also, given those inputs, $\mathcal{KG}$ outputs a verification key vk together with a decryption key $\mathcal{fsk}$ tied to f encoded in a quantum state, as $(\mathcal{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f, 1^n)$. By using $\mathcal{Cert}$, we can generate a (classical) certificate that a quantum decryption key $\mathcal{fsk}$ is deleted, as $\mathsf{cert} \leftarrow \mathcal{Cert}(\mathcal{fsk})$. We check the validity of certificates by using vk and Vrfy, as $\top/\bot \leftarrow \mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$. In addition to the decryption correctness, an SKFE-SKL scheme is required to satisfy the verification correctness that states that a correctly generated certificate is accepted, that is, $\top = \mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$ for $(\mathcal{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f, 1^n)$ and $\mathsf{cert} \leftarrow \mathcal{Cert}(\mathcal{fsk})$.

Security of SKFE-SKL. The security notion of SKFE-SKL we call lessor security intuitively guarantees that if an adversary given $\mathcal{fsk}$ deletes it and the generated certificate is accepted within the declared availability bound, the adversary cannot use $\mathcal{fsk}$ any more. The following indistinguishability experiment formalizes this security notion. For simplicity, we focus on a selective setting where the challenge plaintext pair (x_0^*, x_1^*) and the collusion bound q are fixed outside of the security experiment in this overview.

1. Throughout the experiment, $\mathcal{A}$ can get access to the following oracles, where $L_{\mathcal{KG}}$ is a list that is initially empty.

   $O_{\mathsf{Enc}}(x)$**:** This is the standard encryption oracle that returns $\mathsf{Enc}(\mathsf{msk}, x)$ given x.

   $O_{\mathcal{KG}}(f, 1^n)$**:** This oracle takes as input a function f and an availability bound 1^n, generate $(\mathit{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f, 1^n)$, returns fsk to $\mathcal{A}$, and adds $(f, 1^n, \mathsf{vk}, \bot)$ to $L_{\mathcal{KG}}$. Differently from the standard SKFE, $\mathcal{A}$ can query a function f such that $f(x_0^*) \neq f(x_1^*)$. $\mathcal{A}$ can get access to the key generation oracle at most q times.

   $O_{\mathsf{Vrfy}}(f, \mathsf{cert})$**:** Also, $\mathcal{A}$ can get access to the verification oracle. Intuitively, this oracle checks that $\mathcal{A}$ deletes leased decryption keys correctly within the declared availability bounds. Given (f, cert), it finds an entry $(f, 1^n, \mathsf{vk}, M)$ from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) If $\top = \mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$ and the number of queries to O_{Enc} at this point is less than n, it returns $\top$ and updates the entry into $(f, 1^n, \mathsf{vk}, \top)$. Otherwise, it returns $\bot$.

2. When $\mathcal{A}$ requests the challenge ciphertext, the challenger checks if $\mathcal{A}$ has correctly deleted all leased decryption keys for functions f such that $f(x_0^*) \neq f(x_1^*)$. If so, the challenger gives the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{msk}, x_{\mathsf{coin}}^*)$ for random bit $\mathsf{coin} \leftarrow \{0,1\}$ to $\mathcal{A}$, and otherwise the challenger output 0. Hereafter, $\mathcal{A}$ is not allowed to send a function f such that $f(x_0^*) \neq f(x_1^*)$ to $O_{\mathcal{KG}}$.

3. $\mathcal{A}$ outputs a guess coin' of coin.

We say that the SKFE-SKL scheme is lessor secure if no QPT adversary can guess coin significantly better than random guessing. We see that if $\mathcal{A}$ can use a decryption key after once $\mathcal{A}$ deletes and the deletion certificate is accepted, $\mathcal{A}$ can detect coin with high probability since $\mathcal{A}$ can obtain a decryption key for f such that $f(x_0^*) \neq f(x_1^*)$. Thus, this security notion captures the above intuition. We see that *lessor security implies standard indistinguishability-security for SKFE.*

We basically work with the above indistinguishability based selective security for simplicity. In Appendix B, we also provide the definitions of adaptive security and simulation based security notions and general transformations to achieve those security notions from indistinguishability based selective security.

Dynamic availability bound vs. static availability bound. In SKFE-SKL, we can set the availability bound for each decryption key differently. We can also consider a weaker variant where we statically set the single availability bound applied to each decryption key at the setup algorithm. We call this variant SKFE with static bound secure key leasing (SKFE-sbSKL). In fact, by using a technique developed in the context of dynamic bounded collusion FE [AMVY21, GGLW21], we can generically transform SKFE-sbSKL into SKFE-SKL if the underlying SKFE-sbSKL satisfies some additional security property and efficiency requirement. For the overview of this transformation, see Section 3.2. Therefore, we below focus on how to achieve SKFE-sbSKL. For simplicity, we ignore those additional properties required for the transformation to SKFE-SKL.

SKFE-sbSKL with the availability bound 0 from certified deletion. We start with a simple construction of an SKFE-sbSKL scheme secure for the availability bound 0 based on an SKE scheme with certified deletion [BI20, HMNY21]. The availability bound is 0 means that it is secure if an adversary deletes decryption keys without seeing any ciphertext.

SKE with certified deletion consists of five algorithms $(\mathsf{KG}, \mathcal{Enc}, \mathcal{Dec}, \mathcal{Del}, \mathsf{Vrfy})$. The first three algorithms form a standard SKE scheme except that $\mathcal{Enc}$ output a verification key vk together with a ciphertext encoded in a quantum state ct. By using $\mathcal{Del}$, we can generate a (classical) certificate that ct is deleted. The certificate is verified using vk and Vrfy. In addition to the decryption correctness, it satisfies the verification correctness that guarantees that a correctly generated certificate is accepted. The security notion roughly states that once an adversary deletes a ciphertext ct and the generated certificate is accepted, the adversary cannot obtain any plaintext information encrypted inside ct, even if the adversary is given the secret key after the deletion.

We now construct an SKFE-sbSKL scheme zSKFE-sbSKL that is secure for the availability bound 0, based on a standard SKFE scheme $\mathsf{SKFE} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ and an SKE scheme with certified deletion $\mathsf{CDSKE} = (\mathsf{CD.KG}, \mathsf{CD}.\mathcal{Enc}, \mathsf{CD}.\mathcal{Dec}, \mathsf{CD}.\mathcal{Del}, \mathsf{CD.Vrfy})$. In the setup of zSKFE-sbSKL, we generate $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q)$ and $\mathsf{cd.sk} \leftarrow \mathsf{CD.KG}(1^\lambda)$, and the master secret key of zSKFE-sbSKL is set to $\mathsf{zmsk} = (\mathsf{msk}, \mathsf{cd.sk})$. To generate a decryption key for f, we generate a decryption key for f by SKFE as $\mathsf{fsk} \leftarrow \mathsf{KG}(\mathsf{msk}, f)$ and encrypt it by CDSKE as $(\mathsf{cd}.ct, \mathsf{vk}) \leftarrow \mathsf{CD}.\mathcal{Enc}(\mathsf{cd.sk}, \mathsf{fsk})$. The resulting decryption key is $zfsk := \mathsf{cd}.ct$ and the corresponding verification key is vk. To encrypt a plaintext x, we just encrypt it by SKFE as $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, x)$ and append $\mathsf{cd.sk}$ contained in zmsk, as $\mathsf{zct} := (\mathsf{ct}, \mathsf{cd.sk})$. To decrypt zct with $zfsk$, we first retrieve fsk from cd.ct and cd.sk, and compute $f(x) \leftarrow \mathsf{Dec}(\mathsf{fsk}, \mathsf{ct})$. The certificate generation and verification are simply defined as those of CDSKE since $zfsk$ is a ciphertext of CDSKE.

The security of zSKFE-sbSKL is easily analyzed. Let (x_0^*, x_1^*) be the challenge plaintext pair. When an adversary $\mathcal{A}$ queries f to $O_{\mathcal{KG}}$, $\mathcal{A}$ is given $zfsk := \mathsf{cd}.ct$, where $\mathsf{fsk} \leftarrow \mathsf{KG}(\mathsf{msk}, f)$ and $(\mathsf{cd}.ct, \mathsf{vk}) \leftarrow \mathsf{CD}.\mathcal{Enc}(\mathsf{cd.sk}, \mathsf{fsk})$. If $f(x_0^*) \neq f(x_1^*)$, $\mathcal{A}$ is required to delete $zfsk$ without seeing any ciphertext. This means that $\mathcal{A}$ cannot obtain cd.sk before $zfsk$ is deleted. Then, from the security of CDSKE, $\mathcal{A}$ cannot obtain any information of fsk. This implies that $\mathcal{A}$ can obtain a decryption key of SKFE only for a function f such that $f(x_0^*) = f(x_1^*)$, and thus the lessor security of zSKFE-sbSKL follows form the security of SKFE.

How to amplify the availability bound? We now explain how to amplify the availability bound from 0 to any polynomial n. One possible solution is to rely on the techniques for bounded collusion FE [GVW12, AV19]. Although the bounded collusion techniques can be used to amplify "1-bounded security" to "poly-bounded security", it is not clear how to use it starting from "0-bounded security". For more detailed discussion on this point, see Remark 3.7. Therefore, we use a different technique from the existing bounded collusion FE. At a high level, we reduce the task of amplifying the availability bound to the task of amplifying the security bound, which has been studied in the context of standard FE [AJL+19, JKMS20].

We observe that we can obtain an SKFE-sbSKL scheme with availability bound n for any n that is secure with only inverse polynomial probability by just using many instances of zSKFE-sbSKL in parallel. Concretely, suppose we use $N = \alpha n$ instances of zSKFE-sbSKL to achieve a scheme with availability bound n, where $\alpha \in \mathbb{N}$. To generate a decryption key for f, we generate $(zfsk_j, \mathsf{vk}_j) \leftarrow z\mathcal{KG}(\mathsf{zmsk}_j, f)$ for every $j \in [N]$, and set the resulting decryption key as $(zfsk_j)_{j\in[N]}$ and the corresponding verification key as $(\mathsf{vk}_j)_{j\in[N]}$. To encrypt x, we randomly choose $j \leftarrow [N]$, generate $\mathsf{zct}_j \leftarrow \mathsf{zEnc}(\mathsf{zmsk}_j, x)$, and set the resulting ciphertext as (j, zct_j). To decrypt this ciphertext with $(zfsk_j)_{j\in[N]}$, we just compute $f(x) \leftarrow z\mathcal{Dec}(zfsk_j, \mathsf{zct}_j)$. The certification generation and verification are done by performing them under all N instances. The security of this construction is analyzed as follows. The probability that the j^* chosen when generating the challenge ciphertext collides with some of n indices $j_1, \cdots, j_n$ used by the first n calls of the encryption oracle, is at most $n/N = 1/\alpha$. If such a collision does not happen, we can use the security of j^*-th instance of zSKFE-sbSKL to prove the security of this construction. Therefore, this construction is secure with probability roughly $1 - 1/\alpha$ (denoted by $1/\alpha$-secure scheme).

Thus, all we have to do is to convert an SKFE-sbSKL scheme with inverse polynomial security into one with negligible security. As stated above, such security amplification has been studied for standard FE. In this work, we adopt the amplification technique using homomorphic secret sharing (HSS) [AJL+19, JKMS20].

Amplification using HSS. In this overview, we describe our construction using HSS that requires the LWE assumption with super-polynomial modulus to give a high-level intuition. However, our actual construction uses a primitive called set homomorphic secret sharing (SetHSS) [JKMS20], which is a weak form of HSS and *can be based on OWFs*. [4] See Section 5 for our construction based on OWFs.

An HSS scheme consists of three algorithms $(\mathsf{InpEncode}, \mathsf{FuncEncode}, \mathsf{Decode})$. $\mathsf{InpEncode}$ is given a security parameter 1^λ, a number 1^m, and an input x, and outputs m input shares $(s_i)_{i\in[m]}$. $\mathsf{FuncEncode}$ is given a security parameter 1^λ, a number 1^m, and a function f, and outputs m function shares $(f_i)_{i\in[m]}$. Decode takes a set of evaluations of function shares on their respective input shares $(f_i(s_i))_{i\in[m]}$, and outputs a value $f(x)$. Then, the security property of an HSS scheme roughly guarantees that for any (i^*, x_0^*, x_1^*), given a set of input shares $(s_i)_{i\in[m]\setminus\{i^*\}}$ for some i^*, an adversary cannot detect from which of the challenge inputs they are generated, under the existence of function encode oracle that is given f such that $f(x_0^*) = f(x_1^*)$ and returns $(f_i(s_i))_{i\in[m]}$.

We describe SKFE-sbSKL scheme $\mathsf{SKFE\text{-}sbSKL}$ with the availability bound $n \geq 1$ of our choice using a HSS scheme $\mathsf{HSS} = (\mathsf{InpEncode}, \mathsf{FuncEncode}, \mathsf{Decode})$. In the setup of $\mathsf{SKFE\text{-}sbSKL}$, we first set up $1/2$-secure SKFE-sbSKL scheme $\mathsf{SKFE\text{-}sbSKL}'$ with the availability bound n. This is done by parallelizing $2n$ instances of $\mathsf{zSKFE\text{-}sbSKL}$ as explained before. We generate m master secret keys $\mathsf{msk}_1, \cdots, \mathsf{msk}_m$ of $\mathsf{SKFE\text{-}sbSKL}'$. Then, to generate a decryption key for f by $\mathsf{SKFE\text{-}sbSKL}$, we first generate $(f_i)_{i\in[m]} \leftarrow \mathsf{FuncEncode}(1^\lambda, 1^m, f)$, and generate a decryption key $\mathcal{fsk}_i$ tied to f_i under msk_i for each $i \in [m]$. To encrypt x by $\mathsf{SKFE\text{-}sbSKL}$, we first generate $(s_i)_{i\in[m]} \leftarrow \mathsf{InpEncode}(1^\lambda, 1^m, x)$ and generate a ciphertext ct_i of s_i under msk_i for each $i \in [m]$. The certification generation and verification are done by performing those of $\mathsf{SKFE\text{-}SKL}'$ for all of the m instances. When decrypting the ciphertext $(\mathsf{ct}_i)_{i\in[m]}$ by $(\mathcal{fsk}_i)_{i\in[m]}$, we can obtain $f_i(s_i)$ by decrypting ct_i with $\mathcal{fsk}_i$ for every $i \in [m]$. By combining $(f_i(s_i))_{i\in[m]}$ using Decode, we can obtain $f(x)$.

The lessor security of $\mathsf{SKFE\text{-}sbSKL}$ can be proved as follows. Each of m instances of $\mathsf{SKFE\text{-}sbSKL}'$ is secure independently with probability $1/2$. Thus, there is at least one secure instance without probability $1/2^m$, which is negligible by setting $m = \omega(\log \lambda)$. Suppose i^*-th instance is a secure instance. Let (x_0^*, x_1^*) be the challenge plaintext pair, and let $(s_i^*)_{i\in[m]} \leftarrow \mathsf{InpEncode}(1^\lambda, 1^m, x_{\mathsf{coin}}^*)$ for $\mathsf{coin} \leftarrow \{0,1\}$. In the security experiment, from the security of $\mathsf{SKFE\text{-}sbSKL}'$ under msk_{i^*}, an adversary cannot obtain the information of s_{i^*} except for its evaluation on function shares for a function f queried to $O_{\mathcal{KG}}$ that satisfies that $f(x_0^*) = f(x_1^*)$. Especially, from the security of $\mathsf{SKFE\text{-}sbSKL}'$ under msk_{i^*}, the adversary cannot obtain an evaluation of s_{i^*} on function shares for a function f such that $f(x_0^*) \neq f(x_1^*)$, though $\mathcal{A}$ can query such a function to $O_{\mathcal{KG}}$. Then, we see that the lessor security of $\mathsf{SKFE\text{-}sbSKL}$ can be reduced to the security of HSS. [5]

In the actual construction, we use SetHSS instead of HSS, as stated before. Also, in the main body, we abstract the parallelized $\mathsf{zSKFE\text{-}sbSKL}$ as index-based SKFE-sbSKL. This makes the security proof of our construction using SetHSS simple. Moreover, in the actual construction of an index-based SKFE-sbSKL, we bundle the parallelized instances of $\mathsf{zSKFE\text{-}sbSKL}$ using a PRF. This modification is necessary to achieve the efficiency required for the above transformation into SKFE-SKL.

Goyal et al. [CGO21] use a similar technique using HSS in a different setting (private simultaneous message protocols). However, their technique relies on the LWE assumption unlike ours.

Single decryptor PKFE. In this work, we also define the notion of single decryptor PKFE, which is PKFE whose functional decryption key is copy-protected. The definition is a natural extension of SDE (PKE with copy-protected decryption keys). An adversary $\mathcal{A}$ tries to copy a target functional decryption key $\mathcal{sk}_{f^*}$. More specifically, $\mathcal{A}$ is given $\mathcal{sk}_{f^*}$ and outputs two possibly entangled quantum distinguishers

[4]The definition of HSS provided below is not standard. We modify the definition to be close to SetHSS. Note that HSS defined below can be constructed from multi-key fully homomorphic encryption with simulatable partial decryption property [MW16].

[5]Actual construction and security proof needs to use a technique called the trojan method [ABSV15]. We ignore the issue here for simplicity.

$\mathcal{D}_1$ and $\mathcal{D}_2$ and two plaintexts (x_0, x_1) such that $f^*(x_0) \neq f^*(x_1)$. If $\mathcal{D}_1$ or $\mathcal{D}_2$ cannot distinguish a given ciphertext is encryption of x_0 or x_1, $\mathcal{sk}_{f^*}$ is copy-protected.[6] If both $\mathcal{D}_1$ and $\mathcal{D}_2$ have $\mathcal{sk}_{f^*}$, they can trivially distinguish the challenge ciphertext. Thus, the definition guarantees copy-protection security. We provide a collusion-resistant single-decryptor PKFE scheme, where adversaries obtain polynomially many functional decryption keys, based on IO.

We first show that a single-key single-decryptor PKFE can be constructed from a single-key standard PKFE scheme and SDE scheme. The construction is simple nested encryption. Namely, when encrypting a plaintext x, we first encrypt it by the standard PKFE scheme and then encrypt the ciphertext by the SDE scheme. The secret key of the SDE scheme is included in the functional decryption key of the resulting single-decryptor PKFE scheme. Although a PKFE functional decryption key can be copied, the SDE decryption key cannot be copied and adversaries cannot break the security of PKFE. This is because we need to run the SDE decryption algorithm before we run the PKFE decryption algorithm.

The security notion for SDE by Coladangelo et al. [CLLZ21] is not sufficient for our purpose since SDE plaintexts are ciphertexts of the standard PKFE in the construction. We need to extend the security notion for SDE to prove the security of this construction because we need to handle randomized messages (the PKFE encryption is a randomized algorithm). Roughly speaking, this new security notion guarantees that the security of SDE holds for plaintexts of the form $g(x;r)$, where g and x respectively are a function and an input chosen by an adversary and r is a random coin chosen by the experiment. We can observe that the SDE scheme proposed by Coladangelo et al. [CLLZ21] based on IO satisfies this security notion. Then, by setting g as the encryption circuit of the standard PKFE, the security of the single-key single-decryptor PKFE scheme above can be immediately reduced to the security of the SDE scheme. We also extend adversarial quantum decryptors, which try to output an entire plaintext, to adversarial quantum distinguishers, which try to guess a 1-bit coin used to generate a ciphertext. We need this extension to use SDE as a building block. It is easy to observe the SDE scheme by Coladangelo et al. [CLLZ21] is secure even against quantum distinguishers.

Once we obtain a single-key single-decryptor PKFE scheme, we can transform it into a collusion-resistant single-decryptor PKFE scheme by again using IO. This transformation is based on one from a single-key standard PKFE scheme into a collusion-resistant standard PKFE scheme [BNPW20, KNT21]. The idea is as follows. We need to generate a fresh instance of the single-key scheme above for *each random tag* and bundle (unbounded) polynomially many instances to achieve collusion-resistance. We use IO to bundle multiple instances of single-key SDFE. More specifically, a public key is an obfuscated circuit of the following setup circuit. The setup circuit takes a public tag τ as input, generates a key pair $(\mathsf{pk}_\tau, \mathsf{msk}_\tau)$ of the single-key SDFE scheme using PRF value $\mathsf{F}_\mathsf{K}(\tau)$ as randomness, and outputs only pk_τ. The master secret key is the PRF key K. We can generate a functional decryption key for f by choosing a random tag τ and generating a functional decryption key $\mathcal{sk}_{f,\tau}$ under msk_τ. A functional decryption key of our collusion-resistant scheme consists of $(\tau, \mathcal{sk}_{f,\tau})$. A ciphertext is an obfuscated circuit of the following encryption circuit, where a plaintext x is hardwired. The encryption circuit takes a public tag τ, generates pk_τ by using the public key explained above, and outputs a ciphertext of x under pk_τ. Due to this mechanism, only one functional decryption key $\mathcal{sk}_{f,\tau}$ under msk_τ is issued for each τ, but we can generate polynomially many functional decryption keys by using many tags. If we use a different tag τ', an independent key pair $(\mathsf{pk}_{\tau'}, \mathsf{msk}_{\tau'})$ is generated and it is useless for another instance under $(\mathsf{pk}_\tau, \mathsf{msk}_\tau)$. The IO security guarantees that the information about K (and msk_τ) is hidden.[7] Thus, we can reduce the collusion-resistance to the single-key security of the underlying single-decryptor PKFE. Note that we need to consider super-polynomially many hybrid games to complete the proof since the tag space size must be super-polynomial to treat *unbounded* polynomially many tags. This is the reason why

[6]In the security definition of SDE, quantum *decryptors* try to recover the entire plaintext [CLLZ21]. We extend the definition because quantum decryptors are not sufficient for using SDFE as a building block. See Section 7 for detail.

[7]We use puncturable PRFs and the puncturing technique here as the standard technique for cryptographic primitives based on IO [SW21].

we need the sub-exponential security for building blocks.

1.4 More on Related Work

Secure software leasing. Ananth and La Place [AL21] achieve secure software leasing for subEVS from the QLWE assumption and IO. Kitagawa, Nishimaki, and Yamakawa [KNY21] achieve secure software leasing for PRFs and subEVS only from the QLWE (that is, without IO). Broadbent, Jeffery, Lord, Podder, and Sundaram [BJL+21] achieve secure software leasing for subEVS without assumptions. Coladangelo, Majenz, and Poremba [CMP20] also achieve secure software leasing for subEVS in the quantum random oracle (QROM) model (without assumptions).

In secure software leasing, we force adversaries to run pirate software by an *honest* evaluation algorithm denoted by $\mathcal{Run}$ [AL21]. An honest evaluation algorithm verifies the format of software when software is executed, and adversaries cannot adopt arbitrary strategies for creating pirate software. Aaronson, Liu, Liu, Zhandry, and Zhang [ALL+21] introduce copy-detection, which is essentially the same notion as secure software leasing. Although a check algorithm verifies returned software, there is no honest evaluation algorithm, and adversaries can adopt arbitrary strategies in the copy-detection setting. Aaronson et al. [ALL+21] achieve copy-detection for PRFs, PKE, and signatures from IO and standard cryptographic assumptions. Those cryptographic functionalities are much weaker than FE. Moreover, we need IO to achieve security against adversaries that adopt any strategy to execute pirate software. Secure software leasing by Coladangelo et al. [CMP20] is also secure against such arbitrary adversaries, but their construction relies on QROM, and its functionality is severely limited.

Certified deletion. Broadbent and Islam [BI20] present the notion of quantum encryption with certified deletion. They achieve one-time SKE with certified deletion without assumptions. Hiroka, Morimae, Nishimaki, and Yamakawa [HMNY21] extend the notion to reusable SKE, PKE, and ABE with certified deletion, and instantiate them with standard SKE, PKE, and IO and OWFs, respectively. Our FE with secure key leasing can also be seen as certified deletion for functional decryption keys.

Copy-protection. Aaronson [Aar09] achieves copy-protection for arbitrary unlearnable Boolean functions *relative to a quantum oracle*, and presents two *heuristic* copy-protection schemes for point functions. Aaronson et al. [ALL+21] achieve quantum copy-protection for all unlearnable functions by using *classical oracles*. Georgiou and Zhandry [GZ20] present the notion of SDE and instantiate it with one-shot signatures [AGKZ20] and extractable witness encryption (WE) [GKP+13]. They also introduce the notion of broadcast encryption with unclonable decryption and splittable ABE, which can be seen as copy-protected variants of broadcast encryption and ABE, respectively. They instantiate splittable ABE with the same tools as those for SDE. They instantiate broadcast encryption with unclonable decryption with tokenized signatures [BS17], extractable WE, and collision-resistant hash functions. Coladangelo et al. [CLLZ21] achieve quantum copy-protection for PRFs and SDE from IO and OWFs, the QLWE assumption, and the strong monogamy-of-entanglement conjecture. Later, Culf and Vidick prove that the strong monogamy-of-entanglement conjecture is true without any assumption [CV21]. Those cryptographic functionalities are weaker than FE, as in the case of secure software leasing. Most of those works rely on magical oracles or knowledge-type assumptions such as the existence of extractable WE, which is implausible [GGHW17]. Only the constructions by Coladangelo et al. [CLLZ21, CV21] do not rely on such strong tools. However, note that there is no provably secure *post-quantum* IO from well-founded assumptions so far.[8]

Ben-David and Sattath [BS17] present the notion of tokenized signatures, where we can generate a delegated signing token from a signing key. A signing token enables a user to sign one and only one

[8]There are some candidates [BGMZ18, CHVW19, AP20, DQV+21]. The known IO constructions from well-founded assumptions [JLS21, JLS22] are vulnerable against quantum adversaries.

message. They instantiate it with virtual black-box obfuscation [BGI+12]. Amos, Georgiou, Kiayias, and Zhandry present the notion of one-shot signatures, where adversaries cannot generate two valid signatures for different two messages even under an adversarially generated verification key. They instantiate it with *classical oracles*, which can be seen as an idealized virtual black-box obfuscation. Those are copy-protected variants of signatures. Relationships between (copy-protected) FE and them are not known so far.

Functional encryption. FE has been extensively studied since Boneh, Sahai, and Waters [BSW11] formally defined its notion. Although there are many works on FE, we focus on FE for P/poly that are closely related to our work in this paper.

Sahai and Seyalioglu [SS10] present the first (selectively secure) single-key PKFE for P/poly from PKE. Gorbunov, Vaikuntanathan, and Wee [GVW12] present transformations from (adaptively secure) single-key PKFE for NC^1 into (adaptively secure) bounded collusion-resistant PKFE for P/poly by using pseudorandom generators (PRGs) in NC^1. The transformation also converts single-key and many-ciphertext SKFE for NC^1 into bounded collusion-resistant and many-ciphertext SKFE for P/poly. They also show that we can achieve adaptively secure single-key PKFE (resp. SKFE) for NC^1 from PKE (resp. OWFs). Ananth, Brakerski, Segev, and Vaikuntanathan [ABSV15] observe that we can invert the roles of the functions and plaintexts in SKFE by using the function-privacy of SKFE [BS18], and obtain collusion-resistant and bounded many-ciphertext SKFE for P/poly from OWFs by using the transformations. Later, Ananth and Vaikuntanathan [AV19] improve the assumptions (remove PRGs in NC^1) for the transformation and achieve the optimal ciphertext size (in the asymptotic sense). These transformations [GVW12, AV19] run N independent copies of a single-key FE scheme and encrypt the views of some N-party multi-party computation (MPC) protocol. Agrawal and Rosen [AR17] construct bounded collusion-resistant PKFE for NC^1 by using FE for inner-product [ABDP15, ALS16] and the homomorphic property of some encryption scheme [BV11].[9] Their construction supports arithmetic circuits. Agrawal, Maitra, Vempati, and Yamada [AMVY21] and Garg, Goyal, Lu, and Waters [GGLW21] concurrently and independently present the notion of dynamic bounded collusion-resistant FE and how to achieve it from identity-based encryption (IBE). In the dynamic bounded collusion setting, the setup algorithm of FE does not depend on the number of key queries, and only the encryption algorithm of FE depends on it.

Security amplification is transforming an ϵ-secure cryptographic scheme into a ν-secure one, where $\epsilon \in (0,1)$ is a constant and ν is a negligible function. Ananth, Jain, Lin, Matt, and Sahai [AJL+19] propose security amplification techniques for standard FE. Their techniques are based on multi-key homomorphic encryption, which can be instantiated with the LWE assumption. Jain, Korb, Manohar, and Sahai [JKMS20] also propose how to amplify the security of standard FE by using SetHSS, which can be instantiated with OWFs.

IO for P/poly is necessary and sufficient for collusion-resistant FE (for NC^1) up to sub-exponential security loss [GGH+16, BV18, AJ15, AJS15, KNT22]. Jain, Lin, and Sahai [JLS21, JLS22] achieve the first IO (and collusion-resistant FE for P/poly) from well-founded assumptions. However, they are not post-quantum secure since the constructions heavily rely on cryptographic bilinear maps.

Revocation mechanisms have been extensively studied, especially in the context of broadcast encryption [NP01, NNL01].[10] Nishimaki et al. [NWZ16] introduce revocable FE to achieve a trace-and-revoke system with flexible identities. They achieve the first revocable FE by using IO and OWFs. We need to manage a revocation list and update ciphertexts to revoke a user.

[9] We can upgrade FE for NC^1 to FE for P/poly by using randomized encoding [ABSV15].

[10] We omit references since there are too many previous works on broadcast encryption and trace-and-revoke systems.

2 Preliminaries

Notations and conventions. In this paper, standard math or sans serif font stands for classical algorithms (e.g., C or Gen) and classical variables (e.g., x or pk). Calligraphic font stands for quantum algorithms (e.g., $\mathcal{G}en$) and calligraphic font and/or the bracket notation for (mixed) quantum states (e.g., q or $|\psi\rangle$). For strings x and y, $x\|y$ denotes the concatenation of x and y. Let $\mathbf{0}$ denote a string consisting of an appropriate number of 0. Let $[\ell]$ denote the set of integers $\{1,\cdots,\ell\}$, λ denote a security parameter, and $y := z$ denote that y is set, defined, or substituted by z.

In this paper, for a finite set X and a distribution D, $x \leftarrow X$ denotes selecting an element from X uniformly at random, $x \leftarrow D$ denotes sampling an element x according to D. Let $y \leftarrow \mathsf{A}(x)$ and $y \leftarrow \mathcal{A}(\chi)$ denote assigning to y the output of a probabilistic or deterministic algorithm A and a quantum algorithm $\mathcal{A}$ on an input x and χ, respectively. When we explicitly show that A uses randomness r, we write $y \leftarrow \mathsf{A}(x;r)$. PPT and QPT algorithms stand for probabilistic polynomial-time algorithms and polynomial-time quantum algorithms, respectively. Let negl denote a negligible function.

2.1 Standard Cryptographic Tools

Definition 2.1 (Pseudorandom Function). *Let $\{\mathsf{F}_K : \{0,1\}^{\ell_1} \rightarrow \{0,1\}^{\ell_2} \mid K \in \{0,1\}^\lambda\}$ be a family of polynomially computable functions, where ℓ_1 and ℓ_2 are some polynomials of λ. We say that F is a pseudorandom function (PRF) family if, for any PPT distinguisher $\mathcal{A}$, there exists $\mathsf{negl}(\cdot)$ such that it holds that*

$$\left|\Pr\left[\mathcal{A}^{\mathsf{F}_K(\cdot)}(1^\lambda) = 1 \mid K \leftarrow \{0,1\}^\lambda\right] - \Pr\left[\mathcal{A}^{\mathsf{R}(\cdot)}(1^\lambda) = 1 \mid \mathsf{R} \leftarrow \mathcal{U}\right]\right| \leq \mathsf{negl}(\lambda),$$

where $\mathcal{U}$ is the set of all functions from $\{0,1\}^{\ell_1}$ to $\{0,1\}^{\ell_2}$.

Theorem 2.2 ([GGM86]). *If one-way functions exist, then for all efficiently computable functions $n(\lambda)$ and $m(\lambda)$, there exists a PRF that maps $n(\lambda)$ bits to $m(\lambda)$ bits.*

Definition 2.3 (Secret Key Encryption). *An SKE scheme SKE is a two tuple (E,D) of PPT algorithms.*

- *The encryption algorithm E, given a key $K \in \{0,1\}^\lambda$ and a plaintext $m \in \mathcal{M}$, outputs a ciphertext ct, where $\mathcal{M}$ is the plaintext space of SKE.*
- *The decryption algorithm D, given a key K and a ciphertext ct, outputs a plaintext $\tilde{m} \in \{\bot\} \cup \mathcal{M}$. This algorithm is deterministic.*

Correctness: *We require $\mathsf{D}(K,\mathsf{E}(K,m)) = m$ for every $m \in \mathcal{M}$ and key $K \in \{0,1\}^\lambda$.*

CPA Security: *We define the following experiment $\mathsf{Expt}^{\mathsf{cpa}}_{\mathcal{A},\mathsf{SKE}}(1^\lambda,\mathsf{coin})$ between a challenger and an adversary $\mathcal{A}$.*

1. *The challenger generates $K \leftarrow \{0,1\}^\lambda$. Then, the challenger sends 1^λ to $\mathcal{A}$.*
2. *$\mathcal{A}$ may make polynomially many encryption queries adaptively. $\mathcal{A}$ sends $(m_0,m_1) \in \mathcal{M}\times\mathcal{M}$ to the challenger. Then, the challenger returns $\mathsf{ct} \leftarrow \mathsf{E}(K,m_{\mathsf{coin}})$.*
3. *$\mathcal{A}$ outputs $\mathsf{coin}' \in \{0,1\}$.*

We say that SKE is CPA secure if for any QPT adversary $\mathcal{A}$, we have

$$\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{SKE},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathsf{Expt}^{\mathsf{cpa}}_{\mathcal{A},\mathsf{SKE}}(1^\lambda,0)=1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{cpa}}_{\mathcal{A},\mathsf{SKE}}(1^\lambda,1)=1\right]\right| \leq \mathsf{negl}(\lambda).$$

Definition 2.4 (Ciphertext Pseudorandomness for SKE). *Let $\{0,1\}^\ell$ be the ciphertext space of SKE. We define the following experiment $\mathsf{Exp}^{\mathsf{pr\text{-}ct}}_{\mathcal{A},\mathsf{SKE}}(1^\lambda,\mathsf{coin})$ between a challenger and an adversary $\mathcal{A}$.*

1. *The challenger generates* $K \leftarrow \{0,1\}^\lambda$*. Then, the challenger sends* 1^λ *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *may make polynomially many encryption queries adaptively.* $\mathcal{A}$ *sends* $m \in \mathcal{M}$ *to the challenger. Then, the challenger returns* $\mathsf{ct} \leftarrow \mathsf{E}(K, m)$ *if* $\mathsf{coin} = 0$*, otherwise* $\mathsf{ct} \leftarrow \{0,1\}^\ell$*.*
3. $\mathcal{A}$ *outputs* $\mathsf{coin}' \in \{0,1\}$*.*

We say that SKE *is pseudorandom-secure if for any QPT adversary* $\mathcal{A}$*, we have*

$$\mathsf{Adv}^{\mathsf{pr\text{-}ct}}_{\mathsf{SKE},\mathcal{A}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\mathsf{pr\text{-}ct}}_{\mathcal{A},\mathsf{SKE}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{pr\text{-}ct}}_{\mathcal{A},\mathsf{SKE}}(1^\lambda, 1) = 1\right]\right| \le \mathsf{negl}(\lambda).$$

Theorem 2.5. *If OWFs exist, there exists a pseudorandom-secure SKE scheme.*

Definition 2.6 (Secret-Key Functional Encryption). *An SKFE scheme* SKFE *is a tuple of four PPT algorithms* $(\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$*. Below, let* $\mathcal{X}$*,* $\mathcal{Y}$*, and* $\mathcal{F}$ *be the plaintext, output, and function spaces* SKFE*, respectively.*

$\mathsf{Setup}(1^\lambda, 1^q) \rightarrow \mathsf{msk}$**:** *The setup algorithm takes a security parameter* 1^λ *and a collusion bound* 1^q*, and outputs a master secret key* msk*.*

$\mathsf{KG}(\mathsf{msk}, f) \rightarrow \mathsf{sk}_f$**:** *The key generation algorithm takes a master secret key* msk *and a function* $f \in \mathcal{F}$*, and outputs a functional decryption key* sk_f*.*

$\mathsf{Enc}(\mathsf{msk}, x) \rightarrow \mathsf{ct}$**:** *The encryption algorithm takes a master secret key* msk *and a plaintext* $x \in \mathcal{X}$*, and outputs a ciphertext* ct*.*

$\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) \rightarrow y$**:** *The decryption algorithm takes a functional decryption key* sk_f *and a ciphertext* ct*, and outputs* $y \in \{\bot\} \cup \mathcal{Y}$*.*

Correctness: *We require that for every* $x \in \mathcal{X}$*,* $f \in \mathcal{F}$*,* $q \in \mathbb{N}$*, we have that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) = f(x) \;\middle|\; \begin{array}{l} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q), \\ \mathsf{sk}_f \leftarrow \mathsf{KG}(\mathsf{msk}, f), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, x) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Definition 2.7 (Selective Indistinguishability-Security). *We say that* SKFE *is a selectively indistinguishability-secure SKFE scheme for* $\mathcal{X}, \mathcal{Y}$*, and* $\mathcal{F}$*, if it satisfies the following requirement, formalized from the experiment* $\mathsf{Exp}^{\mathsf{sel\text{-}ind}}_{\mathcal{A},\mathsf{SKFE}}(1^\lambda, \mathsf{coin})$ *between an adversary* $\mathcal{A}$ *and a challenger:*

1. *At the beginning,* $\mathcal{A}$ *sends* $(1^q, x_0^*, x_1^*)$ *to the challenger. The challenger runs* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q)$*. Also, the challenger generates* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{msk}, x^*_{\mathsf{coin}})$ *and sends* ct^* *to* $\mathcal{A}$*. Throughout the experiment,* $\mathcal{A}$ *can access the following oracles.*

   $O_{\mathsf{Enc}}(x)$**:** *Given* x*, it returns* $\mathsf{Enc}(\mathsf{msk}, x)$*.*

   $O_{\mathsf{KG}}(f)$**:** *Given* f*, if* $f(x_0^*) \neq f(x_1^*)$*, it returns* $\bot$*. Otherwise, it returns* $\mathsf{KG}(\mathsf{msk}, f)$*.* $\mathcal{A}$ *can access this oracle at most* q *times.*

2. $\mathcal{A}$ *outputs a guess* coin' *for* coin*. The challenger outputs* coin' *as the final output of the experiment.*

We say that SKFE *is selectively indistinguishability-secure if, for any QPT* $\mathcal{A}$*, it holds that*

$$\mathsf{Adv}^{\mathsf{sel\text{-}ind}}_{\mathsf{SKFE},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{sel\text{-}ind}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{sel\text{-}ind}}_{\mathsf{SKFE},\mathcal{A}}(1^\lambda, 1) = 1\right]\right| \le \mathsf{negl}(\lambda).$$

It is easy to see that we can consider the adaptively indistinguishability-secure variant as in Definition 7.15.

Theorem 2.8 ([GVW12]). *If there exist OWFs, then there exists selectively indistinguishability-secure SKFE for* P/poly*.*

2.2 Advanced Tools

We present the definitions for reusable SKE with certified deletion introduced by Hiroka et al. [HMNY21]

Definition 2.9 (Reusable SKE with Certified Deletion (Syntax)). *A secret key encryption scheme with certified deletion is a tuple of quantum algorithms* $(\mathsf{KG}, \mathcal{Enc}, \mathcal{Dec}, \mathcal{Del}, \mathsf{Vrfy})$ *with plaintext space* $\mathcal{M}$ *and key space* $\mathcal{K}$.

$\mathsf{KG}(1^\lambda) \to \mathsf{sk}$**:** *The key generation algorithm takes as input the security parameter* 1^λ *and outputs a secret key* $\mathsf{sk} \in \mathcal{K}$.

$\mathcal{Enc}(\mathsf{sk}, m) \to (\mathsf{vk}, ct)$**:** *The encryption algorithm takes as input* sk *and a plaintext* $m \in \mathcal{M}$ *and outputs a verification key* vk *and a ciphertext* ct.

$\mathcal{Dec}(\mathsf{sk}, ct) \to m'$**:** *The decryption algorithm takes as input* sk *and* ct *and outputs a plaintext* $m' \in \mathcal{M}$ *or* $\bot$.

$\mathcal{Del}(ct) \to \mathsf{cert}$**:** *The deletion algorithm takes as input* ct *and outputs a certification* cert.

$\mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert}) \to \top$ **or** $\bot$**:** *The verification algorithm takes* vk *and* cert *and outputs* $\top$ *or* $\bot$.

Decryption correctness: *There exists a negligible function* negl *such that for any* $m \in \mathcal{M}$,

$$\Pr\left[\mathcal{Dec}(\mathsf{sk}, ct) = m \;\middle|\; \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{KG}(1^\lambda) \\ (\mathsf{vk}, ct) \leftarrow \mathcal{Enc}(\mathsf{sk}, m) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Verification correctness: *There exists a negligible function* negl *such that for any* $m \in \mathcal{M}$,

$$\Pr\left[\mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert}) = \top \;\middle|\; \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{KG}(1^\lambda) \\ (\mathsf{vk}, ct) \leftarrow \mathcal{Enc}(\mathsf{sk}, m) \\ \mathsf{cert} \leftarrow \mathcal{Del}(ct) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Definition 2.10 (IND-CPA-CD Security for Reusable SKE with Certified Deletion). *Let* $\Sigma = (\mathsf{KG}, \mathcal{Enc}, \mathcal{Dec}, \mathcal{Del}, \mathsf{Vrfy})$ *be a secret key encryption with certified deletion. We consider the following security experiment* $\mathsf{Exp}^{\mathsf{sk\text{-}cert\text{-}del}}_{\Sigma,\mathcal{A}}(\lambda, b)$.

1. *The challenger computes* $\mathsf{sk} \leftarrow \mathsf{KG}(1^\lambda)$.
2. $\mathcal{A}$ *sends an encryption query* m *to the challenger. The challenger computes* $(\mathsf{vk}, ct) \leftarrow \mathcal{Enc}(\mathsf{sk}, m)$ *to* $\mathcal{A}$ *and returns* (vk, ct) *to* $\mathcal{A}$. *This process can be repeated polynomially many times.*
3. $\mathcal{A}$ *sends* $(m_0, m_1) \in \mathcal{M}^2$ *to the challenger.*
4. *The challenger computes* $(\mathsf{vk}_b, ct_b) \leftarrow \mathcal{Enc}(\mathsf{sk}, m_b)$ *and sends* ct_b *to* $\mathcal{A}$.
5. *Again,* $\mathcal{A}$ *can send encryption queries.*
6. *At some point,* $\mathcal{A}$ *sends* cert *to the challenger.*
7. *The challenger computes* $\mathsf{Vrfy}(\mathsf{vk}_b, \mathsf{cert})$. *If the output is* $\bot$, *the challenger sends* $\bot$ *to* $\mathcal{A}$. *If the output is* $\top$, *the challenger sends* sk *to* $\mathcal{A}$.
8. *If the challenger sends* $\bot$ *in the previous item,* $\mathcal{A}$ *can send encryption queries again.*
9. $\mathcal{A}$ *outputs* $b' \in \{0, 1\}$.

We say that Σ *is IND-CPA-CD secure if for any QPT* $\mathcal{A}$, *it holds that*

$$\mathsf{Adv}^{\mathsf{sk\text{-}cert\text{-}del}}_{\Sigma,\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{sk\text{-}cert\text{-}del}}_{\Sigma,\mathcal{A}}(\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{sk\text{-}cert\text{-}del}}_{\Sigma,\mathcal{A}}(\lambda, 1) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

We introduce an additional property for certificates.

Definition 2.11 (Unique Certificate). *We say that an SKE scheme with certified deletion has the unique certificate property if there is at most one certification* cert *for each* vk *output by* Enc *such that* $\top = \mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$.

Known constructions of SKE (and PKE) with certified deletion based on BB84 states has the unique certificate property.

Theorem 2.12 ([HMNY21]). *If there exist OWFs, there exists an IND-CPA-CD secure SKE scheme that has the unique certificate property.*

We introduce a variant of IND-CPA-CD security where the adversary can send many verification queries, called indistinguishability against Chosen Verification Attacks (CVA). We use this security notion to achieve SKFE with secure key leasing in Section 4.

Definition 2.13 (IND-CVA-CD Security for Reusable SKE with Certified Deletion). *Let* $\Sigma = (\mathsf{KG}, \mathcal{E}nc, \mathcal{D}ec, \mathcal{D}el, \mathsf{Vrfy})$ *be a secret key encryption with certified deletion. We consider the following security experiment* $\mathsf{Exp}^{\mathsf{sk\text{-}cert\text{-}vo}}_{\Sigma,\mathcal{A}}(\lambda, b)$.

1. *The challenger computes* $\mathsf{sk} \leftarrow \mathsf{KG}(1^\lambda)$.
2. $\mathcal{A}$ *sends an encryption query* m *to the challenger. The challenger computes* $(\mathsf{vk}, \mathsf{ct}) \leftarrow \mathcal{E}nc(\mathsf{sk}, m)$ *to* $\mathcal{A}$ *and returns* $(\mathsf{vk}, \mathsf{ct})$ *to* $\mathcal{A}$. *This process can be repeated polynomially many times.*
3. $\mathcal{A}$ *sends* $(m_0, m_1) \in \mathcal{M}^2$ *to the challenger.*
4. *The challenger computes* $(\mathsf{vk}_b, ct_b) \leftarrow \mathcal{E}nc(\mathsf{sk}, m_b)$ *and sends* ct_b *to* $\mathcal{A}$.
5. *Again,* $\mathcal{A}$ *can send encryption queries.* $\mathcal{A}$ *can also send a verification query* cert *to the challenger. The challenger returns* sk *if* $\top = \mathsf{Vrfy}(\mathsf{vk}_b, \mathsf{cert})$, $\bot$ *otherwise. This process can be repeated polynomially many times.*
6. $\mathcal{A}$ *outputs* $b' \in \{0, 1\}$.

We say that Σ *is IND-CVA-CD secure if for any QPT* $\mathcal{A}$, *it holds that*

$$\mathsf{Adv}^{\mathsf{sk\text{-}cert\text{-}vo}}_{\Sigma,\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{sk\text{-}cert\text{-}vo}}_{\Sigma,\mathcal{A}}(\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{sk\text{-}cert\text{-}vo}}_{\Sigma,\mathcal{A}}(\lambda, 1) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Known IND-CPA-CD secure SKE schemes satisfy IND-CVA-CD security thanks to the following theorem since they satisfy Definition 2.11.

Theorem 2.14. *If an SKE scheme is IND-CPA-CD secure and has the unique certificate property, then it also satisfies IND-CVA-CD security.*

Proof. We construct an adversary $\mathcal{B}$ for IND-CPA-CD by using an adversary $\mathcal{A}$ for IND-CVA-CD as follows.

1. At the beginning, the adversary $\mathcal{A}$ is fixed, so the number of the verification query by $\mathcal{A}$ is also fixed. We denote the number by q.
2. When $\mathcal{A}$ sends an encryption query m, $\mathcal{B}$ sends m to its challenger, receives $(\mathsf{vk}, ct) \leftarrow \mathcal{E}nc(\mathsf{sk}, m)$, and passes (vk, ct) to $\mathcal{A}$.
3. When $\mathcal{A}$ sends (m_0, m_1), $\mathcal{B}$ sends (m_0, m_1) to its challenger, receives ct_b, passes ct_b to $\mathcal{A}$.
4. Then, $\mathcal{B}$ guesses an index $i^* \leftarrow [q + 1]$.

5. When $\mathcal{A}$ sends cert_i as the i-th verification query, if $i < i^*$, $\mathcal{B}$ returns $\bot$. When $\mathcal{A}$ sends cert_{i^*} as the i^*-th verification query, if there exits i such that $\mathsf{cert}_i = \mathsf{cert}_{i^*}$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ forwards cert_{i^*} to its challenger. If $\mathcal{B}$ receives $\bot$, $\mathcal{B}$ aborts. If $\mathcal{B}$ receives sk, then $\mathcal{B}$ passes sk to $\mathcal{A}$. After $\mathcal{B}$ receives sk, if a query $\mathsf{cert}_i = \mathsf{cert}_{i^*}$, $\mathcal{B}$ answers sk, otherwise $\bot$.

6. $\mathcal{B}$ outputs what $\mathcal{A}$ outputs.

Since we consider unique certificate SKE with certified deletion, if $\mathcal{B}$ receives sk from its challenger, it is the only one correct certificate under vk_b. Hence, $\mathcal{B}$ can correctly simulate the IND-CVA-CD game with probability $1/(q+1)$ (Note that $i^* = q+1$ means $\mathcal{A}$ did not output a correct certificate). Therefore, we obtain $\mathsf{Adv}^{\mathsf{sk\text{-}cert\text{-}del}}_{\Sigma,\mathcal{A}}(\lambda) = \frac{1}{q+1}\mathsf{Adv}^{\mathsf{sk\text{-}cert\text{-}vo}}_{\Sigma,\mathcal{A}}(\lambda)$, and the proof completes. ■

Definition 2.15 (Set Homomorphic Secret Sharing). *A set homomorphic secret sharing scheme* SetHSS *consists of three algorithms* $(\mathsf{InpEncode}, \mathsf{FuncEncode}, \mathsf{Decode})$. *Below, we let* $\mathcal{X}$, $\mathcal{Y}$, *and* $\mathcal{F}$ *be the input, output, and function spaces of* SetHSS, *respectively.*

$\mathsf{SetGen}(1^\lambda) \to \mathsf{params} := (p, \ell, (T_i)_{i\in[m]})$: *The set generation algorithm takes as input a security parameter, and outputs parameters p and ℓ, and a collection of m sets $(T_i)_{i\in[m]}$, where each set $T_i \subseteq [\ell]$.*

$\mathsf{InpEncode}(\mathsf{params}, x) \to (s_i)_{i\in[m]}$: *The input encoding algorithm takes as input* params *output by* SetGen *and an input $x \in \mathcal{X}$, and outputs a set of input shares $(s_i)_{i\in[m]}$.*

$\mathsf{FuncEncode}(\mathsf{params}, f) \to (f_i)_{i\in[m]}$: *The function encoding algorithm takes as input* params *output by* SetGen *and a function $f \in \mathcal{F}$, and outputs a set of function shares $(f_i)_{i\in[m]}$.*

$\mathsf{Decode}((f_i(s_i))_{i\in[m]}) \to y$: *The decoding algorithm takes as input a set of evaluations of function shares on their respective input shares $(f_i(s_i))_{i\in[m]}$ and outputs a value $y \in \mathcal{Y}$.*

Correctness: *We require that for every $x \in \mathcal{X}$, $f \in \mathcal{F}$, we have that*

$$\Pr\left[\mathsf{Decode}((f_i(s_i))_{i\in[m]}) \neq f(x) \;\middle|\; \begin{array}{l} \mathsf{params} = (p, \ell, (T_i)_{i\in[m]}) \leftarrow \mathsf{SetGen}(1^\lambda), \\ (s_i)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x), \\ (f_i)_{i\in[m]} \leftarrow \mathsf{FuncEncode}(\mathsf{params}, f) \end{array}\right] = \mathsf{negl}(\lambda).$$

Existence of Unmarked Element: *Let* $\mathsf{params} = (p, \ell, (T_i)_{i\in[m]}) \leftarrow \mathsf{SetGen}(1^\lambda)$. *Suppose we randomly generate a set $S \subseteq [m]$ so that each $i \in [m]$ is independently included in S with probability at most p. Then, without negligible probability, there exists $e \in [\ell]$ such that $e \notin \bigcup_{i\in S} T_i$.*

Selective Indistinguishability-Security: *Consider the following experiment* $\mathsf{Exp}^{\mathsf{sel\text{-}ind}}_{\mathcal{A},\mathsf{SetHSS}}(1^\lambda, \mathsf{coin})$ *between an adversary $\mathcal{A}$ and a challenger:*

1. *The challenger generates* $\mathsf{params} = (p, \ell, (T_i)_{i\in[m]}) \leftarrow \mathsf{SetGen}(1^\lambda)$ *and sends* params *to $\mathcal{A}$. $\mathcal{A}$ sends (e^*, x_0^*, x_1^*) to the challenger, where $e^* \in [\ell]$. The challenger runs* $(s_i)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x_{\mathsf{coin}})$ *and sends $(s_i)_{i\in[m]_{e^*\notin}}$ to $\mathcal{A}$, where $[m]_{e^*\notin}$ denotes the subset of $[m]$ consisting of i such that $e^* \notin T_i$. $\mathcal{A}$ can access the following oracle.*

   $O_{\mathsf{FuncEncode}}(f)$: *Given f, if $f(x_0^*) \neq f(x_1^*)$, it returns $\bot$. Otherwise, it generates* $(f_i)_{i\in[m]} \leftarrow \mathsf{FuncEncode}(\mathsf{params}, f)$ *and returns $(f_i, f_i(s_i))_{i\in[m]}$.*

2. *$\mathcal{A}$ outputs a guess* coin' *for* coin. *The challenger outputs* coin' *as the final output of the experiment.*

We say that SetHSS *is selectively indistinguishability-secure if, for any QPT $\mathcal{A}$, it holds that*

$$\mathsf{Adv}^{\mathsf{sel\text{-}ind}}_{\mathsf{SetHSS},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{sel\text{-}ind}}_{\mathsf{SetHSS},\mathcal{A}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{sel\text{-}ind}}_{\mathsf{SetHSS},\mathcal{A}}(1^\lambda, 0) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Theorem 2.16 ([JKMS20]). *If there exist OWFs, there exists a set homomorphic secret sharing.*

The above definition of SetHSS is different from the original one by Jain et al. [JKMS20] in that the original definition does not have the set generation algorithm and does not require the existence of unmarked element property. Jain et al. defined SetHSS so that it works for any collection of sets. Then, Jain et al. separately introduced the set generation algorithm and properties of it, which perfectly match the security bound amplification for FE. We can obtain Theorem 2.16 from [JKMS20, Theorem 5.3, Lemma 6.1 and 6.2, in eprint ver.] and the parmeter setting provided in the proof of [JKMS20, Theorem 8.1 in eprint ver.]. In their parameter setting, $p = 1/\mathrm{poly}(\lambda)$, $\ell = \lambda$, and $m = \mathrm{polylog}(\lambda)$.

3 Definition of SKFE with Secure Key Leasing

We introduce the definition of SKFE with secure key leasing and its variants.

3.1 SKFE with Secure Key Leasing

We first define SKFE with secure key leasing (SKFE-SKL).

Definition 3.1 (SKFE with Secure Key Leasing). *An SKFE-SKL scheme* $\mathsf{SKFE\text{-}SKL}$ *is a tuple of six algorithms* $(\mathsf{Setup}, \mathcal{KG}, \mathsf{Enc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$. *Below, let* $\mathcal{X}$, $\mathcal{Y}$, *and* $\mathcal{F}$ *be the plaintext, output, and function spaces of* $\mathsf{SKFE\text{-}SKL}$, *respectively.*

$\mathsf{Setup}(1^\lambda, 1^q) \rightarrow \mathsf{msk}$: *The setup algorithm takes a security parameter* 1^λ *and a collusion bound* 1^q, *and outputs a master secret key* msk.

$\mathcal{KG}(\mathsf{msk}, f, 1^n) \rightarrow (\mathit{fsk}, \mathsf{vk})$: *The key generation algorithm takes a master secret key* msk, *a function* $f \in \mathcal{F}$, *and an availability bound* 1^n, *and outputs a functional decryption key* fsk *and a verification key* vk.

$\mathsf{Enc}(\mathsf{msk}, x) \rightarrow \mathsf{ct}$: *The encryption algorithm takes a master secret key* msk *and a plaintext* $x \in \mathcal{X}$, *and outputs a ciphertext* ct.

$\mathcal{Dec}(\mathit{fsk}, \mathsf{ct}) \rightarrow \tilde{x}$: *The decryption algorithm takes a functional decryption key* fsk *and a ciphertext* ct, *and outputs a value* $\tilde{x}$.

$\mathcal{Cert}(\mathit{fsk}) \rightarrow \mathsf{cert}$: *The certification algorithm takes a function decryption key* fsk, *and outputs a classical string* cert.

$\mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert}) \rightarrow \top/\bot$: *The certification-verification algorithm takes a verification key* vk *and a string* cert, *and outputs* $\top$ *or* $\bot$.

Decryption correctness: *For every* $x \in \mathcal{X}$, $f \in \mathcal{F}$, *and* $q, n \in \mathbb{N}$, *we have*

$$\Pr\left[\mathcal{Dec}(\mathit{fsk}, \mathsf{ct}) = f(x) \;\middle|\; \begin{array}{l} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q) \\ (\mathit{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f, 1^n) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, x) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Verification correctness: *For every* $f \in \mathcal{F}$ *and* $q, n \in \mathbb{N}$, *we have*

$$\Pr\left[\mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert}) = \top \;\middle|\; \begin{array}{l} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q) \\ (\mathit{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f, 1^n) \\ \mathsf{cert} \leftarrow \mathcal{Cert}(\mathit{fsk}) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Definition 3.2 (Selective Lessor Security). *We say that* $\mathsf{SKFE\text{-}SKL}$ *is a selectively lessor secure SKFE-SKL scheme for* $\mathcal{X}, \mathcal{Y}$, *and* $\mathcal{F}$, *if it satisfies the following requirement, formalized from the experiment* $\mathsf{Exp}^{\mathsf{sel\text{-}lessor}}_{\mathcal{A},\mathsf{SKFE\text{-}SKL}}(1^\lambda, \mathsf{coin})$ *between an adversary* $\mathcal{A}$ *and a challenger:*

1. *At the beginning, $\mathcal{A}$ sends $(1^q, x_0^*, x_1^*)$ to the challenger. The challenger runs* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q)$. *Throughout the experiment, $\mathcal{A}$ can access the following oracles.*

   $O_{\mathsf{Enc}}(x)$**:** *Given x, it returns* $\mathsf{Enc}(\mathsf{msk}, x)$.

   $O_{\mathcal{KG}}(f, 1^n)$**:** *Given $(f, 1^n)$, it generates $(\mathit{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f, 1^n)$, sends fsk to $\mathcal{A}$, and adds $(f, 1^n, \mathsf{vk}, \bot)$ to $L_{\mathcal{KG}}$. $\mathcal{A}$ can access this oracle at most q times.*

   $O_{\mathsf{Vrfy}}(f, \mathsf{cert})$**:** *Given (f, cert), it finds an entry $(f, 1^n, \mathsf{vk}, M)$ from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) If $\top = \mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$ and the number of queries to O_{Enc} at this point is less than n, it returns $\top$ and updates the entry into $(f, 1^n, \mathsf{vk}, \top)$. Otherwise, it returns $\bot$.*

2. *When $\mathcal{A}$ requests the challenge ciphertext, the challenger checks if for any entry $(f, 1^n, \mathsf{vk}, M)$ in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$. If so, the challenger generates $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{msk}, x_{\mathsf{coin}}^*)$ and sends ct^* to $\mathcal{A}$. Otherwise, the challenger outputs 0. Hereafter, $\mathcal{A}$ is not allowed to sends a function f such that $f(x_0^*) \neq f(x_1^*)$ to $O_{\mathcal{KG}}$.*

3. *$\mathcal{A}$ outputs a guess coin' for coin. The challenger outputs coin' as the final output of the experiment.*

For any QPT $\mathcal{A}$, it holds that

$$\mathsf{Adv}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}^{\mathsf{sel\text{-}lessor}}(\lambda) := \left|\Pr\left[\mathsf{Exp}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}^{\mathsf{sel\text{-}lessor}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}^{\mathsf{sel\text{-}lessor}}(1^\lambda, 1) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Remark 3.3 (On the adaptive security). We can similarly define adaptive lessor security where we allow $\mathcal{A}$ to adaptively choose the challenge plaintext pair (x_0^*, x_1^*). For standard FE, we can generically convert a selectively secure one into an adaptively secure one without any additional assumption [ABSV15]. We observe that a similar transformation works for SKFE with secure key leasing. Thus, for simplicity, we focus on selective lessor security in this work. See Appendix B.1 for the definition and transformation.

Remark 3.4 (On the simulation-based security). We can also define a simulation-based variant of selective/adaptive lessor security where a simulator simulates a challenge ciphertext without the challenge plaintext x^* as the simulation-based security for standard FE [BSW11, GVW12]. We can generically convert indistinguishability-based lessor secure SKFE with secure key leasing into a simulation-based lessor secure one without any additional assumptions as standard FE [DIJ$^+$13]. See Appendix B.2 for the simulation-based definition and the transformation.

3.2 SKFE with Static-Bound Secure Key Leasing

In this section, we define SKFE with static-bound secure key leasing (SKFE-sbSKL). It is a weaker variant of SKFE-SKL in which a single availability bound n applied to every decryption key is fixed at the setup time. We design SKFE-sbSKL so that it can be transformed into SKFE-SKL in a generic way. For this reason, we require an SKFE-sbSKL scheme to satisfy an efficiency requirement called weak optimal efficiency and slightly stronger variant of the lessor security notion. [11]

Below, we first introduce the syntax of SKFE-sbSKL. Then, before introducing the definition of (selective) lessor security for it, we provide the overview of the transformation to SKFE-SKL since we think the overview makes it easy to understand the security notion.

Definition 3.5 (SKFE with Static-Bound Secure Key Leasing). *An SKFE-sbSKL scheme* $\mathsf{SKFE\text{-}sbSKL}$ *is a tuple of six algorithms* $(\mathsf{Setup}, \mathcal{KG}, \mathsf{Enc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$. *The only difference from a normal SKFE scheme with secure key leasing is that $\mathcal{KG}$ does not take as input the availability bound n, and instead,* Setup *takes it as an input. Moreover,* Setup *takes it in binary as* $\mathsf{Setup}(1^\lambda, 1^q, n)$, *and we require the following weak optimal efficiency.*

[11] We borrow the term weak optimal efficiency from the paper by Garg, Goyal, Lu, and Waters [GGLW21], which studies dynamic bounded collusion security for standard FE.

Weak Optimal Efficiency: *We require that the running time of* Setup *and* Enc *is bounded by a fixed polynomial of* λ, q, *and* $\log n$.

Overview of the transformation to SKFE-SKL. As seen above, Setup and Enc of an SKFE-sbSKL scheme $\mathsf{SKFE\text{-}sbSKL}$ is required to run in time $\log n$. This is because, in the transformation to SKFE-SKL, we use λ instances of $\mathsf{SKFE\text{-}sbSKL}$ where the k-th instance is set up with the availability bound 2^k for every $k \in [\lambda]$. The weak optimal efficiency ensures that Setup and Enc of all λ instances run in polynomial time. The details of the transformation are as follows.

We construct an SKFE-SKL scheme $\mathsf{SKFE\text{-}SKL}$ from an SKFE-sbSKL scheme $\mathsf{SKFE\text{-}sbSKL} = (\mathsf{Setup}, \mathcal{KG}, \mathsf{Enc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$. When generating a master secret key $\mathsf{skl.msk}$ of $\mathsf{SKFE\text{-}SKL}$, we generate $\mathsf{msk}_k \leftarrow \mathsf{Setup}(1^\lambda, 1^q, 2^k)$ for every $k \in [\lambda]$, and set $\mathsf{skl.msk} := (\mathsf{msk}_k)_{k\in[\lambda]}$. To encrypt x by $\mathsf{SKFE\text{-}SKL}$, we encrypt it by all λ instances, that is, generate $\mathsf{ct}_k \leftarrow \mathsf{Enc}(\mathsf{msk}_k, x)$ for every $k \in [\lambda]$. The resulting ciphertext is $\mathsf{skl.ct} := (\mathsf{ct}_k)_{k\in[\lambda]}$. To generate a decryption key of $\mathsf{SKFE\text{-}SKL}$ for a function f and an availability bound n, we first compute $k' \in [\lambda]$ such that $2^{k'-1} \le n \le 2^{k'}$. Then, we generate $(\mathit{fsk}_{k'}, \mathsf{vk}_{k'}) \leftarrow \mathcal{KG}(\mathsf{msk}_{k'}, f)$. The resulting decryption key is $\mathsf{skl}.\mathit{fsk} := (k', \mathit{fsk}_{k'})$ and the corresponding verification key is $\mathsf{vk} := \mathsf{vk}_{k'}$. Decryption is performed by decrypting $\mathsf{ct}_{k'}$ included in $\mathsf{skl.ct} := (\mathsf{ct}_k)_{k\in[\lambda]}$ by $\mathit{fsk}_{k'}$. The certification generation and verification of $\mathsf{SKFE\text{-}SKL}$ are simply those of $\mathsf{SKFE\text{-}SKL}$.

We now consider the security proof of $\mathsf{SKFE\text{-}SKL}$. In the experiment $\mathsf{Exp}^{\mathsf{sel\text{-}lessor}}_{\mathcal{A},\mathsf{SKFE\text{-}SKL}}(1^\lambda, 0)$, an adversary $\mathcal{A}$ is given the challenge ciphertext $\mathsf{skl.ct}^* := (\mathsf{ct}^*_k)_{k\in[\lambda]}$, where $\mathsf{ct}^*_k \leftarrow \mathsf{Enc}(\mathsf{msk}_k, x^*_0)$ for every $k \in [\lambda]$. The proof is done if we can switch all of ct^*_k into $\mathsf{Enc}(\mathsf{msk}_k, x^*_1)$ without being detected by $\mathcal{A}$. To this end, the underlying $\mathsf{SKFE\text{-}sbSKL}$ needs to satisfy a stronger variant of lessor security notion where an adversary is allowed to declare the availability bound such that $\mathcal{KG}$ does not run in polynomial time, if the adversary does not make any query to the key generation oracle. For example, to switch ct^*_λ, the reduction algorithm attacking $\mathsf{SKFE\text{-}sbSKL}$ needs to declare the availability bound 2^λ, under which $\mathcal{KG}$ might not run in polynomial time. Note that Setup and Enc run in polynomial time even for such an availability bound due to the weak optimal efficiency. Thus, we formalize the security notion of SKFE-sbSKL as follows.

Definition 3.6 (Selective Strtong Lessor Security). *We define selective strong lessor security for SKFE-sbSKL in the same way as that for SKFE-SKL defined in Definition 3.2 , except the following changes for the security experiment.*

- $\mathcal{A}$ *outputs* n *at the beginning, and the challenger generates* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q, n)$. *If* $\mathcal{A}$ *makes a query to* $O_{\mathcal{KG}}$ *or* O_{Vrfy}, $\mathcal{A}$ *is required to output* n *such that* $\mathcal{KG}$ *and* Vrfy *run in polynomial time.*
- $O_{\mathcal{KG}}$ *does not take* 1^n *as an input.*

Remark 3.7 (Insufficiency of existing bounded collusion techniques). In Section 1.3, we stated that it is not clear how to use the existing bounded collusion techniques [GVW12, AV19] for constructing SKFE-sbSKL. We provide a more detailed discussion on this point.

The bounded collusion technique essentially enables us to increase the number of decryption keys that an adversary can obtain. Thus, to try to use the bounded collusion technique in our context, imagine the following naive construction using standard SKFE SKFE and SKE with certified deletion CDSKE. This construction is a flipped version of the naive construction provided in Section 1.3. In the construction, we encrypt a ciphertext of SKFE by CDSKE, and we include the key of CDSKE into the decryption key of the resulting scheme. The construction can be seen as an SKFE scheme with certified deletion (for ciphertexts) that is secure if an adversary deletes the challenge ciphertext before seeing any decryption key. The roles of ciphertexts and decryption keys are almost symmetric in SKFE [BS18]. Thus, if we can amplify the security of this construction so that it is secure if an adversary sees some decryption keys before deleting the challenge ciphertext, it would lead to SKFE-sbSKL. The question is whether we can perform such an amplification using the existing bounded collusion techniques [GVW12, AV19]. We observe that it is highly non-trivial to adapt the existing bounded collusion technique starting from "0-bounded" security. Especially, it seems difficult to design such a transformation so that the resulting SKFE-sbSKL obtained by flipping the roles of ciphertexts and decryption keys satisfies weak optimal efficiency and security against unbounded number of encryption queries such as Definition 3.6.

We develop a different technique due to the above reason. Namely, we reduce the task of amplifying the availability bound of SKFE-sbSKL into the task of amplifying the security bound of it. In fact, our work implicitly shows that security bound amplification for FE can be used to achieve bounded collusion-resistance. We see that we can construct bounded collusion secure FE from single-key FE by our parallelizing then security bound amplification technique.

3.3 Index-Based SKFE with Static-Bound Secure Key Leasing

We define index-based SKFE-sbSKL. Similarly to SKFE-sbSKL, it needs to satisfy weak optimal efficiency and (selective) strong lessor security.

Definition 3.8 (Index-Base SKFE with Static-Bound Secure Key Leasing). *An index-base SKFE-sbSKL scheme* $\mathsf{iSKFE\text{-}sbSKL}$ *is a tuple of six algorithms* $(\mathsf{Setup}, \mathcal{KG}, \mathsf{iEnc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$. *The only difference from an SKFE-sbSKL scheme is that the encryption algorithm* iEnc *additionally takes as input an index* $j \in [n]$.

Decryption correctness: *For every* $x \in \mathcal{X}$, $f \in \mathcal{F}$, $q, n \in \mathbb{N}$, *and* $j \in [n]$, *we have*

$$\Pr\left[\mathcal{Dec}(\mathit{fsk}, \mathsf{ct}) = f(x) \;\middle|\; \begin{array}{l} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q, n) \\ (\mathit{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{msk}, j, x) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Verification correctness: *For every* $f \in \mathcal{F}$ *and* $q, n \in \mathbb{N}$, *we have*

$$\Pr\left[\mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert}) = \top \;\middle|\; \begin{array}{l} \mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q, n) \\ (\mathit{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f) \\ \mathsf{cert} \leftarrow \mathcal{Cert}(\mathit{fsk}) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Weak Optimal Efficiency: *We require that the running time of* Setup *and* Enc *is bounded by a fixed polynomial of* λ, q, *and* $\log n$.

Definition 3.9 (Selective Strong Lessor Security). *We say that* $\mathsf{iSKFE\text{-}sbSKL}$ *is a selectively strong lessor secure index-based SKFE-sbSKL scheme for* $\mathcal{X}, \mathcal{Y}$, *and* $\mathcal{F}$, *if it satisfies the following requirement, formalized from the experiment* $\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathcal{A},\mathsf{iSKFE\text{-}sbSKL}}(1^\lambda, \mathsf{coin})$ *between an adversary* $\mathcal{A}$ *and a challenger:*

1. *At the beginning,* $\mathcal{A}$ *sends* $(1^q, n, j^*, x_0^*, x_1^*)$ *to the challenger. If* $\mathcal{A}$ *makes a query to* $O_{\mathcal{KG}}$ *or* O_{Vrfy}, $\mathcal{A}$ *is required to output* n *such that* $\mathcal{KG}$ *and* Vrfy *run in polynomial time. The challenger runs* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^q, n)$. *Throughout the experiment,* $\mathcal{A}$ *can access the following oracles.*

   $O_{\mathsf{Enc}}(j, x)$**:** *Given* j *and* x, *it returns* $\mathsf{Enc}(\mathsf{msk}, j, x)$.

   $O_{\mathcal{KG}}(f)$**:** *Given* f, *it generates* $(\mathit{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f)$, *sends* fsk *to* $\mathcal{A}$, *and adds* $(f, \mathsf{vk}, \bot)$ *to* $L_{\mathcal{KG}}$. $\mathcal{A}$ *can access this oracle at most* q *times.*

   $O_{\mathsf{Vrfy}}(f, \mathsf{cert})$**:** *Given* (f, cert), *it finds an entry* (f, vk, M) *from* $L_{\mathcal{KG}}$. *(If there is no such entry, it returns* $\bot$.*) If* $\top = \mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$, *it returns* $\top$ *and updates the entry into* $(f, \mathsf{vk}, \top)$. *Otherwise, it returns* $\bot$.

2. *When $\mathcal{A}$ requests the challenge ciphertext, the challenger checks if for any entry (f, vk, M) in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$, and $\mathcal{A}$ does not make a query with j^* to O_{Enc} at this point. If so, the challenger generates $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{msk}, j^*, x_{\mathsf{coin}}^*)$ and sends ct^* to $\mathcal{A}$. Otherwise, the challenger outputs 0. Hereafter, $\mathcal{A}$ is not allowed to sends a function f such that $f(x_0^*) \neq f(x_1^*)$ to $O_{\mathcal{KG}}$.*
3. *$\mathcal{A}$ outputs a guess coin' for coin. The challenger outputs coin' as the final output of the experiment.*

For any QPT $\mathcal{A}$, it holds that

$$\mathsf{Adv}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathsf{iSKFE\text{-}sbSKL},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathsf{iSKFE\text{-}sbSKL},\mathcal{A}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathsf{iSKFE\text{-}sbSKL},\mathcal{A}}(1^\lambda, 0) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

4 Index-Base SKFE with Static-Bound Secure Key Leasing

We construct an index-based SKFE-sbSKL scheme $\mathsf{iSKFE\text{-}sbSKL} = (\mathsf{iSetup}, i\mathcal{KG}, \mathsf{iEnc}, i\mathcal{Dec}, i\mathcal{Cert}, \mathsf{iVrfy})$ using the following tools:

- An SKFE scheme $\mathsf{SKFE} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$.
- An SKE scheme with Certified Deletion $\mathsf{CDSKE} = (\mathsf{CD.KG}, \mathsf{CD}.\mathcal{Enc}, \mathsf{CD}.\mathcal{Dec}, \mathsf{CD}.\mathcal{Del}, \mathsf{CD.Vrfy})$.
- A PRF F.

The description of $\mathsf{iSKFE\text{-}sbSKL}$ is as follows.

$\mathsf{iSetup}(1^\lambda, 1^q, n)$:

- Generate $K \leftarrow \{0,1\}^\lambda$.
- Output $\mathsf{skl.msk} := (q, n, K)$.

$i\mathcal{KG}(\mathsf{msk}, f)$:

- Parse $(q, n, K) \leftarrow \mathsf{skl.msk}$.
- Compute $r_j \| w_j \leftarrow \mathsf{F}_K(j)$, $\mathsf{msk}_j \leftarrow \mathsf{Setup}(1^\lambda, 1^q; r_j)$, and $\mathsf{cd.sk}_j \leftarrow \mathsf{CD.KG}(1^\lambda; w_j)$ for every $j \in [n]$.
- Generate $\mathsf{fsk}_j \leftarrow \mathsf{KG}(\mathsf{msk}_j, f)$ for every $j \in [n]$.
- Generate $(\mathsf{cd}.ct_j, \mathsf{vk}_j) \leftarrow \mathsf{CD}.\mathcal{Enc}(\mathsf{cd.sk}_j, \mathsf{fsk}_j)$ for every $j \in [n]$.
- Output $\mathsf{skl}.\mathcal{fsk} := (\mathsf{cd}.ct_j)_{j\in[n]}$ and $\mathsf{vk} := (\mathsf{vk}_j)_{j\in[n]}$.

$\mathsf{iEnc}(\mathsf{skl.msk}, j, x)$:

- Parse $(q, n, K) \leftarrow \mathsf{skl.msk}$.
- Compute $r_j \| w_j \leftarrow \mathsf{F}_K(j)$, $\mathsf{msk}_j \leftarrow \mathsf{Setup}(1^\lambda, 1^q; r_j)$, and $\mathsf{cd.sk}_j \leftarrow \mathsf{CD.KG}(1^\lambda; w_j)$.
- Generate $\mathsf{ct}_j \leftarrow \mathsf{Enc}(\mathsf{msk}_j, x)$.
- Output $\mathsf{skl.ct} := (j, \mathsf{ct}_j, \mathsf{cd.sk}_j)$.

$i\mathcal{Dec}(\mathsf{skl}.\mathcal{fsk}, \mathsf{skl.ct})$:

- Parse $(\mathsf{cd}.ct_j)_{j\in[n]} \leftarrow \mathsf{skl}.\mathcal{fsk}$ and $(j, \mathsf{ct}_j, \mathsf{cd.sk}_j) \leftarrow \mathsf{skl.ct}$.
- Compute $\mathsf{fsk}_j \leftarrow \mathsf{CD}.\mathcal{Dec}(\mathsf{cd.sk}_j, \mathsf{skl}.\mathcal{fsk}_j)$.
- Output $y \leftarrow \mathsf{Dec}(\mathsf{fsk}_j, \mathsf{ct}_j)$.

$i\mathcal{Cert}(\mathsf{skl}.\mathcal{fsk})$:

- Parse $(\mathsf{cd}.ct_j)_{j\in[n]} \leftarrow \mathsf{skl}.\mathit{fsk}$.
- Compute $\mathsf{cert}_j \leftarrow \mathsf{CD}.\mathcal{Del}(\mathsf{cd.ct}_j)$ for every $j \in [n]$.
- Output $\mathsf{cert} := (\mathsf{cert}_j)_{j\in[n]}$.

$\mathsf{iVrfy}(\mathsf{vk}, \mathsf{cert})$:

- Parse $(\mathsf{vk}_j)_{j\in[n]} \leftarrow \mathsf{vk}$ and $(\mathsf{cert}_j)_{j\in[n]} \leftarrow \mathsf{cert}$.
- Output $\top$ if $\top = \mathsf{CD.Vrfy}(\mathsf{vk}_j, \mathsf{cert}_j)$ for every $j \in [n]$, and otherwise $\bot$.

It is clear that iSKFE-sbSKL satisfies correctness and weak optimal efficiency. For security, we have the following theorem.

Theorem 4.1. *If* SKFE *is selective indistinguishability-secure,* CDSKE *is IND-CVA-CD secure,*[12] *and* F *is a secure PRF, then* iSKFE-sbSKL *satisfies selective strong lessor security.*

Proof of Theorem 4.1. We define a sequence of hybrid games to prove the theorem.

Hyb_0: This is the same as $\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathcal{A},\mathsf{iSKFE\text{-}sbSKL}}(1^\lambda, 0)$.

1. At the beginning, $\mathcal{A}$ sends $(1^q, n, j^*, x_0^*, x_1^*)$ to the challenger. The challenger generates $K \leftarrow \{0,1\}^\lambda$. Below, we let $r_j \| w_j \leftarrow \mathsf{F}_K(j)$, $\mathsf{msk}_j \leftarrow \mathsf{Setup}(1^\lambda, 1^q; r_j)$, and $\mathsf{cd.sk}_j \leftarrow \mathsf{CD.KG}(1^\lambda; w_j)$ for every $j \in [n]$. Throughout the experiment, $\mathcal{A}$ can access the following oracles.

   $O_{\mathsf{Enc}}(j, x)$: Given j and x, it generates $\mathsf{ct}_j \leftarrow \mathsf{Enc}(\mathsf{msk}_j, x)$ and returns $\mathsf{skl.ct} := (j, \mathsf{ct}_j, \mathsf{cd.sk}_j)$.

   $O_{\mathcal{KG}}(f)$: Given f, it does the following.

   - Compute $\mathsf{fsk}_j \leftarrow \mathsf{KG}(\mathsf{msk}_j, f)$ for every $j \in [n]$.
   - Compute $(\mathsf{cd}.ct_j, \mathsf{vk}_j) \leftarrow \mathsf{CD}.\mathcal{Enc}(\mathsf{cd.sk}_j, \mathsf{fsk}_j)$ for every $j \in [n]$.
   - Sets $\mathsf{skl}.\mathit{fsk} := (\mathsf{cd}.ct_j)_{j\in[n]}$ and $\mathsf{skl.vk} := (\mathsf{vk}_j)_{j\in[n]}$.

   It sends $\mathsf{skl}.\mathit{fsk}$ to $\mathcal{A}$ and adds $(f, \mathsf{skl.vk}, \bot)$ to $L_{\mathcal{KG}}$. $\mathcal{A}$ is allowed to make at most q queries to this oracle.

   $O_{\mathsf{Vrfy}}(f, \mathsf{cert} := (\mathsf{cert}_j)_{j\in[n]})$: Given $(f, \mathsf{cert} := (\mathsf{cert}_j)_{j\in[n]})$, it finds an entry (f, vk, M) from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) If $\top = \mathsf{Vrfy}(\mathsf{vk}_j, \mathsf{cert}_j)$ for every $j \in [n]$, it returns $\top$ and updates the entry into $(f, \mathsf{vk}, \top)$. Otherwise, it returns $\bot$.

2. When $\mathcal{A}$ requests the challenge ciphertext, the challenger checks if for any entry (f, vk, M) in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$, and $\mathcal{A}$ does not make a query with j^* to O_{Enc} at this point. If so, the challenger generates $\mathsf{ct}^*_{j^*} \leftarrow \mathsf{Enc}(\mathsf{msk}_{j^*}, x_0^*)$ and sends $\mathsf{skl.ct}^* := (j^*, \mathsf{ct}^*_{j^*}, \mathsf{cd.sk}_{j^*})$ to $\mathcal{A}$. Otherwise, the challenger outputs 0. Hereafter, $\mathcal{A}$ is not allowed to sends a function f such that $f(x_0^*) \neq f(x_1^*)$ to $O_{\mathcal{KG}}$.
3. $\mathcal{A}$ outputs coin'. The challenger outputs coin' as the final output of the experiment.

Hyb_1: This is the same as Hyb_0 except that $r_j \| w_j$ is generated as a uniformly random string for every $j \in [n]$.

We have $|\Pr[\mathsf{Hyb}_0 = 1] - \Pr[\mathsf{Hyb}_1 = 1]| = \mathsf{negl}(\lambda)$ from the security of F.

Hyb_2: This hybrid is the same as Hyb_1 except that when $\mathcal{A}$ sends f to $O_{\mathcal{KG}}$, if $f(x_0^*) \neq f(x_1^*)$, the challenger generates $\mathsf{cd}.ct_{j^*}$ included in $\mathsf{skl}.\mathit{fsk} := (\mathsf{cd}.ct_j)_{j\in[n]}$ as $(\mathsf{cd}.ct_{j^*}, \mathsf{vk}_{j^*}) \leftarrow \mathsf{CD}.\mathcal{Enc}(\mathsf{cd.sk}_{j^*}, \mathbf{0})$.

[12] See Definition 2.13 for the defition of IND-CVA-CD.

We can show that $|\Pr[\mathsf{Hyb}_1 = 1] - \Pr[\mathsf{Hyb}_2 = 1]| = \mathsf{negl}(\lambda)$ from the security of CDSKE as follows. We say $\mathcal{A}$ is valid if when $\mathcal{A}$ requests the challenge ciphertext, for any entry (f, vk, M) in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$, and $\mathcal{A}$ does not make a query with j^* to O_{Enc} at this point. In the estimation of $|\Pr[\mathsf{Hyb}_1 = 1] - \Pr[\mathsf{Hyb}_2 = 1]|$, we have to consider the case where $\mathcal{A}$ is valid since if $\mathcal{A}$ is not valid, the output of the experiment is 0. In this transition of experiments, we change a plaintext encrypted under $\mathsf{cd.sk}_{j^*}$. If $\mathcal{A}$ is valid, $\mathcal{A}$ cannot obtain $\mathsf{cd.sk}_{j^*}$ before $\mathcal{A}$ is given $\mathsf{skl.ct}^*$, and $\mathcal{A}$ returns all ciphertexts under $\mathsf{cd.sk}_{j^*}$ before it gets $\mathsf{cd.sk}_{j^*}$. Although the reduction does not have vk_{j^*} here, it can simulate O_{Vrfy} by using the verification oracle in IND-CVA-CD game. Then, we see that $|\Pr[\mathsf{Hyb}_1 = 1] - \Pr[\mathsf{Hyb}_2 = 1]| = \mathsf{negl}(\lambda)$ follows from the security of CDSKE under the key $\mathsf{cd.sk}_{j^*}$.

Hyb_3**:** This hybrid is the same as Hyb_2 except that the challenger generates $\mathsf{ct}_{j^*}^*$ included in $\mathsf{skl.ct}^*$ as $\mathsf{ct}_{j^*}^* \leftarrow \mathsf{Enc}(\mathsf{msk}_{j^*}, x_1^*)$.

By the previous transition, in Hyb_2 and Hyb_3, $\mathcal{A}$ can obtain a decryption key under msk_{j^*} for a function f such that $f(x_0^*) = f(x_1^*)$. Thus, $|\Pr[\mathsf{Hyb}_2 = 1] - \Pr[\mathsf{Hyb}_3 = 1]| = \mathsf{negl}(\lambda)$ holds from the security of SKFE.

Hyb_4**:** This hybrid is the same as Hyb_3 except that we undo the changes from Hyb_0 to Hyb_2. Hyb_4 is the same as $\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathcal{A},\mathsf{iSKFE\text{-}sbSKL}}(1^\lambda, 1)$.

$|\Pr[\mathsf{Hyb}_3 = 1] - \Pr[\mathsf{Hyb}_4 = 1]| = \mathsf{negl}(\lambda)$ holds from the security of F and CDSKE.

From the above discussions, $\mathsf{iSKFE\text{-}sbSKL}$ satisfies selective lessor security. ■

5 SKFE with Static-Bound Secure Key Leasing

We construct an SKFE-sbSKL scheme $\mathsf{SKFE\text{-}sbSKL} = (\mathsf{sbSKL.Setup}, \mathsf{sbSKL}.\mathcal{KG}, \mathsf{sbSKL.Enc}, \mathsf{sbSKL}.\mathcal{Dec}, \mathsf{sbSKL}.\mathcal{Cert}, \mathsf{sbSKL.Vrfy})$ from the following tools:

- An index-based SKFE-sbSKL scheme $\mathsf{iSKFE\text{-}sbSKL} = (\mathsf{iSetup}, i\mathcal{KG}, \mathsf{iEnc}, i\mathcal{Dec}, i\mathcal{Cert}, \mathsf{iVrfy})$.
- A set homomorphic secret sharing $\mathsf{SetHSS} = (\mathsf{SetGen}, \mathsf{InpEncode}, \mathsf{FuncEncode}, \mathsf{Decode})$.
- An SKE scheme $\mathsf{SKE} = (\mathsf{E}, \mathsf{D})$.

The description of $\mathsf{SKFE\text{-}sbSKL}$ is as follows.

$\mathsf{sbSKL.Setup}(1^\lambda, 1^q, n)$**:**

- Generate $\mathsf{params} := (p, \ell, (T_i)_{i \in [m]}) \leftarrow \mathsf{SetGen}(1^\lambda)$.
- Generate $\mathsf{msk}_i \leftarrow \mathsf{iSetup}(1^\lambda, 1^q, N)$ for every $i \in [m]$, where $N = n/p$.
- Generate $K \leftarrow \{0,1\}^\lambda$.
- Output $\mathsf{sbskl.msk} := (\mathsf{params}, N, (\mathsf{msk})_{i \in [m]}, K)$.

$\mathsf{sbSKL}.\mathcal{KG}(\mathsf{sbskl.msk}, f)$**:**

- Parse $(\mathsf{params}, N, (\mathsf{msk})_{i \in [m]}, K) \leftarrow \mathsf{sbskl.msk}$.
- Generate $\mathsf{sct}_i \leftarrow \mathsf{E}(K, \mathbf{0})$ for every $i \in [m]$.
- Generate $(f_i)_{i \in [m]} \leftarrow \mathsf{FuncEncode}(\mathsf{params}, f)$.
- Generate $(\mathit{fsk}_i, \mathsf{vk}_i) \leftarrow i\mathcal{KG}(\mathsf{msk}_i, F[f_i, \mathsf{sct}_i])$ for every $i \in [m]$, where the circuit F is described in Figure 1.
- Output $\mathsf{sbskl}.\mathit{fsk} := (\mathit{fsk}_i)_{i \in [m]}$ and $\mathsf{sbskl.vk} := (\mathsf{vk}_i)_{i \in [m]}$.

$\mathsf{sbSKL.Enc}(\mathsf{sbskl.msk}, x)$**:**

Circuit $F[f_i, \mathsf{sct}_i](s_i, K, b)$

Hardwired: A function share f_i and an SKE's ciphertext sct_i.

Input: an input share s_i, an SKE's secret key K, and a bit b.

1. If $b = 1$, output $\mathsf{D}(K, \mathsf{sct}_i)$.
2. Otherwise, output $f_i(s_i)$.

Figure 1: Description of $F[f_i, \mathsf{sct}_i](s_i, K, b)$.

- Parse $(\mathsf{params}, N, (\mathsf{msk})_{i\in[m]}, K) \leftarrow \mathsf{sbskl.msk}$.
- Generate $(s_i)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x)$.
- Generate $j_i \leftarrow [N]$ for every $i \in [m]$.
- Generate $\mathsf{ct}_i \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i, (s_i, \mathbf{0}, 0))$ for every $i \in [m]$.
- Output $\mathsf{sbskl.ct} := (\mathsf{ct}_i)_{i\in[m]}$.

$\mathsf{sbSKL}.\mathcal{D}ec(\mathsf{sbskl}.\mathcal{fsk}, \mathsf{sbskl.ct})$**:**

- Parse $(\mathcal{fsk}_i)_{i\in[m]} \leftarrow \mathsf{sbskl}.\mathcal{fsk}$ and $(\mathsf{ct}_i)_{i\in[m]} \leftarrow \mathsf{sbskl.ct}$.
- Compute $y_I \leftarrow i\mathcal{D}ec(\mathcal{fsk}_i, \mathsf{ct}_i)$ for every $i \in [m]$.
- Output $y \leftarrow \mathsf{Decode}((y_i)_{i\in[m]})$.

$\mathsf{sbSKL}.\mathcal{C}ert(\mathsf{sbskl}.\mathcal{fsk})$**:**

- Parse $(\mathcal{fsk}_i)_{i\in[m]} \leftarrow \mathsf{sbskl}.\mathcal{fsk}$.
- Compute $\mathsf{cert}_i \leftarrow i\mathcal{C}ert(\mathcal{fsk}_i)$ for every $i \in [m]$.
- Output $\mathsf{sbskl.cert} := (\mathsf{cert}_i)_{i\in[m]}$.

$\mathsf{sbSKL.Vrfy}(\mathsf{sbskl.vk}, \mathsf{sbskl.cert})$**:**

- Parse $(\mathsf{vk}_i)_{i\in[m]} \leftarrow \mathsf{sbskl.vk}$ and $(\mathsf{cert}_i)_{i\in[m]} \leftarrow \mathsf{sbskl.cert}$.
- Output $\top$ if $\top = \mathsf{iVrfy}(\mathsf{vk}_i, \mathsf{cert}_i)$ for every $i \in [m]$, and otherwise $\bot$.

We show the correctness of $\mathsf{SKFE\text{-}sbSKL}$. Let $\mathsf{sbskl}.\mathcal{fsk} := (\mathcal{fsk}_i)_{i\in[m]}$ be a decryption key for f and let $\mathsf{sbskl.ct} := (\mathsf{ct}_i)_{i\in[m]}$ be a ciphertext of x. From the correctness of $\mathsf{iSKFE\text{-}sbSKL}$, we obtain $f_i(s_i)$ by decrypting ct_i with $\mathcal{fsk}_i$ for every $i \in [m]$, where $(f_i)_{i\in[m]} \leftarrow \mathsf{FuncEncode}(\mathsf{params}, f)$ and $(s_i)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x)$. Thus, we obtains $f(x) \leftarrow \mathsf{Decode}((f_i(s_i))_{i\in[m]})$ from the correctness of SetHSS. It is clear that $\mathsf{SKFE\text{-}sbSKL}$ also satisfies verification correctness.

Also, the weak optimal efficiency of $\mathsf{SKFE\text{-}sbSKL}$ easily follows from that of $\mathsf{iSKFE\text{-}sbSKL}$ since the running time of algorithms of SetHSS is independent of n. Note that $\mathsf{sbSKL.Enc}$ samples indices from $[N] = [n/p]$, but it can be done in time $\log n$.

For security, we have the following theorems.

Theorem 5.1. *If* $\mathsf{iSKFE\text{-}sbSKL}$ *is a selectively strong lessor secure index-based SKFE-sbSKL scheme and* SetHSS *is a set homomorphic secret sharing scheme, and* SKE *is a CPA secure SKE scheme, then* $\mathsf{SKFE\text{-}sbSKL}$ *is selectively strong lessor secure.*

Proof of Theorem 5.1. We define a sequence of hybrid games to prove the theorem.

Hyb_0**:** This is the same as $\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathcal{A},\mathsf{SKFE\text{-}sbSKL}}(1^\lambda, 0)$.

1. At the beginning, $\mathcal{A}$ sends $(1^q, n, x_0^*, x_1^*)$ to the challenger. The challenger generates $\mathsf{params} := (p, \ell, (T_i)_{i\in[m]}) \leftarrow \mathsf{SetGen}(1^\lambda)$, $\mathsf{msk}_i \leftarrow \mathsf{iSetup}(1^\lambda, 1^q, N)$ for every $i \in [m]$, and $K \leftarrow \{0,1\}^\lambda$, where $N = n/p$. Throughout the experiment, $\mathcal{A}$ can access the following oracles.

   $O_{\mathsf{Enc}}(x^k)$**:** Given the k-th query x^k, it returns $\mathsf{sbskl.ct}^k$ generated as follows.

   - Generate $(s_i^k)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x^k)$.
   - Generate $j_i^k \leftarrow [N]$ for every $i \in [m]$.
   - Generate $\mathsf{ct}_i^k \leftarrow \mathsf{iEnc}(\mathsf{msk}, i, j_i^k, (s_i^k, \mathbf{0}, 0))$ for every $i \in [m]$.
   - Set $\mathsf{sbskl.ct}^k := (\mathsf{ct}_i^k)_{i\in[m]}$.

   $O_{\mathcal{KG}}(f)$**:** Given f, it generates $\mathsf{sbskl}.\mathit{fsk}$ and $\mathsf{sbskl.vk}$ as follows.

   - Generate $(f_i)_{i\in[m]} \leftarrow \mathsf{FuncEncode}(\mathsf{params}, f)$.
   - Generate $\mathsf{sct}_i \leftarrow \mathsf{E}(K, \mathbf{0})$ for every $i \in [m]$.
   - Generate $(\mathit{fsk}_i, \mathsf{vk}_i) \leftarrow i\mathcal{KG}(\mathsf{msk}_i, F[f_i, \mathsf{sct}_i])$ for every $i \in [m]$.
   - Set $\mathsf{sbskl}.\mathit{fsk} := (\mathit{fsk}_i)_{i\in[m]}$ and $\mathsf{sbskl.vk} := (\mathsf{vk}_i)_{i\in[m]}$.

   It sends $\mathsf{sbskl}.\mathit{fsk}$ to $\mathcal{A}$ and adds $(f, \mathsf{sbskl.vk}, \bot)$ to $L_{\mathcal{KG}}$.

   $O_{\mathsf{Vrfy}}(f, \mathsf{cert} := (\mathsf{cert}_i)_{i\in[m]})$**:** Given $(f, \mathsf{cert} := (\mathsf{cert}_i)_{i\in[m]})$, it finds an entry (f, vk, M) from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) If $\top = \mathsf{Vrfy}(\mathsf{vk}_i, \mathsf{cert}_i)$ for every $i \in [m]$, it returns $\top$ and updates the entry into $(f, \mathsf{vk}, \top)$. Otherwise, it returns $\bot$.

2. When $\mathcal{A}$ requests the challenge ciphertext, the challenger checks if for any entry (f, vk, M) in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$, and the number of queries to O_{Enc} at this point is less than n. If so, the challenger sends $\mathsf{sbskl.ct}^*$ computed as follows to $\mathcal{A}$.

   - Generate $(s_i^*)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x_0^*)$.
   - Generate $j_i^* \leftarrow [N]$ for every $i \in [m]$.
   - Generate $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, (s_i^*, \mathbf{0}, 0))$ for every $i \in [m]$.
   - Set $\mathsf{sbskl.ct}^* := (\mathsf{ct}_i^*)_{i\in[m]}$.

   Otherwise, the challenger outputs 0. Hereafter, $\mathcal{A}$ is not allowed to sends a function f such that $f(x_0^*) \neq f(x_1^*)$ to $O_{\mathcal{KG}}$.

3. $\mathcal{A}$ outputs coin'. The challenger outputs coin' as the final output of the experiment.

Below, we call $i \in [m]$ *a secure instance index* if $j_i^* \neq j_i^k$ holds for every $k \in [n]$. We also call $i \in [m]$ *an insecure instance index* if it is not a secure instance index. Let $S_{\mathtt{secure}} \subseteq [m]$ be the set of secure instance indices, and $S_{\mathtt{insecure}} = S \setminus S_{\mathtt{secure}}$. Since each j_i^k is sampled from $[N] = [n/p]$, for each $i \in [m]$, i is independently included in $S_{\mathtt{insecure}}$ with probability at most $n/N = p$. Then, from the existence of unmarked element property of SetHSS, without negligible probability, there exists $e \in [\ell]$ such that $e \notin \bigcup_{i\in S_{\mathtt{insecure}}} T_i$. Below, for simplicity, we assume that there always exists at least one such instance index, and we denote it as e^*.

Hyb_1**:** This is the same as Hyb_0 except that we generate j_i^k for every $i \in [m]$ and $k \in [n]$ and j_i^* for every $i \in [m]$ at the beginning of the experiment. Note that by this change, secure instance indices and i^* are determined at the beginning of the experiment.

$|\Pr[\mathsf{Hyb}_0 = 1] - \Pr[\mathsf{Hyb}_1 = 1]| = 0$ holds since the change at this step is only conceptual.

Hyb_2**:** This is the same as Hyb_1 except that when $\mathcal{A}$ makes a query f to $O_{\mathcal{KG}}$, if $f(x_0^*) = f(x_1^*)$, it generates sct_i as $\mathsf{sct}_i \leftarrow \mathsf{E}(K, f_i(s_i^*))$ for every $i \in S_{\mathtt{secure}}$.

$|\Pr[\mathsf{Hyb}_1 = 1] - \Pr[\mathsf{Hyb}_2 = 1]| = \mathsf{negl}(\lambda)$ holds from the security of SKE.

Hyb_3**:** This is the same as Hyb_2 except that the challenger generates ct_i^* as $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i^*, (\mathbf{0}, K, 1))$ for every $i \in S_{\mathtt{secure}}$.

$|\Pr[\mathsf{Hyb}_2 = 1] - \Pr[\mathsf{Hyb}_3 = 1]| = \mathsf{negl}(\lambda)$ holds from the selective lessor security of $\mathsf{iSKFE\text{-}sbSKL}$. We provide the proof of it in Proposition 5.2.

Hyb_4**:** This is the same as Hyb_3 except that the challenger generates $(s_i^*)_{i\in[m]}$ as $(s_i^*)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x_1^*)$.

$|\Pr[\mathsf{Hyb}_3 = 1] - \Pr[\mathsf{Hyb}_4 = 1]| = \mathsf{negl}(\lambda)$ holds from the selective indistinguishability-security of SetHSS. We provide the proof of it in Proposition 5.3.

Hyb_5**:** This is the same as Hyb_4 except that we undo the changes from Hyb_0 to Hyb_3. This is the same experiment as $\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathcal{A},\mathsf{SKFE\text{-}sbSKL}}(1^\lambda, 1)$.

$|\Pr[\mathsf{Hyb}_4 = 1] - \Pr[\mathsf{Hyb}_5 = 1]| = \mathsf{negl}(\lambda)$ holds from the security of SKE and $\mathsf{iSKFE\text{-}sbSKL}$.

Proposition 5.2. $|\Pr[\mathsf{Hyb}_2 = 1] - \Pr[\mathsf{Hyb}_3 = 1]| = \mathsf{negl}(\lambda)$ *holds if* $\mathsf{iSKFE\text{-}sbSKL}$ *is selectively lessor secure.*

Proof of Proposition 5.2. We define intermediate experiments $\mathsf{Hyb}_{2,i'}$ between Hyb_2 and Hyb_3 for $i' \in [m]$.

$\mathsf{Hyb}_{2,i'}$**:** This is the same as Hyb_2 except that the challenger generates ct_i^* as $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i^*, (\mathbf{0}, K, 1))$ for every i such that $i \in S_{\mathtt{secure}}$ and $i \leq i'$.

Then, we have

$$|\Pr[\mathsf{Hyb}_2 = 1] - \Pr[\mathsf{Hyb}_3 = 1]| \leq \sum_{i' \in m} \left|\Pr\left[\mathsf{Hyb}_{2,i'-1} = 1 \wedge i' \in S_{\mathtt{secure}}\right] - \Pr\left[\mathsf{Hyb}_{2,i} = 1 \wedge i' \in S_{\mathtt{secure}}\right]\right|, \tag{1}$$

where we define $\mathsf{Hyb}_{2,0} = \mathsf{Hyb}_2$ and $\mathsf{Hyb}_{2,m} = \mathsf{Hyb}_3$. To estimate each term of Equation (1), we construct the following adversary $\mathcal{B}$ that attacks selective lessor security of $\mathsf{iSKFE\text{-}sbSKL}$.

1. $\mathcal{B}$ executes $\mathcal{A}$ and obtains $(1^q, n, x_0^*, x_1^*)$. $\mathcal{B}$ generates $\mathsf{params} := (p, \ell, (T_i)_{i\in[m]}) \leftarrow \mathsf{SetGen}(1^\lambda)$. $\mathcal{B}$ generates $j_i^k \leftarrow [N]$ for every $i \in [m]$ and $k \in [n]$ and $j_i^* \leftarrow [N]$ for every $i \in [m]$, and identifies $S_{\mathtt{secure}}$ and $S_{\mathtt{insecure}}$, where $N = n/p$. If $i' \notin S_{\mathtt{secure}}$, $\mathcal{B}$ aborts with output 0. Otherwise, $\mathcal{B}$ behaves as follows. Below, we let $S_{\mathtt{secure},<i'} = S_{\mathtt{secure}} \cap [i'-1]$. $\mathcal{B}$ computes $(s_i^*)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x_0^*)$. $\mathcal{B}$ also generates $K \leftarrow \{0,1\}^\lambda$. $\mathcal{B}$ sends $(1^q, N, j_{i'}^*, (s_{i'}^*, \mathbf{0}, 0), (\mathbf{0}, K, 1))$. $\mathcal{B}$ also generates $\mathsf{msk}_i \leftarrow \mathsf{iSetup}(1^\lambda, 1^q, N)$ for every $i \in [m] \setminus \{i'\}$. $\mathcal{B}$ simulates oracles for $\mathcal{A}$ as follows.

   $O_{\mathsf{Enc}}(x^k)$**:** Given the k-th query x^k, $\mathcal{B}$ returns $\mathsf{sbskl.ct}^k$ generated as follows.

   - Generate $(s_i^k)_{i\in[m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x^k)$.
   - If $k \leq n$, use $(j_i^k)_{i\in[m]}$ generated at the beginning. Otherwise, Generate $j_i^k \leftarrow [N]$ for every $i \in [m]$.
   - Query $(j_{i'}^k, (s_{i'}^k, \mathbf{0}, 0))$ to its encryption oracle and obtain $\mathsf{ct}_{i'}^k$.
   - Generate $\mathsf{ct}_i^k \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i^k, (s_i^k, \mathbf{0}, 0))$ for every $i \in [m] \setminus \{i'\}$.
   - Set $\mathsf{sbskl.ct}^k := (\mathsf{ct}_i^k)_{i\in[m]}$.

   $O_{\mathcal{KG}}(f)$**:** Given f, $\mathcal{B}$ returns $\mathsf{sbskl}.\mathit{fsk}$ computed as follows.

   - Generate $(f_i)_{i\in[m]} \leftarrow \mathsf{FuncEncode}(\mathsf{params}, f)$.

- Generate $\mathsf{sct}_i \leftarrow \mathsf{E}(K, \mathbf{0})$ for every $i \in S_{\mathtt{insecure}}$. Generate also $\mathsf{sct}_i \leftarrow \mathsf{E}(K, f_i(s_i^*))$ for every $i \in S_{\mathtt{secure}}$ if $f(x_0^*) = f(x_1^*)$, and otherwise generate $\mathsf{sct}_i \leftarrow \mathsf{E}(K, \mathbf{0})$ for every $i \in S_{\mathtt{secure}}$.
- Query $F[f_{i'}, \mathsf{sct}_{i'}]$ to its key generation oracle and obtain $(\mathit{fsk}_{i'}, \mathsf{vk}_{i'})$.
- Generate $(\mathit{fsk}_i, \mathsf{vk}_i) \leftarrow i\mathcal{KG}(\mathsf{msk}_i, F[f_i, \mathsf{sct}_i])$ for every $i \in [m] \setminus \{i'\}$.
- Set $\mathsf{sbskl}.\mathit{fsk} := (\mathit{fsk}_i)_{i \in [m]}$.

Also, $\mathcal{B}$ adds $(f, (\mathsf{vk}_i)_{i \in [m] \setminus \{i'\}}, \bot)$ to $L_{\mathcal{KG}}$.

$O_{\mathsf{Vrfy}}(f, \mathsf{cert} := (\mathsf{cert}_i)_{i \in [m]})$**:** Given $(f, \mathsf{cert} := (\mathsf{cert}_i)_{i \in [m]})$, it finds an entry $(f, (\mathsf{vk}_i)_{i \in [m] \setminus \{i'\}}, \bot)$ from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) $\mathcal{B}$ sends $(f, \mathsf{cert}_{i'})$ to its verification oracle and obtains $M_{i'}$. If $M = \top$ and $\top = \mathsf{Vrfy}(\mathsf{vk}_i, \mathsf{cert}_i)$ for every $i \in [m] \setminus \{i'\}$, $\mathcal{B}$ returns $\top$ and updates the entry into $(f, (\mathsf{vk}_i)_{i \in [m] \setminus \{i'\}}, \top)$. Otherwise, $\mathcal{B}$ returns $\bot$.

2. When $\mathcal{A}$ requests the challenge ciphertext, $\mathcal{B}$ checks if for any entry $(f, (\mathsf{vk}_i)_{i \in [m] \setminus \{i'\}}, M)$ in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$. If so, $\mathcal{B}$ requests the challenge ciphertext to its challenger and obtains $\mathsf{ct}_{i'}^*$. $\mathcal{B}$ also generates $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i^*, (\mathbf{0}, K, 1))$ for every $i \in S_{\mathtt{secure},<i'}$ and $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i^*, (s_i^*, \mathbf{0}, 0))$ for every $i \in [m] \setminus (S_{\mathtt{secure},<i'} \cup \{i'\})$. $\mathcal{B}$ sends $\mathsf{sbskl.ct} := (\mathsf{ct}_i^*)_{i \in [m]}$ to $\mathcal{A}$. Hereafter, $\mathcal{B}$ rejects $\mathcal{A}$'s query f to $O_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$.
3. When $\mathcal{A}$ outputs coin', $\mathcal{B}$ outputs coin'.

$\mathcal{B}$ simulates $\mathsf{Hyb}_{2,i'-1}$ (resp. $\mathsf{Hyb}_{2,i'}$) if $\mathcal{B}$ runs in $\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathcal{B},\mathsf{SKFE\text{-}sbSKL}}(1^\lambda, 0)$ (resp. $\mathsf{Exp}^{\mathsf{sel\text{-}s\text{-}lessor}}_{\mathcal{B},\mathsf{SKFE\text{-}sbSKL}}(1^\lambda, 1)$) and $i' \in S_{\mathtt{secure}}$. This completes the proof. ■

Proposition 5.3. $|\Pr[\mathsf{Hyb}_3 = 1] - \Pr[\mathsf{Hyb}_4 = 1]| = \mathsf{negl}(\lambda)$ *holds if* SetHSS *is a set homomorphic secret sharing.*

Proof of Proposition 5.3. We construct the following adversary $\mathcal{B}$ that attacks the selective indistinguishability-security of SetHSS.

1. Given $\mathsf{params} := (p, \ell, (T_i)_{i \in [m]})$, $\mathcal{B}$ executes $\mathcal{A}$ and obtains $(1^q, n, x_0^*, x_1^*)$. $\mathcal{B}$ generates $j_i^k \leftarrow [N]$ for every $i \in [m]$ and $k \in [n]$ and $j_i^* \leftarrow [N]$ for every $i \in [m]$, and identifies $S_{\mathtt{secure}}$, $S_{\mathtt{insecure}}$, and the unmarked element e^*, where $N = n/p$. $\mathcal{B}$ sends (e^*, x_0^*, x_1^*) to the challenger and obtains $(s_i^*)_{i \in [m]_{e^* \notin}}$, where $[m]_{e^* \notin}$ denotes the subset of $[m]$ consisting of i such that $e^* \notin T_i$. $\mathcal{B}$ also generates $\mathsf{msk}_i \leftarrow \mathsf{iSetup}(1^\lambda, 1^q, N)$ for every $i \in [m]$ and $K \leftarrow \{0,1\}^\lambda$. $\mathcal{B}$ simulates oracles for $\mathcal{A}$ as follows.

$O_{\mathsf{Enc}}(x^k)$**:** Given the k-th query x^k, $\mathcal{B}$ returns $\mathsf{sbskl.ct}^k$ generated as follows.

- Generate $(s_i^k)_{i \in [m]} \leftarrow \mathsf{InpEncode}(\mathsf{params}, x^k)$.
- If $k \leq n$, use $(j_i^k)_{i \in [m]}$ generated at the beginning. Otherwise, Generate $j_i^k \leftarrow [N]$ for every $i \in [m]$.
- Generate $\mathsf{ct}_i^k \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i^k, (s_i^k, \mathbf{0}, 0))$ for every $i \in [m]$.
- Set $\mathsf{sbskl.ct}^k := (\mathsf{ct}_i^k)_{i \in [m]}$.

$O_{\mathcal{KG}}(f)$**:** Given f, $\mathcal{B}$ returns $\mathsf{sbskl}.\mathit{fsk}$ computed as follows.

- Queries f to its function encode oracle and obtain $(f_i, y_i := f_i(s_i^*))_{i \in [m]})$ if $f(x_0^*) = f(x_1^*)$. Otherwise, compute $(f_i)_{i \in [m]} \leftarrow \mathsf{FuncEncode}(\mathsf{params}, f)$.
- Generate $\mathsf{sct}_i \leftarrow \mathsf{E}(K, \mathbf{0})$ for every $i \in S_{\mathtt{insecure}}$. Generate also $\mathsf{sct}_i \leftarrow \mathsf{E}(K, f_i(s_i^*))$ for every $i \in S_{\mathtt{secure}}$ if $f(x_0^*) = f(x_1^*)$, and otherwise generate $\mathsf{sct}_i \leftarrow \mathsf{E}(K, \mathbf{0})$ for every $i \in S_{\mathtt{secure}}$.

- Generate $(\mathit{fsk}_i, \mathsf{vk}_i) \leftarrow i\mathcal{KG}(\mathsf{msk}_i, F[f_i, \mathsf{sct}_i])$ for every $i \in [m]$.
- Set $\mathsf{sbskl}.\mathit{fsk} := (\mathit{fsk}_i)_{i\in[m]}$.

Also, $\mathcal{B}$ adds $(f, (\mathsf{vk}_i)_{i\in[m]}, \bot)$ to $L_{\mathcal{KG}}$.

$O_{\mathsf{Vrfy}}(f, \mathsf{cert} := (\mathsf{cert}_i)_{i\in[m]})$**:** Given $(f, \mathsf{cert} := (\mathsf{cert}_i)_{i\in[m]})$, it finds an entry $(f, (\mathsf{vk}_i)_{i\in[m]}, \bot)$ from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) If $\top = \mathsf{Vrfy}(\mathsf{vk}_i, \mathsf{cert}_i)$ for every $i \in [m]$ and the number of queries to O_{Enc} at this point is less than n, $\mathcal{B}$ returns $\top$ and updates the entry into $(f, (\mathsf{vk}_i)_{i\in[m]}, \top)$. Otherwise, $\mathcal{B}$ returns $\bot$.

2. When $\mathcal{A}$ requests the challenge ciphertext, $\mathcal{B}$ checks if for any entry $(f, (\mathsf{vk}_i)_{i\in[m]\setminus\{i'\}}, M)$ in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$. If so, $\mathcal{B}$ generates $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i^*, (\mathbf{0}, K, 1))$ for every $i \in S_{\mathtt{secure}}$ and $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, j_i^*, (s_i^*, \mathbf{0}, 0))$ for every $i \in S_{\mathtt{insecure}}$, and $\mathcal{B}$ sends $\mathsf{sbskl.ct} := (\mathsf{ct}_i^*)_{i\in[m]}$ to $\mathcal{A}$. Otherwise, $\mathcal{B}$ outputs 0 and terminates. Hereafter, $\mathcal{B}$ rejects $\mathcal{A}$'s query f to $O_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$.
3. When $\mathcal{A}$ outputs coin', $\mathcal{B}$ outputs coin'.

$\mathcal{B}$ simulates Hyb_3 (resp. Hyb_4) if $\mathcal{B}$ runs in $\mathsf{Exp}^{\mathsf{sel\text{-}ind}}_{\mathsf{SetHSS},\mathcal{B}}(1^\lambda, 0)$ (resp. $\mathsf{Exp}^{\mathsf{sel\text{-}ind}}_{\mathsf{SetHSS},\mathcal{B}}(1^\lambda, 1)$). This completes the proof. ■

From the above discussions, SKFE-sbSKL satisfies selective strong lessor security. ■

Remark 5.4 (Difference from FE security amplification). A savvy reader notices that although we use the technique used in the FE security amplification by Jain et al. [JKMS20], we do not use their probabilistic replacement theorem [JKMS20, Theorem 7.1 in eprint ver.] and the nested construction [JKMS20, Section 9 in eprint ver.] in the proofs of Theorem 5.1. We do not need them for our purpose due to the following reason.

Jain et al. need the nested construction to achieve a secure FE scheme whose adversary's advantage is less than $1/6$ from one whose adversary's advantage is any constant $\epsilon \in (0, 1)$. We do not need the nested construction since we can start with a secure construction whose adversary's advantage is less than $1/6$ by setting a large index space in the index-based construction.

Jain et al. need the probabilistic replacement theorem due to the following reason. We do not know which FE instance is secure at the beginning of the FE security game in the security amplification context, while the adversary in set homomorphic secret sharing must declare the index of a secure instance at the beginning. In our case, whether each index-based FE instance is secure or not depends on whether randomly sampled indices collide or not. In addition, we can sample all indices used in the security game at the beginning of the game, and a secure FE instance is fixed at the beginning. Thus, we can apply the security of set homomorphic secret sharing without the probabilistic replacement theorem.

6 SKFE with Secure Key Leasing

We construct an SKFE-SKL scheme $\mathsf{SKFE\text{-}SKL} = (\mathsf{SKL.Setup}, \mathsf{SKL}.\mathcal{KG}, \mathsf{SKL.Enc}, \mathsf{SKL}.\mathcal{Dec}, \mathsf{SKL}.\mathcal{Cert}, \mathsf{SKL.Vrfy})$ from an SKFE-sbSKL scheme $\mathsf{SKFE\text{-}sbSKL} = (\mathsf{sbSKL.Setup}, \mathsf{sbSKL}.\mathcal{KG}, \mathsf{sbSKL.Enc}, \mathsf{sbSKL}.\mathcal{Dec}, \mathsf{sbSKL}.\mathcal{Cert}, \mathsf{sbSKL.Vrfy})$. The description of SKFE-SKL is as follows.

$\mathsf{SKL.Setup}(1^\lambda, 1^q)$**:**

- Generate $\mathsf{msk}_k \leftarrow \mathsf{sbSKL.Setup}(1^\lambda, 1^q, 2^k)$ for every $k \in [\lambda]$.
- Output $\mathsf{skl.msk} := (\mathsf{msk}_k)_{k\in[\lambda]}$.

$\mathsf{SKL}.\mathcal{KG}(\mathsf{skl.msk}, f, 1^n)$**:**

- Parse $(\mathsf{msk}_k)_{k\in[\lambda]} \leftarrow \mathsf{skl.msk}$.
- Compute k' such that $2^{k'-1} \le n \le 2^{k'}$.
- Generate $(\mathit{fsk}_{k'}, \mathsf{vk}_{k'}) \leftarrow \mathsf{sbSKL}.\mathcal{KG}(\mathsf{msk}_{k'}, f)$.
- Output $\mathsf{skl}.\mathit{fsk} := (k', \mathit{fsk}_{k'})$ and $\mathsf{vk}_{k'}$.

$\mathsf{SKL.Enc}(\mathsf{skl.msk}, x)$:

- Parse $(\mathsf{msk}_k)_{k\in[\lambda]} \leftarrow \mathsf{skl.msk}$.
- Generate $\mathsf{ct}_k \leftarrow \mathsf{sbSKL.Enc}(\mathsf{msk}_k, x)$ for every $k \in [\lambda]$.
- Output $\mathsf{skl.ct} := (\mathsf{ct}_k)_{k\in[\lambda]}$.

$\mathsf{SKL}.\mathcal{Dec}(\mathsf{skl}.\mathit{sk}_f, \mathsf{skl.ct})$:

- Parse $(k', \mathit{fsk}_{k'}) \leftarrow \mathsf{skl}.\mathit{fsk}$ and $(\mathsf{ct}_k)_{k\in[\lambda]} \leftarrow \mathsf{skl.ct}$.
- Output $y \leftarrow \mathsf{sbSKL}.\mathcal{Dec}(\mathit{fsk}_{k'}, \mathsf{ct}_{k'})$.

$\mathsf{SKL}.\mathcal{Cert}(\mathsf{skl}.\mathit{sk}_f)$:

- Parse $(k', \mathit{fsk}_{k'}) \leftarrow \mathsf{skl}.\mathit{sk}_f$.
- Output $\mathsf{cert} \leftarrow \mathsf{sbSKL}.\mathcal{Cert}(\mathit{fsk}_{k'})$.

$\mathsf{SKL.Vrfy}(\mathsf{vk}, \mathsf{cert})$:

- Output $\top/\bot \leftarrow \mathsf{sbSKL.Vrfy}(\mathsf{vk}, \mathsf{cert})$.

The correctness of SKFE-SKL follows from that of SKFE-sbSKL. Also, we can confirm that all algorithms of SKFE-SKL run in polynomial time since sbSKL.Setup and sbSKL.Enc of SKFE-sbSKL run in polynomial time even for the availability bound 2^λ due to its weak optimal efficiency. For security, we have the following theorem.

Theorem 6.1. *If* SKFE-sbSKL *satisfies selective strong lessor security, then* SKFE-SKL *satisfies selective lessor security.*

Proof of Theorem 6.1. We define a sequence of hybrid games to prove the theorem.

Hyb_0: This is the same as $\mathsf{Exp}^{\mathsf{sel\text{-}lessor}}_{\mathcal{A},\mathsf{SKFE\text{-}SKL}}(1^\lambda, 0)$.

1. At the beginning, $\mathcal{A}$ sends $(1^q, x_0^*, x_1^*)$ to the challenger. The challenger runs $\mathsf{msk}_k \leftarrow \mathsf{sbSKL.Setup}(1^\lambda, 1^q, 2^k)$ for every $k \in [\lambda]$. Throughout the experiment, $\mathcal{A}$ can access the following oracles.

   $O_{\mathsf{Enc}}(x)$: Given x, it generates $\mathsf{ct}_k \leftarrow \mathsf{sbSKL.Enc}(\mathsf{msk}_k, x)$ for every $k \in [\lambda]$ and returns $\mathsf{skl.ct} := (\mathsf{ct}_k)_{k\in[\lambda]}$.

   $O_{\mathcal{KG}}(f, 1^n)$: Given $(f, 1^n)$, it computes k such that $2^{k-1} \le n \le 2^k$, generates $(\mathit{fsk}_k, \mathsf{vk}_k) \leftarrow \mathsf{sbSKL}.\mathcal{KG}(\mathsf{msk}_k, f)$, and sets $\mathsf{skl}.\mathit{fsk} := (k, \mathit{fsk}_k)$. It returns $\mathsf{skl}.\mathit{fsk}$ to $\mathcal{A}$ and adds $(f, 1^n, \mathsf{vk}_k, \bot)$ to $L_{\mathcal{KG}}$. $\mathcal{A}$ can access this oracle at most q times.

   $O_{\mathsf{Vrfy}}(f, \mathsf{cert})$: Given (f, cert), it finds an entry $(f, 1^n, \mathsf{vk}, M)$ from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) If $\top = \mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$ and the number of queries to O_{Enc} at this point is less than n, it returns $\top$ and updates the entry into $(f, 1^n, \mathsf{vk}, \top)$. Otherwise, it returns $\bot$.

2. When $\mathcal{A}$ requests the challenge ciphertext, the challenger checks if for any entry $(f, 1^n, \mathsf{vk}, M)$ in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$. If so, the challenger generates $\mathsf{ct}_k^* \leftarrow \mathsf{sbSKL.Enc}(\mathsf{msk}_k, x_0^*)$ for every $k \in [\lambda]$ and sends $\mathsf{skl.ct}^* := (\mathsf{ct}_k^*)_{k \in [\lambda]}$ to $\mathcal{A}$. Otherwise, the challenger outputs 0. Hereafter, $\mathcal{A}$ is not allowed to sends a function f such that $f(x_0^*) \neq f(x_1^*)$ to $O_{\mathcal{KG}}$.
3. $\mathcal{A}$ outputs a guess coin' for coin. The challenger outputs coin' as the final output of the experiment.

We define $\mathsf{Hyb}_{k'}$ for every $k' \in [\lambda]$.

$\mathsf{Hyb}_{k'}$**:** This hybrid is the same as $\mathsf{Hyb}_{k'-1}$ except that $\mathsf{ct}_{k'}^*$ is generated as $\mathsf{ct}_{k'}^* \leftarrow \mathsf{Enc}(\mathsf{msk}_{k'}, x_1^*)$.

Hyb_λ is exactly the same experiment as $\mathsf{Exp}^{\mathsf{sel\text{-}lessor}}_{\mathcal{A},\mathsf{SKFE\text{-}SKL}}(1^\lambda, 1)$.

For every $k' \in [\lambda]$, we let $\mathtt{SUC}_{k'}$ be the event that the output of the experiment $\mathsf{Hyb}_{k'}$ is 1. Then, we have

$$\mathsf{Adv}^{\mathsf{sel\text{-}lessor}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}(\lambda) = |\Pr[\mathsf{Hyb}_0 = 1] - \Pr[\mathsf{Hyb}_\lambda = 1]| \leq \sum_{k'=1}^{\lambda} |\Pr[\mathtt{SUC}_{k'-1}] - \Pr[\mathtt{SUC}_{k'}]|.$$

Proposition 6.2. *It holds that* $|\Pr[\mathsf{Hyb}_{k'-1} = 1] - \Pr[\mathsf{Hyb}_{k'} = 1]| = \mathsf{negl}(\lambda)$ *for every* $k' \in [\lambda]$ *if* $\mathsf{SKFE\text{-}sbSKL}$ *is selectively lessor secure.*

Proof of Proposition 6.2. We construct the following adversary $\mathcal{B}$ that attacks selective lessor security of $\mathsf{SKFE\text{-}sbSKL}$ with respect to $\mathsf{msk}_{k'}$.

1. $\mathcal{B}$ executes $\mathcal{A}$ and obtains $(1^q, x_0^*, x_1^*)$ from $\mathcal{A}$. $\mathcal{B}$ sends $(1^q, x_0^*, x_1^*, 2^{k'})$ to the challenger. $\mathcal{B}$ generates $\mathsf{msk}_k \leftarrow \mathsf{sbSKL.Setup}(1^\lambda, 1^q, 2^k)$ for every $k \in [\lambda] \setminus \{k'\}$. $\mathcal{B}$ simulates queries made by $\mathcal{A}$ as follows.

   $O_{\mathsf{Enc}}(x)$**:** Given x, $\mathcal{B}$ generates $\mathsf{ct}_k \leftarrow \mathsf{sbSKL.Enc}(\mathsf{msk}_k, x)$ for every $k \in [\lambda] \setminus \{k'\}$. $\mathcal{B}$ also queries x to its encryption oracle and obtains $\mathsf{ct}_{k'}$. $\mathcal{B}$ returns $\mathsf{skl.ct} := (\mathsf{ct}_k)_{k \in [\lambda]}$.

   $O_{\mathcal{KG}}(f, 1^n)$**:** Given $(f, 1^n)$, $\mathcal{B}$ computes k such that $2^{k-1} \leq n \leq 2^k$. If $k \neq k'$, $\mathcal{B}$ generates $(\mathit{fsk}_k, \mathsf{vk}_k) \leftarrow \mathsf{sbSKL}.\mathcal{KG}(\mathsf{msk}_k, f)$, and otherwise $\mathcal{B}$ queries f to its key generation oracle and obtains fsk_k and sets $\mathsf{vk}_k := \bot$. $\mathcal{B}$ returns $\mathsf{skl}.\mathit{fsk} := \mathit{fsk}_k$. $\mathcal{B}$ adds $(f, 1^n, \mathsf{vk}_k, \bot)$ to $L_{\mathcal{KG}}$.

   $O_{\mathsf{Vrfy}}(f, \mathsf{cert})$**:** Given (f, cert), it finds an entry $(f, 1^n, \mathsf{vk}, M)$ from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) If $\mathsf{vk} = \bot$, $\mathcal{B}$ sends cert to its verification oracle and obtains M, and otherwise it computes $M = \mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$. If $M = \top$ and the number of queries to O_{Enc} at this point is less than n, it returns $\top$ and updates the entry into $(f, 1^n, \mathsf{vk}, \top)$. Otherwise, it returns $\bot$.

2. When $\mathcal{A}$ requests the challenge ciphertext, the challenger checks if for any entry $(f, 1^n, \mathsf{vk}, M)$ in $L_{\mathcal{KG}}$ such that $f(x_0^*) \neq f(x_1^*)$, it holds that $M = \top$. If so, $\mathcal{B}$ requests the challenge ciphertext to its challenger and obtains $\mathsf{ct}_{k'}^*$, generates $\mathsf{ct}_k^* \leftarrow \mathsf{sbSKL.Enc}(\mathsf{msk}_k, x_1^*)$ for every $1 \leq k < k'$ and $\mathsf{ct}_k^* \leftarrow \mathsf{sbSKL.Enc}(\mathsf{msk}_k, x_0^*)$ for every $k' < k \leq \lambda$, and sends $\mathsf{skl.ct}^* := (\mathsf{ct}_k^*)_{k \in [\lambda]}$ to $\mathcal{A}$. Otherwise, the challenger outputs 0. Hereafter, $\mathcal{A}$ is not allowed to sends a function f such that $f(x_0^*) \neq f(x_1^*)$ to $O_{\mathcal{KG}}$.
3. When $\mathcal{A}$ outputs coin', $\mathcal{B}$ outputs coin' and terminates.

$\mathcal{B}$ simulates $\mathsf{Hyb}_{k'-1}$ (resp. $\mathsf{Hyb}_{k'}$) for $\mathcal{A}$ if $\mathcal{B}$ runs in $\mathsf{Exp}^{\mathsf{sel\text{-}lessor}}_{\mathcal{B},\mathsf{SKFE\text{-}sbSKL}}(1^\lambda, 0)$ (resp. $\mathsf{Exp}^{\mathsf{sel\text{-}lessor}}_{\mathcal{B},\mathsf{SKFE\text{-}sbSKL}}(1^\lambda, 1)$.). This completes the proof. ■

From the above discussions, $\mathsf{SKFE\text{-}SKL}$ satisfies selective lessor security. ■

By Theorems 2.2, 2.8, 2.12, 2.16, 4.1, 5.1 and 6.1, we obtain the following theorem.

Theorem 6.3. *If there exist OWFs, there exists selectively lessor secure SKFE-SKL for* P/poly *(in the sense of Definition 3.2).*

Although we describe our results on SKFE-SKL in the bounded collusion-resistant setting, our transformation from standard SKFE to SKFE-SKL also works in the fully collusion-resistant setting. The fully collusion-resistance guarantees that the SKFE scheme is secure even if an adversary accesses the key generation oracle a-priori unbounded times. Namely, if we start with fully collusion-resistant SKFE, we can obtain fully collusion-resistant SKFE-SKL by our transformations.

7 Single-Decryptor Functional Encryption

This section introduces single-decryptor FE (SDFE), whose functional decryption keys are copy-protected.

7.1 Preliminaries for SDFE

Quantum information. We review some basics of qunatum information in this subsection.

Let $\mathcal{H}$ be a finite-dimensional complex Hilbert space. A (pure) quantum state is a vector $|\psi\rangle \in \mathcal{H}$. Let $\mathcal{S}(\mathcal{H})$ be the space of Hermitian operators on $\mathcal{H}$. A density matrix is a Hermitian operator $\mathcal{X} \in \mathcal{S}(\mathcal{H})$ with $\mathrm{Tr}(\mathcal{X}) = 1$, which is a probabilistic mixture of pure states. A quantum state over $\mathcal{H} = \mathbb{C}^2$ is called qubit, which can be represented by the linear combination of the standard basis $\{|0\rangle, |1\rangle\}$. More generally, a quantum system over $(\mathbb{C}^2)^{\otimes n}$ is called an n-qubit quantum system for $n \in \mathbb{N} \setminus \{0\}$.

A Hilbert space is divided into registers $\mathcal{H} = \mathcal{H}^{\mathsf{R}_1} \otimes \mathcal{H}^{\mathsf{R}_2} \otimes \cdots \otimes \mathcal{H}^{\mathsf{R}_n}$. We sometimes write $\mathcal{X}^{\mathsf{R}_i}$ to emphasize that the operator $\mathcal{X}$ acts on register $\mathcal{H}^{\mathsf{R}_i}$.[13] When we apply $\mathcal{X}^{\mathsf{R}_1}$ to registers $\mathcal{H}^{\mathsf{R}_1}$ and $\mathcal{H}^{\mathsf{R}_2}$, $\mathcal{X}^{\mathsf{R}_1}$ is identified with $\mathcal{X}^{\mathsf{R}_1} \otimes \boldsymbol{I}^{\mathsf{R}_2}$.

A unitary operation is represented by a complex matrix $\boldsymbol{U}$ such that $\boldsymbol{U}\boldsymbol{U}^{\dagger} = \boldsymbol{I}$. The operation $\boldsymbol{U}$ transforms $|\psi\rangle$ and $\mathcal{X}$ into $\boldsymbol{U}|\psi\rangle$ and $\boldsymbol{U}\mathcal{X}\boldsymbol{U}^{\dagger}$, respectively. A projector $\boldsymbol{P}$ is a Hermitian operator ($\boldsymbol{P}^{\dagger} = \boldsymbol{P}$) such that $\boldsymbol{P}^2 = \boldsymbol{P}$.

For a quantum state $\mathcal{X}$ over two registers $\mathcal{H}^{\mathsf{R}_1}$ and $\mathcal{H}^{\mathsf{R}_2}$, we denote the state in $\mathcal{H}^{\mathsf{R}_1}$ as $\mathcal{X}[\mathsf{R}_1]$, where $\mathcal{X}[\mathsf{R}_1] = \mathrm{Tr}_2[\mathcal{X}]$ is a partial trace of $\mathcal{X}$ (trace out R_2).

Definition 7.1 (Quantum Program with Classical Inputs and Outputs [ALL+21]). *A quantum program with classical inputs is a pair of quantum state q and unitaries $\{\boldsymbol{U}_x\}_{x\in[N]}$ where $[N]$ is the domain, such that the state of the program evaluated on input x is equal to $\boldsymbol{U}_x q \boldsymbol{U}_x^{\dagger}$. We measure the first register of $\boldsymbol{U}_x q \boldsymbol{U}_x^{\dagger}$ to obtain an output. We say that $\{\boldsymbol{U}_x\}_{x\in[N]}$ has a compact classical description $\boldsymbol{U}$ when applying $\boldsymbol{U}_x$ can be efficiently computed given $\boldsymbol{U}$ and x.*

Definition 7.2 (Positive Operator-Valued Measure). *Let $\mathcal{I}$ be a finite index set. A positive operator valued measure (POVM) $\mathcal{M}$ is a collection $\{\boldsymbol{M}_i\}_{i\in\mathcal{I}}$ of Hermitian positive semi-define matrices $\boldsymbol{M}_i$ such that $\sum_{i\in\mathcal{I}} \boldsymbol{M}_i = \boldsymbol{I}$. When we apply POVM $\mathcal{M}$ to a quantum state $\mathcal{X}$, the measurement outcome is i with probability $p_i = \mathrm{Tr}(\mathcal{X}\boldsymbol{M}_i)$. We denote by $\mathcal{M}(|\psi\rangle)$ the distribution obtained by applying $\mathcal{M}$ to $|\psi\rangle$.*

Definition 7.3 (Quantum Measurement). *A quantum measurement $\mathcal{E}$ is a collection $\{\boldsymbol{E}_i\}_{i\in\mathcal{I}}$ of matrices $\boldsymbol{E}_i$ such that $\sum_{i\in\mathcal{I}} \boldsymbol{E}_i^{\dagger}\boldsymbol{E}_i = \boldsymbol{I}$. When we apply $\mathcal{E}$ to a quantum state $\mathcal{X}$, the measurement outcome is i with probability $p_i = \mathrm{Tr}(\mathcal{X}\boldsymbol{E}_i^{\dagger}\boldsymbol{E}_i)$. Conditioned on the outcome being i, the post-measurement state is $\boldsymbol{E}_i\mathcal{X}\boldsymbol{E}_i^{\dagger}/p_i$.*

We can construct a POVM $\mathcal{M}$ from any quantum measurement $\mathcal{E}$ by setting $\boldsymbol{M}_i := \boldsymbol{E}_i^{\dagger}\boldsymbol{E}_i$. We say that $\mathcal{E}$ is an implementation of $\mathcal{M}$. The implementation of a POVM may not be unique.

[13]The superscript parts are gray colored.

Definition 7.4 (Projective Measurement/POVM). *A quantum measurement $\mathcal{E} = \{\mathbf{E}_i\}_{i\in\mathcal{I}}$ is projective if for all $i \in \mathcal{I}$, $\mathbf{E}_i$ is a projector. This implies that $\mathbf{E}_i\mathbf{E}_j = \mathbf{0}$ for distinct $i, j \in \mathcal{I}$. In particular, two-outcome projective measurement is called a binary projective measurement, and is written as $\mathcal{E} = (\boldsymbol{P}, \boldsymbol{I} - \boldsymbol{P})$, where $\boldsymbol{P}$ is associated with the outcome 1, and $\boldsymbol{I} - \boldsymbol{P}$ with the outcome 0. Similarly, a POVM $\mathcal{M}$ is projective if for all $i \in \mathcal{I}$, $\boldsymbol{M}_i$ is a projector. This also implies that $\boldsymbol{M}_i\boldsymbol{M}_j = \mathbf{0}$ for distinct $i, j \in \mathcal{I}$.*

Definition 7.5 (Projective Implementation). *Let:*

- *$\mathcal{D}$ be a finite set of distributions over an index set $\mathcal{I}$.*
- *$\mathcal{P} = \{\boldsymbol{P}_i\}_{i\in\mathcal{I}}$ be a POVM*
- *$\mathcal{E} = \{\mathbf{E}_D\}_{D\in\mathcal{D}}$ be a projective measurement with index set $\mathcal{D}$.*

We consider the following measurement procedure.

1. *Measure under the projective measurement $\mathcal{E}$ and obtain a distribution D.*
2. *Output a random sample from the distribution D.*

We say $\mathcal{E}$ is the projective implementation of $\mathcal{P}$, denoted by $\mathsf{ProjImp}(\mathcal{P})$, if the measurement process above is equivalent to $\mathcal{P}$.

Theorem 7.6 ([Zha20, Lemma 1]). *Any binary outcome POVM $\mathcal{P} = (\boldsymbol{P}, \boldsymbol{I} - \boldsymbol{P})$ has a unique projective implementation $\mathsf{ProjImp}(\mathcal{P})$.*

Threshold implementation. We review the notion of threshold implementation and related notions since we need them for single decryptor (functional) encryption. This part is mostly taken from the paper by Coladangelo et al. [CLLZ21].

Definition 7.7 (Threshold Implementation [Zha20, ALL+21]). *Let*

- *$\mathcal{P} = (\boldsymbol{P}, \boldsymbol{I} - \boldsymbol{P})$ be a binary POVM*
- *$\mathsf{ProjImp}(\mathcal{P})$ and $\mathcal{E}$ be a projective implementation of $\mathcal{P}$ and the projective measurement in the first step of $\mathsf{ProjImp}(\mathcal{P})$, respectively*
- *$\gamma > 0$.*

A threshold implementation of $\mathcal{P}$, denoted by $\mathcal{TI}_\gamma(\mathcal{P})$, is the following measurement procedure.

- *Apply $\mathcal{E}$ to a quantum state and obtain $(p, 1-p)$ as an outcome.*
- *Output 1 if $p \geq \gamma$, and 0 otherwise.*

For any quantum state q, we denote by $\mathrm{Tr}[\mathcal{TI}_\gamma(\mathcal{P})q]$ the probability that the threshold implementation applied to q outputs 1 as Coladangelo et al. did [CLLZ21]. This means that whenever $\mathcal{TI}_\gamma(\mathcal{P})$ appears inside a trace Tr, we treat $\mathcal{TI}_\gamma(\mathcal{P})$ as a projection onto the 1 outcome.

Lemma 7.8 ([ALL+21]). *Any binary POVM $\mathcal{P} = (\boldsymbol{P}, \boldsymbol{I} - \boldsymbol{P})$ has a threshold implementation $\mathcal{TI}_\gamma(\mathcal{P})$ for any γ.*

Definition 7.9 (Mixture of Projetive Measurement [Zha20]). *Let $D : \mathcal{R} \to \mathcal{I}$ where $\mathcal{R}$ and $\mathcal{I}$ are some sets. Let $\{(\boldsymbol{P}_i, \boldsymbol{Q}_i)\}_{\in\mathcal{I}}$ be a collection of binary projective measurement. The mixture of projective measurements associated to $\mathcal{R}$, $\mathcal{I}$, D, and $\{(\boldsymbol{P}_i, \boldsymbol{Q}_i)\}_{\in\mathcal{I}}$ is the binary POVM $\mathcal{P}_D = (\boldsymbol{P}_D, \boldsymbol{Q}_D)$ defined as follows*

$$\boldsymbol{P}_D = \sum_{i\in\mathcal{I}} \Pr[i \leftarrow D(R)]\boldsymbol{P}_i \qquad\qquad \boldsymbol{Q}_D = \sum_{i\in\mathcal{I}} \Pr[i \leftarrow D(R)]\boldsymbol{Q}_i,$$

where R is uniformly distributed in $\mathcal{R}$.

Theorem 7.10 ([Zha20, ALL$^+$21]). *Let*

- $\gamma > 0$
- $\mathcal{P}$ *be a collection of projective measurements indexed by some sets*
- q *be an efficiently constructible mixed state*
- D_0 *and* D_1 *be two efficiencntly samplable and computationally indistinguishable distributions over* $\mathcal{I}$.

For any inverse polynomial ϵ*, there exists a negligible function* δ *such that*

$$\mathrm{Tr}[\mathcal{TI}_{\gamma-\epsilon}(\mathcal{P}_{D_1})q] \geq \mathrm{Tr}[\mathcal{TI}_{\gamma}(\mathcal{P}_{D_0})q] - \delta,$$

where $\mathcal{P}_{D_{\mathsf{coin}}}$ *is the mixture of projective measurements associated to* $\mathcal{P}$, D_{coin}*, and* $\mathsf{coin} \in \{0,1\}$.

Cryptographic tools. Before we introduce definitions, we introduce a convention. We say that a cryptographic scheme is sub-exponentially secure if there exists some constant $0 < \alpha < 1$ such that for every QPT $\mathcal{A}$ the advantage of the security game is bounded by $O(2^{-\lambda^{\alpha}})$.

Definition 7.11 (Learning with Errors). *Let* $n, m, q \in \mathbb{N}$ *be integer functions of the security parameter* λ*. Let* $\chi = \chi(\lambda)$ *be an error distribution over* $\mathbb{Z}$*. The LWE problem* $LWE_{n,m,q,\chi}$ *is to distinguish the following two distributions.*

$$D_0 := \{(\boldsymbol{A}, \boldsymbol{s}^{\mathsf{T}}\boldsymbol{A} + \boldsymbol{e}) \mid \boldsymbol{A} \leftarrow \mathbb{Z}_q^{n\times m}, \boldsymbol{s} \leftarrow \mathbb{Z}_q^n, \boldsymbol{e} \leftarrow \chi^m\} \text{ and } D_1 := \{(\boldsymbol{A}, \boldsymbol{u}) \mid \boldsymbol{A} \leftarrow \mathbb{Z}_q^{n\times m}, \boldsymbol{u} \leftarrow \mathbb{Z}_q^m\}.$$

When we say we assume the hardness of the LWE problem or the QLWE assumption holds, we assume that for any QPT adversary $\mathcal{A}$*, it holds that*

$$|\Pr[\mathcal{A}(D_0) = 1] - \Pr[\mathcal{A}(D_1) = 1]| \leq \mathsf{negl}(\lambda).$$

Definition 7.12 (Puncturable PRF). *A puncturable PRF (PPRF) is a tuple of algorithms* $\mathsf{PPRF} = (\mathsf{PRF.Gen}, \mathsf{F}, \mathsf{Puncture})$ *where* $\{\mathsf{F}_K : \{0,1\}^{\ell_1} \to \{0,1\}^{\ell_2} \mid K \in \{0,1\}^{\lambda}\}$ *is a PRF family and satisfies the following two conditions. Note that* ℓ_1 *and* ℓ_2 *are polynomials of* λ.

Punctured correctness: *For any polynomial-size set* $S \subseteq \{0,1\}^{\ell_1}$ *and any* $x \in \{0,1\}^{\ell_1} \setminus S$*, it holds that*

$$\Pr\left[\mathsf{F}_K(x) = \mathsf{F}_{K_{\notin S}}(x) \mid K \leftarrow \mathsf{PRF.Gen}(1^{\lambda}), K_{\notin S} \leftarrow \mathsf{Puncture}(K, S)\right] = 1.$$

Pseudorandom at punctured point: *For any polynomial-size set* $S \subseteq \{0,1\}^{\ell_1}$ *and any QPT distinguisher* $\mathcal{A}$*, it holds that*

$$|\Pr[\mathcal{A}(\mathsf{F}_{K_{\notin S}}, \{\mathsf{F}_K(x_i)\}_{x_i \in S}) = 1] - \Pr\left[\mathcal{A}(\mathsf{F}_{K_{\notin S}}, (\mathcal{U}_{\ell_2})^{|S|}) = 1\right]| \leq \mathsf{negl}(\lambda),$$

where $K \leftarrow \mathsf{PRF.Gen}(1^{\lambda})$, $K_{\notin S} \leftarrow \mathsf{Puncture}(K, S)$ *and* $\mathcal{U}_{\ell_2}$ *denotes the uniform distribution over* $\{0,1\}^{\ell_2}$.

If $S = \{x^*\}$ *(i.e., puncturing a single point), we simply write* $\mathsf{F}_{\neq x^*}(\cdot)$ *instead of* $\mathsf{F}_{K_{\notin S}}(\cdot)$ *and consider* $\mathsf{F}_{\neq x^*}$ *as a keyed function.*

It is easy to see that the Goldwasser-Goldreich-Micali tree-based construction of PRFs (GGM PRF) [GGM86] from one-way function yield puncturable PRFs where the size of the punctured key grows polynomially with the size of the set S being punctured [BW13, BGI14, KPTZ13]. Thus, we have:

Theorem 7.13 ([GGM86, BW13, BGI14, KPTZ13]). *If OWFs exist, then for any polynomials $\ell_1(\lambda)$ and $\ell_2(\lambda)$, there exists a PPRF that maps ℓ_1-bits to ℓ_2-bits.*

Definition 7.14 (Public-Key Functional Encryption). *A PKFE scheme PKFE is a tuple of four PPT algorithms $(\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$. Below, let $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{F}$ be the plaintext, output, and function spaces of PKFE, respectively.*

$\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{msk})$**:** *The setup algorithm takes a security parameter 1^λ and outputs a public key pk and master secret key msk.*

$\mathsf{KG}(\mathsf{msk}, f) \rightarrow \mathsf{sk}_f$**:** *The key generation algorithm KG takes a master secret key msk and a function $f \in \mathcal{F}$, and outputs a functional decryption key sk_f.*

$\mathsf{Enc}(\mathsf{pk}, x) \rightarrow \mathsf{ct}$**:** *The encryption algorithm takes a public key pk and a message $x \in \mathcal{X}$, and outputs a ciphertext ct.*

$\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) \rightarrow y$**:** *The decryption algorithm takes a functional decryption key sk_f and a ciphertext ct, and outputs $y \in \{\bot\} \cup \mathcal{Y}$.*

Correctness: *We require we have that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}_f, \mathsf{ct}) = f(x) \;\middle|\; \begin{array}{l} (\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda), \\ \mathsf{sk}_f \leftarrow \mathsf{KG}(\mathsf{msk}, f), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x) \end{array}\right] = 1 - \mathsf{negl}(\lambda).$$

Note that Setup does not take collusion bound 1^q unlike SKFE in Definition 2.6 since we consider only single-key and collusion-resistant PKFE in this work.

Definition 7.15 (Adaptive Indistinguishability-Security for PKFE). *We say that* PKFE *is an* adaptively indistinguishability-secure *SDFE scheme for $\mathcal{X}, \mathcal{Y}$, and $\mathcal{F}$, if it satisfies the following requirement, formalized from the experiment $\mathsf{Exp}_{\mathcal{A}}^{\mathsf{ada\text{-}ind}}(1^\lambda, \mathsf{coin})$ between an adversary $\mathcal{A}$ and a challenger:*

1. *The challenger runs $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and sends pk to $\mathcal{A}$.*
2. *$\mathcal{A}$ sends arbitrary key queries. That is, $\mathcal{A}$ sends function $f_i \in \mathcal{F}$ to the challenger and the challenger responds with $\mathsf{sk}_{f_i} \leftarrow \mathsf{KG}(\mathsf{msk}, f_i)$ for the i-th query f_i.*
3. *At some point, $\mathcal{A}$ sends (x_0, x_1) to the challenger. If $f_i(x_0) = f_i(x_1)$ for all i, the challenger generates a ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}, x_{\mathsf{coin}})$. The challenger sends ct^* to $\mathcal{A}$.*
4. *Again, $\mathcal{A}$ can sends function queries f_i such that $f_i(x_0) = f_i(x_1)$.*
5. *$\mathcal{A}$ outputs a guess coin' for coin.*
6. *The experiment outputs coin'.*

We say that PKFE is adaptively indistinguishability-secure if, for any QPT $\mathcal{A}$, it holds that

$$\mathsf{Adv}_{\mathsf{PKFE},\mathcal{A}}^{\mathsf{ada\text{-}ind}}(\lambda) := \left|\Pr\left[\mathsf{Exp}_{\mathsf{PKFE},\mathcal{A}}^{\mathsf{ada\text{-}ind}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathsf{PKFE},\mathcal{A}}^{\mathsf{ada\text{-}ind}}(1^\lambda, 1) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

If $\mathcal{A}$ can send only one key query during the experiment, we say PKFE is adaptively single-key indistinguishability-secure.

Theorem 7.16 ([GVW12]). *If there exists PKE, there exists adaptively single-key indistinguishability-secure PKFE for $\mathsf{P/poly}$.*

Definition 7.17 (Indistinguishability Obfuscator [BGI+12]). *A PPT algorithm $i\mathcal{O}$ is a secure IO for a classical circuit class $\{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ if it satisfies the following two conditions.*

Functionality: *For any security parameter $\lambda \in \mathbb{N}$, circuit $C \in \mathcal{C}_\lambda$, and input x, we have that*

$$\Pr[C'(x) = C(x) \mid C' \leftarrow i\mathcal{O}(C)] = 1 \ .$$

Indistinguishability: *For any PPT* Samp *and QPT distinguisher* $\mathcal{D}$, *the following holds:*

If $\Pr[\forall x\ C_0(x) = C_1(x) \wedge |C_0| = |C_1| \mid (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Samp}(1^\lambda)] > 1 - \mathsf{negl}(\lambda)$, *then we have*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{io}}_{i\mathcal{O},\mathcal{D}}(\lambda) := \Big|&\Pr\left[\mathcal{D}(i\mathcal{O}(C_0), \mathsf{aux}) = 1 \mid (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Samp}(1^\lambda)\right] \\ &- \Pr\left[\mathcal{D}(i\mathcal{O}(C_1), \mathsf{aux}) = 1 \mid (C_0, C_1, \mathsf{aux}) \leftarrow \mathsf{Samp}(1^\lambda)\right]\Big| \le \mathsf{negl}(\lambda).\end{aligned}$$

There are a few candidates of secure IO for polynomial-size classical circuits against quantum adversaries [BGMZ18, CHVW19, AP20, DQV$^+$21].

7.2 Single-Decryptor Encryption

We review the notion of single-decryptor encryption (SDE) [GZ20, CLLZ21] since it is a crucial building block of our SDFE schemes. We also extend the existing definitions for SDE. Some definitions are taken from the paper by Coladangelo et al. [CLLZ21].

Definition 7.18 (Single-Decryptor Encryption [CLLZ21]). *A single-decryptor encryption scheme* SDE *is a tuple of four algorithms* $(\mathsf{Setup}, \mathcal{QKG}, \mathsf{Enc}, \mathcal{D}ec)$. *Below, let* $\mathcal{M}$ *be the message space of* SDE.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$**:** *The setup algorithm takes a security parameter* 1^λ, *and outputs a public key* pk *and a secret key* sk.

$\mathcal{QKG}(\mathsf{sk}) \to sk$**:** *The key generation algorithm* $\mathcal{QKG}$ *takes a secret key* sk, *and outputs a quantum decryption key* sk.

$\mathsf{Enc}(\mathsf{pk}, m) \to \mathsf{ct}$**:** *The encryption algorithm takes a public key* pk *and a message* $m \in \mathcal{M}$, *and outputs a ciphertext* ct.

$\mathcal{D}ec(sk, \mathsf{ct}) \to \tilde{m}$**:** *The decryption algorithm takes a quantum decryption key* sk *and a ciphertext* ct, *and outputs a message* $\tilde{m} \in \{\bot\} \cup \mathcal{M}$.

Correctness: *We require* $\mathcal{D}ec(sk, \mathsf{ct}) = m$ *for every* $m \in \mathcal{M}$, $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $sk \leftarrow \mathcal{QKG}(\mathsf{sk})$, *and* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$.

Note that Setup and Enc are classical algorithms. Although they could be quantum algorithms, we select the definition above since the construction by Coladangelo et al. [CLLZ21] satisfies it. This property is crucial in our schemes.

Definition 7.19 (CPA Security). *An SDE scheme satisfies CPA security if for any (stateful) QPT* $\mathcal{A}$, *it holds that*

$$2\left|\Pr\left[\mathcal{A}(\mathsf{ct}_b) = b \;\middle|\; \begin{array}{l}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda),\\ (m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk}), b \leftarrow \{0,1\},\\ \mathsf{ct}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)\end{array}\right] - \frac{1}{2}\right| \le \mathsf{negl}(\lambda).$$

Definition 7.20 (Quantum Decryptor [CLLZ21]). *Let* ρ *and* $\boldsymbol{U}$ *be a quantum state and a general quantum circuit acting on* $n + m$ *qubits, where* m *is the number of qubits of* ρ *and* n *is the length of ciphertexts. A quantum decryptor for ciphertexts is a pair* $(\rho, \boldsymbol{U})$.

When we say that we run the quantum decryptor $(\rho, \boldsymbol{U})$ *on ciphertext* ct, *we execute the circuit* $\boldsymbol{U}$ *on inputs* $|\mathsf{ct}\rangle$ *and* ρ.

Coladangelo et al. [CLLZ21] introduced a few security notions for SDE. We introduce the stronger security notion among them, γ-anti-piracy security.

Definition 7.21 (Testing a Quantum Decryptor [CLLZ21]). *Let $\gamma \in [0,1]$. Let pk and (m_0, m_1) be a public key and a pair of messages, respectively. A test for a γ-good quantum decryptor with respect to pk and (m_0, m_1) is the following procedure.*

- *The procedure takes as input a quantum decryptor $\mathcal{D} = (q, \boldsymbol{U})$.*
- *Let $\mathcal{P} = (\boldsymbol{P}, \boldsymbol{I} - \boldsymbol{P})$ be the following mixture of projective measurements acting on some quantum state q':*
  - *Sample a uniformly random $\mathsf{coin} \leftarrow \{0,1\}$ and compute $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, m_{\mathsf{coin}})$.*
  - *Run $m' \leftarrow \mathcal{D}(\mathsf{ct})$. If $m' = m_{\mathsf{coin}}$, output 1, otherwise output 0.*

  Let $\mathcal{TI}_{1/2+\gamma}(\mathcal{P})$ be the threshold implementation of $\mathcal{P}$ with threshold $\frac{1}{2} + \gamma$. Apply $\mathcal{TI}_{1/2+\gamma}(\mathcal{P})$ to q. If the outcome is 1, we say that the test passed, otherwise the test failed.

Definition 7.22 (Strong Anti-Piracy Security [CLLZ21]). *Let $\gamma \in [0,1]$. We consider the strong γ-anti-piracy game $\mathsf{Exp}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDE},\mathcal{A}}(\lambda, \gamma(\lambda))$ between the challenger and an adversary $\mathcal{A}$ below.*

1. *The challenger generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$.*
2. *The challenger generates $\mathpzc{sk} \leftarrow \mathcal{QKG}(\mathsf{sk})$ and sends $(\mathsf{pk}, \mathpzc{sk})$ to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs (m_0, m_1) and two (possibly entangled) quantum decryptors $\mathcal{D}_1 = (q[\mathsf{R}_1], \boldsymbol{U}_1)$ and $\mathcal{D}_2 = (q[\mathsf{R}_2], \boldsymbol{U}_2)$, where $m_0 \neq m_1$, $|m_0| = |m_1|$, q is a quantum state over registers R_1 and R_2, and $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are general quantum circuits.*
4. *The challenger runs the test for a γ-good decryptor with respect to (m_0, m_1) on $\mathcal{D}_1$ and $\mathcal{D}_2$. The challenger outputs 1 if both tests pass; otherwise outputs 0.*

We say that SDE is strong γ-anti-piracy secure if for any QPT adversary $\mathcal{A}$, it satisfies that

$$\Pr\left[\mathsf{Exp}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDE},\mathcal{A}}(\lambda, \gamma(\lambda)) = 1\right] \leq \mathsf{negl}(\lambda).$$

Theorem 7.23 ([CLLZ21, CV21]). *Assuming the existence of sub-exponentially secure IO for $\mathsf{P/poly}$ and OWFs, the hardness of QLWE, there exists an SDE scheme that satisfies strong γ-anti-piracy security for any inverse polynomial γ.*

Extension of SDE. We define a more liberal security notion for γ-anti-piracy security of SDE to use an SDE scheme as a building block of some cryptographic primitive.

We define a slightly stronger version of quantum decryptor than that in Definition 7.20.

Definition 7.24 (Testing a Quantum Distinguisher for Randomized Message). *Let $\gamma \in [0,1]$. Let pk and (m_0, m_1) be a public key and a pair of messages, respectively. Let $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda \in \mathbb{N}}$ be a function family. A test for a γ-good quantum distinguisher with respect to pk and $(m_0, m_1, g \in \mathcal{G}_\lambda)$ is the following procedure.*

- *The procedure takes as input a quantum decryptor $\mathcal{D} = (q, \boldsymbol{U})$.*
- *Let $\mathcal{P} = (\boldsymbol{P}, \boldsymbol{I} - \boldsymbol{P})$ be the following mixture of projective measurements acting on some quantum state q':*
  - *Sample a uniformly random $\mathsf{coin} \leftarrow \{0,1\}$ and $r \leftarrow \mathcal{R}$, and compute $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, g(m_{\mathsf{coin}}; r))$.*
  - *Run $\mathsf{coin}' \leftarrow \mathcal{D}(\mathsf{ct})$. If $\mathsf{coin}' = \mathsf{coin}$, output 1, otherwise output 0.*

Let $\mathcal{TI}_{1/2+\gamma}(\mathcal{P})$ be the threshold implementation of $\mathcal{P}$ with threshold $\frac{1}{2}+\gamma$. Apply $\mathcal{TI}_{1/2+\gamma}(\mathcal{P})$ to q. If the outcome is 1, we say that the test passed, otherwise the test failed.

The definition above is different from Definition 7.21. First, we apply a randomized function to m_{coin} and encrypt $g(m_{\mathsf{coin}};r)$. Second, $\mathcal{D}$ distinguish whether ct is a ciphertext of $g(m_0;r)$ or $g(m_1;r)$ rather than computing $g(m_{\mathsf{coin}};r)$ (or m_{coin}). These differences are crucial when using SDE as a building block of some cryptographic primitive.

We finished preparation for defining a new security notion for SDE. In the security game defined below, the adversary can send a randomized function applied to challenge messages by the challenger.

Definition 7.25 (Strong Anti-Piracy Security for Randomized Message). *Let $\gamma \in [0,1]$. We consider the strong γ-anti-piracy with randomized function family $\mathcal{G} = \{\mathcal{G}_\lambda\}_{\lambda\in\mathbb{N}}$ game $\mathsf{Exp}^{\mathsf{strong\text{-}anti\text{-}piracy\text{-}rand}}_{\mathsf{SDE},\mathcal{A},\mathcal{G}}(\lambda,\gamma(\lambda))$ between the challenger and an adversary $\mathcal{A}$ below.*

1. *The challenger generates $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$.*
2. *The challenger generates $\mathcal{sk} \leftarrow \mathcal{QKG}(\mathsf{sk})$ and sends $(\mathsf{pk},\mathcal{sk})$ to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs (m_0,m_1), $g \in \mathcal{G}_\lambda$, and two (possibly entangled) quantum decryptors $\mathcal{D}_1 = (q[\mathsf{R}_1],\boldsymbol{U}_1)$ and $\mathcal{D}_2 = (q[\mathsf{R}_2],\boldsymbol{U}_2)$, where $|m_0| = |m_1|$, q is a quantum state over registers R_1 and R_2, and $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are general quantum circuits.*
4. *The challenger runs the test for a γ-good distinguisher with respect to pk and (m_0,m_1,g) on $\mathcal{D}_1$ and $\mathcal{D}_2$. The challenger outputs 1 if both tests pass; otherwise outputs 0.*

We say that SDE is strong γ-anti-piracy secure with randomized function family $\mathcal{G}$ if for any QPT adversary $\mathcal{A}$, it satisfies that

$$\Pr\left[\mathsf{Exp}^{\mathsf{strong\text{-}anti\text{-}piracy\text{-}rand}}_{\mathsf{SDE},\mathcal{A},\mathcal{G}}(\lambda,\gamma(\lambda)) = 1\right] \le \mathsf{negl}(\lambda).$$

Note that we remove the restriction $m_0 \neq m_1$, which is in Definition 7.22, since the adversary has no advantage even if it sets $m_0 = m_1$.

We do not know how to prove that an SDE scheme that satisfies strong γ-anti-piracy secure also satisfies strong γ-anti-piracy secure with randomized function in a black-box way. This is because the adversary $\mathcal{A}$ does not receive a challenge ciphertext, but quantum decryptor $\mathcal{D}_1$ and $\mathcal{D}_2$ receives it. That is, a reduction that plays the role of $\mathcal{A}$ does not receive the challenge ciphertext. [14]

However, it is easy to see that the SDE scheme by Coladangelo et al. [CLLZ21] satisfies strong γ-anti-piracy secure with randomized function family $\mathcal{G}$. Intuitively, selecting g does not give more power to adversaries since they can select *any* (m_0,m_1) in the original strong anti-piracy security. We can easily obtain the following theorem.

Theorem 7.26 ([CLLZ21, CV21]). *Assuming the existence of sub-exponentially secure IO for $\mathsf{P/poly}$ and OWFs, the hardness of QLWE, and $\mathcal{G}$ is a randomized function family, there exists an SDE scheme that satisfies strong γ-anti-piracy security with randomize function family $\mathcal{G}$ for any inverse polynomial γ.*

The proof of Theorem 7.26 is almost the same as that of the strong anti-piracy security of the SDE by Coladangelo et al. [CLLZ21, Theorem 6.13 in the eprint ver.]. We explain only the differences to avoid replication.

[14]In the standard PKE setting, it is easy to see that the standard CPA security implies the liberal CPA security variant, where the adversary select not only (m_0,m_1) but also $g \in \mathcal{G}_\lambda$.

On randomized message. To obtain this theorem, we simply replace m_b with $g(m_b; r_b)$ in the proof by Coladangelo et al. [CLLZ21, Section 6.4 in the ePrint ver.], where r_b is randomness chosen by the challenger. The intuition is as follows. Even if the challenger applies the adversarially chosen randomized function g to m_0 or m_1, it does not give more power to the adversary for breaking strong γ-anti-piracy security. This is because the adversary can send *any* two messages (m_0, m_1) as a challenge message pair in the original strong γ-anti-piracy security game. The proof by Coladangelo et al. [CLLZ21] does not use any specific property of challenge messages. The one issue is whether $g(m_0; r_0) \neq g(m_1; r_1)$ holds for any $r_0, r_1 \in \mathcal{R}$ since the original strong γ-anti-piracy game requires that challenge messages must be different. However, this restriction is just a convention. If the adversary sets $m_0 = m_1$, it just loses how to distinguish ciphertexts. That is, even if $g(m_0; r_0) = g(m_1; r_1)$ happens, it does not give more power to the adversary, and is not a problem.

On distinguish-based definition. We require a pirate to be a distinguisher rather than a decryptor (computing entire m_{coin}). In general, security against distinguisher-based pirates is not implied by security against decryptor-based pirates since adversaries do not need to compute the entire m_{coin} in the distinguisher-based definition (the requirement on adversaries is milder).

In the original proof by Coladangelo et al. [CLLZ21], they show that if pirate decryptors compute m_{coin}, the reduction can distinguish whether a ciphertext is a real compute-and-compare obfuscation or a simulated compute-and-compare obfuscation.[15] This breaks the unpredictability of the supported distribution in compute-and-compare obfuscation. Thus, pirate decryptors cannot compute m_{coin}. Here, even if pirate decryptors only distinguish m_0 from m_1 (that is, the output of $\mathcal{D}$ is only one-bit information) instead of computing entire m_{coin}, the reduction can distinguish real or simulation for compute-and-compare obfuscation. This is because the security of compute-and-compare obfuscation is the decision-type. The reduction in the original proof use only information about coin and *does not use* m_{coin}. Thus, we can prove strong γ-anti-piracy security with any randomized function family $\mathcal{G}$ defined in Definition 7.25.

Other security notions for SDE. Coladangelo et al. introduced a few variants of anti-piracy security [CLLZ21]. One is the CPA-style anti-piracy security, and the other is the anti-piracy security with random challenge plaintexts. Both are weaker than the strong anti-piracy security in the SDE setting. See Definition A.1 for the CPA-style definition. We omit the anti-piracy security with random challenge plaintexts since we do not use it in this work.

7.3 Definitions for Single-Decryptor Functional Encryption

We define the notion of single-decryptor functional encryption (SDFE).

Definition 7.27 (Single-Decryptor Functional Encryption). *A single-decryptor functional encryption scheme* SDFE *is a tuple of five algorithms* $(\mathsf{Setup}, \mathsf{KG}, \mathcal{QKG}, \mathsf{Enc}, \mathcal{Dec})$. *Below, let* $\mathcal{X}$, $\mathcal{Y}$, *and* $\mathcal{F}$ *be the plaintext, output, and function spaces of* SDFE.

$\mathsf{Setup}(1^\lambda) \rightarrow (\mathsf{pk}, \mathsf{msk})$: *The setup algorithm takes a security parameter* 1^λ, *and outputs a public key* pk *and a master secret key* msk.

$\mathcal{QKG}(\mathsf{msk}, f) \rightarrow sk_f$: *The key generation algorithm takes a master secret key* msk *and a function* $f \in \mathcal{F}$, *and outputs a quantum functional decryption key* sk_f.

$\mathsf{Enc}(\mathsf{pk}, x) \rightarrow \mathsf{ct}$: *The encryption algorithm takes a public key* pk *and a plaintext* $x \in \mathcal{X}$, *and outputs a ciphertext* ct.

[15]The SDE scheme by Coladangelo et al. [CLLZ21] is constructed from IO and compute-and-compare obfuscation [WZ17, GKW17].

$\mathcal{Dec}(\mathcal{sk}_f, \mathsf{ct}) \to y$**:** *The decryption algorithm takes a quantum functional decryption key* $\mathcal{sk}_f$ *and a ciphertext* ct*, and outputs* $y \in \{\bot\} \cup \mathcal{Y}$*.*

Correctness: *For every* $\lambda \in \mathbb{N}$*,* $x \in \mathcal{X}$*,* $f \in \mathcal{F}$*,* $(\mathsf{pk}, \mathsf{msk})$ *in the support of* $\mathsf{Setup}(1^\lambda)$*, we require that*

$$\mathcal{Dec}(\mathcal{QKG}(\mathsf{msk}, f), \mathsf{Enc}(\mathsf{pk}, x)) = f(x).$$

Note that Setup and Enc are classical algorithms as Definition 7.18.

Definition 7.28 (Adaptive Security for SDFE). *We say that* SDFE *is an* adaptively secure *SDFE scheme for* $\mathcal{X}, \mathcal{Y}$*, and* $\mathcal{F}$*, if it satisfies the following requirement, formalized from the experiment* $\mathsf{Exp}^{\mathsf{ada\text{-}ind}}_{\mathsf{SDFE},\mathcal{A}}(1^\lambda, \mathsf{coin})$ *between an adversary* $\mathcal{A}$ *and a challenger:*

1. *The challenger runs* $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and sends* pk *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *sends arbitrary key queries. That is,* $\mathcal{A}$ *sends function* $f_i \in \mathcal{F}$ *to the challenger and the challenger responds with* $\mathcal{sk}_{f_i} \leftarrow \mathcal{QKG}(\mathsf{msk}, f_i)$ *for the i-th query* f_i*.*
3. *At some point,* $\mathcal{A}$ *sends* (x_0, x_1) *to the challenger. If* $f_i(x_0) = f_i(x_1)$ *for all i, the challenger generates a ciphertext* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}, x_{\mathsf{coin}})$*. The challenger sends* ct^* *to* $\mathcal{A}$*.*
4. *Again,* $\mathcal{A}$ *can sends function queries* f_i *such that* $f_i(x_0) = f_i(x_1)$*.*
5. $\mathcal{A}$ *outputs a guess* coin' *for* coin*.*
6. *The experiment outputs* coin'*.*

We say that SDFE *is adaptively secure if, for any QPT* $\mathcal{A}$*, it holds that*

$$\mathsf{Adv}^{\mathsf{ada\text{-}ind}}_{\mathsf{SDFE},\mathcal{A}}(\lambda) := \left| \Pr\left[\mathsf{Exp}^{\mathsf{ada\text{-}ind}}_{\mathsf{SDFE},\mathcal{A}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{ada\text{-}ind}}_{\mathsf{SDFE},\mathcal{A}}(1^\lambda, 1) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

If $\mathcal{A}$ *can send only one key query during the experiment, we say* SDFE *is adaptively single-key secure.*

Definition 7.29 (Testing a Quantum FE Distinguisher). *Let* $\gamma \in [0, 1]$*. Let* pk*,* (x_0, x_1)*, and* f^* *be a public key, a pair of plaintexts, and a function respectively such that* $f^*(x_0) \neq f^*(x_1)$*. A test for a* γ*-good quantum FE distinguisher with respect to* pk*,* (x_0, x_1)*, and* f^* *is the following procedure.*

- *The procedure takes as input a quantum FE decryptor* $\mathcal{D} = (q, \boldsymbol{U})$*.*
- *Let* $\mathcal{P} = (\boldsymbol{P}, \boldsymbol{I} - \boldsymbol{P})$ *be the following mixture of projective measurements acting on some quantum state* q'*:*
  - *Sample a uniformly random* $\mathsf{coin} \leftarrow \{0, 1\}$ *and compute* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, x_{\mathsf{coin}})$*.*
  - *Run* $\mathsf{coin}' \leftarrow \mathcal{D}(\mathsf{ct})$*. If* $\mathsf{coin}' = \mathsf{coin}$*, output* 1*; otherwise output* 0*.*

  Let $\mathcal{TI}_{1/2+\gamma}(\mathcal{P})$ *be the threshold implementation of* $\mathcal{P}$ *with threshold* $\frac{1}{2} + \gamma$*. Apply* $\mathcal{TI}_{1/2+\gamma}(\mathcal{P})$ *to* q*. If the outcome is* 1*, we say that the test passed, otherwise the test failed.*

We follow the spirit of Definition 7.24. That is, we consider a quantum distinguisher rather than a quantum decryptor that computes $f^*(x_{\mathsf{coin}})$.

Definition 7.30 (Strong Anti-Piracy Security for FE). *Let* $\gamma \in [0, 1]$*. We consider the strong* γ*-anti-piracy game* $\mathsf{Exp}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDFE},\mathcal{A}}(\lambda, \gamma(\lambda))$ *between the challenger and an adversary* $\mathcal{A}$ *below.*

1. *The challenger generates* $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and sends* pk *to* $\mathcal{A}$*.*
2. $\mathcal{A}$ *sends key queries* f_i *to the challenger and receives* $\mathcal{sk}_{f_i} \leftarrow \mathcal{QKG}(\mathsf{msk}, f_i)$*.*
3. *At some point* $\mathcal{A}$ *sends a challenge query* f^* *to the challenger and receives* $\mathcal{sk}_{f^*} \leftarrow \mathcal{QKG}(\mathsf{msk}, f^*)$*.*

4. *Again, $\mathcal{A}$ sends f_i to the challenger and receives $\mathcal{sk}_{f_i} \leftarrow \mathcal{QKG}(\mathsf{msk}, f_i)$.*

5. *$\mathcal{A}$ outputs (x_0, x_1) and two (possibly entangled) quantum decryptors $\mathcal{D}_1 = (q[\mathsf{R}_1], \boldsymbol{U}_1)$ and $\mathcal{D}_2 = (q[\mathsf{R}_2], \boldsymbol{U}_2)$, where $\forall i\ f_i(x_0) = f_i(x_1)$, $f^*(x_0) \neq f^*(x_1)$, q is a quantum state over registers R_1 and R_2, and $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are general quantum circuits.*

6. *The challenger runs the test for a γ-good FE distinguisher with respect to pk, (x_0, x_1), and f^* on $\mathcal{D}_1$ and $\mathcal{D}_2$. The challenger outputs 1 if both tests pass, otherwise outputs 0.*

We say that SDFE *is strong γ-anti-piracy secure if for any QPT adversary $\mathcal{A}$, it satisfies that*

$$\mathsf{Adv}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDFE},\mathcal{A}}(\lambda, \gamma(\lambda)) \coloneqq \Pr\left[\mathsf{Exp}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDFE},\mathcal{A}}(\lambda, \gamma(\lambda)) = 1\right] \leq \mathsf{negl}(\lambda).$$

If $\mathcal{A}$ can send only the challenge query f^* during the experiment, we say that SDFE *is challenge-only strong γ-anti-piracy secure.*

If a pirate has $\mathcal{sk}_{f^*}$, it can easily compute $f^*(x_{\mathsf{coin}})$ and is a good FE distinguisher since $f^*(x_0) \neq f^*(x_1)$. The definition says either $\mathcal{D}_1$ or $\mathcal{D}_2$ do not have the power of $\mathcal{sk}_{f^*}$.

Other security notions for SDFE. We can define CPA-style anti-piracy security for SDFE as SDE. We provide the CPA-style anti-piracy security for SDFE in Definition A.3. We can also define anti-piracy security with random challenge plaintexts for SDFE. However, if we allow adversaries to select a constant-valued function as f^*, the security is trivially broken. It might be plausible to define security with random challenge plaintexts for functions with high min-entropy. It limits supported classes of functions. However, we focus on FE for all circuits in this work. We omit the definition of anti-piracy with random challenge plaintexts.

Implication to FE with secure key leasing. Loosely speaking, SDFE implies FE with secure key leasing. However, there are subtle issues for this implication. Concretely, the implication holds if

- we allow the deletion certificate to be a quantum state, and
- restrict our attention to the setting where an adversary is given single decryption key that can be used to detect the challenge bit. We consider this setting for SDFE by default as we can see in Definition 7.30. However, we consider more powerful adversaries who can obtain multiple such decryption keys for secure key leasing as we can see in Definition 3.2. (Although we consider PKFE for SDFE and SKFE for secure key leasing in this work, we ignore the difference between public key and secret key for simplicity.)

Under these conditions, we can see the implication as follows. We require an adversary to send back the original quantum state encoding the decryption key as the deletion certificate. We can check the validity of this (quantum) certificate by estimating its success probability using the threshold implementation defined in Definition 7.7. Then, we see that an adversary who breaks the secure key leasing security of this construction clearly violates the single-decryptor security of the underlying scheme.

We can remove the second condition above if SDFE is also secure against adversaries who can obtain multiple decryption keys that can be used to detect the challenge bit. Such scheme is called collusion-resistant SDFE. Currently, we do not even have a construction of collusion-resistant SDE (that is, single decryptor PKE).

7.4 Single-Key Secure Single-Decryptor Functional Encryption

We use the following tools:

- A single-decryptor encryption $\mathsf{SDE} = (\mathsf{SDE.Setup}, \mathsf{SDE}.\mathcal{QKG}, \mathsf{SDE.Enc}, \mathsf{SDE}.\mathcal{Dec})$.

- An adaptively secure single-key PKFE scheme $\mathsf{PKFE} = (\mathsf{PKFE.Setup}, \mathsf{PKFE.KG}, \mathsf{PKFE.Enc}, \mathsf{PKFE.Dec})$.

The description of $\mathsf{1SDFE}$ is as follows.

$\mathsf{1SDFE.Setup}(1^\lambda)$:

- Generate $(\mathsf{fe.pk}, \mathsf{fe.msk}) \leftarrow \mathsf{PKFE.Setup}(1^\lambda)$.
- Generate $(\mathsf{sde.pk}, \mathsf{sde.dk}) \leftarrow \mathsf{SDE.Setup}(1^\lambda)$.
- Output $\mathsf{pk} := (\mathsf{fe.pk}, \mathsf{sde.pk})$ and $\mathsf{msk} := (\mathsf{fe.msk}, \mathsf{sde.dk})$.

$\mathsf{1SDFE}.\mathcal{QKG}(\mathsf{msk}, f)$:

- Parse $\mathsf{msk} = (\mathsf{fe.msk}, \mathsf{sde.dk})$.
- Generate $\mathsf{fe.sk}_f \leftarrow \mathsf{PKFE.KG}(\mathsf{msk}, f)$.
- Generate $\mathsf{sde}.\mathcal{dk} \leftarrow \mathsf{SDE}.\mathcal{QKG}(\mathsf{sde.dk})$.
- Output $\mathcal{sk}_f := (\mathsf{fe.sk}_f, \mathsf{sde}.\mathcal{dk})$.

$\mathsf{1SDFE.Enc}(\mathsf{pk}, x)$:

- Parse $\mathsf{pk} = (\mathsf{fe.pk}, \mathsf{sde.pk})$.
- Generate $\mathsf{fe.ct}_x \leftarrow \mathsf{PKFE.Enc}(\mathsf{fe.pk}, x)$.
- Generate $\mathsf{sde.ct} \leftarrow \mathsf{SDE.Enc}(\mathsf{sde.pk}, \mathsf{fe.ct}_x)$.
- Output $\mathsf{ct} := \mathsf{sde.ct}$.

$\mathsf{1SDFE}.\mathcal{Dec}(\mathcal{sk}_f, \mathsf{ct})$:

- Parse $\mathcal{sk}_f = (\mathsf{fe.sk}_f, \mathsf{sde}.\mathcal{dk})$ and $\mathsf{ct} = \mathsf{sde.ct}$.
- Compute $\mathsf{fe.ct}'_x \leftarrow \mathsf{SDE}.\mathcal{Dec}(\mathsf{sde}.\mathcal{dk}, \mathsf{sde.ct})$.
- Output $y \leftarrow \mathsf{PKFE.Dec}(\mathsf{fe.ct}'_x)$.

Theorem 7.31. *If* PKFE *is adaptively single-key indistinguishability-secure,* $\mathsf{1SDFE}$ *is adaptively single-key secure.*

Proof. Suppose that $\mathsf{Adv}^{\mathsf{ada\text{-}ind}}_{\mathsf{1SDFE},\mathcal{A}}(\lambda)$ is non-negligible for a contradiction. We construct a QPT algorithm $\mathcal{B}$ for the adaptive single-key security of PKFE by using the adversary $\mathcal{A}$ of the adaptive single-key security game of $\mathsf{1SDFE}$ as follows.

1. $\mathcal{B}$ receives $\mathsf{fe.pk}$ from its challenger, generates $(\mathsf{sde.pk}, \mathsf{sde.dk}) \leftarrow \mathsf{SDE.Setup}(1^\lambda)$, and sends $\mathsf{pk} := (\mathsf{fe.pk}, \mathsf{sde.pk})$ to $\mathcal{A}$.
2. When $\mathcal{A}$ sends a key query f, $\mathcal{B}$ sends it to its challenger, receives $\mathsf{fe.sk}_f \leftarrow \mathsf{PKFE.KG}(\mathsf{fe.msk}, f)$, computes $\mathsf{sde}.\mathcal{dk} \leftarrow \mathsf{SDE}.\mathcal{QKG}(\mathsf{sde.dk})$, and passes $\mathcal{sk}_f := (\mathsf{fe.sk}_f, \mathsf{sde}.\mathcal{dk})$ to $\mathcal{A}$.
3. When $\mathcal{A}$ sends a pair (x_0, x_1), $\mathcal{B}$ passes (x_0, x_1) to its challenger and receives $\mathsf{fe.ct}^* \leftarrow \mathsf{PKFE.Enc}(\mathsf{fe.pk}, x_{\mathsf{coin}})$. $\mathcal{B}$ generates $\mathsf{sde.ct} \leftarrow \mathsf{SDE.Enc}(\mathsf{sde.pk}, \mathsf{fe.ct}^*)$ and passes $\mathsf{sde.ct}$ to $\mathcal{A}$.
4. If $f(x_0) \neq f(x_1)$, $\mathcal{B}$ aborts. Otherwise, go to the next step.
5. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

It is easy to see that $\mathcal{B}$ perfectly simulates $\mathsf{Exp}^{\mathsf{ada\text{-}ind}}_{\mathsf{1SDFE},\mathcal{A}}(\lambda)$ since $\mathcal{A}$ must send (f, x_0, x_1) such that $f(x_0) = f(x_1)$ due to the condition of $\mathsf{Exp}^{\mathsf{ada\text{-}ind}}_{\mathsf{1SDFE},\mathcal{A}}(\lambda)$. Thus, $\mathcal{B}$ breaks the adaptive single-key security of PKFE by using the distinguishing power of $\mathcal{A}$. This is a contradiction and we finished the proof. ■

Theorem 7.32. *If* SDE *is strong* $\gamma/4$*-anti-piracy secure for randomized function family,* $\mathsf{1SDFE}$ *is challenge-only strong* γ*-anti-piracy secure.*

Proof. We recall the original game.

Hyb_0**:** This is the same as $\mathsf{Exp}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathcal{A},\mathsf{1SDFE}}(\lambda,\gamma(\lambda))$. The detailed description is as follows.

1. Compute $(\mathsf{fe.pk},\mathsf{fe.msk}) \leftarrow \mathsf{PKFE.Setup}(1^\lambda)$ and $(\mathsf{sde.pk},\mathsf{sde.dk}) \leftarrow \mathsf{SDE.Setup}(1^\lambda)$. Set $\mathsf{pk} := (\mathsf{fe.pk},\mathsf{sde.pk})$ and $\mathsf{msk} := (\mathsf{fe.msk},\mathsf{sde.dk})$.
2. Compute $\mathsf{fe.sk}_{f^*} \leftarrow \mathsf{PKFE.KG}(\mathsf{fe.msk},f^*)$ and $\mathsf{sde}.\mathcal{dk} \leftarrow \mathsf{SDE}.\mathcal{QKG}(\mathsf{sde.dk})$, and send $\mathcal{sk}_{f^*} := (\mathsf{fe.sk}_{f^*},\mathsf{sde}.\mathcal{dk})$ to $\mathcal{A}$.
3. Receive (x_0,x_1), $\mathcal{D}_1 = (q[\mathsf{R}_1],\boldsymbol{U}_1)$, and $\mathcal{D}_2 = (q[\mathsf{R}_2],\boldsymbol{U}_2)$, where $f^*(x_0) \neq f^*(x_1)$.
4. For $i \in \{1,2\}$, let $\mathcal{P}_{i,D}$ be the following mixture of projective measurements acting on some quantum state q':
   - Sample a uniform $\mathsf{coin} \leftarrow \{0,1\}$. Compute $\mathsf{fe.ct} \leftarrow \mathsf{PKFE.Enc}(\mathsf{fe.pk},x_{\mathsf{coin}})$ and $\mathsf{sde.ct} \leftarrow \mathsf{SDE.Enc}(\mathsf{sde.pk},\mathsf{fe.ct})$ and set $\mathsf{ct} := \mathsf{sde.ct}$.
   - Run the quantum decryptor $(q',\boldsymbol{U}_i)$ on input ct. If the outcome is coin, output 1. Otherwise, output 0.

   Let D be the distribution over pairs $(\mathsf{coin},\mathsf{ct})$ defined in the first item above, and let $\mathcal{E}_i = \{\boldsymbol{E}^i_{(\mathsf{coin},\mathsf{ct})}\}_{\mathsf{coin},\mathsf{ct}}$ be a collection of projective measurements where $\boldsymbol{E}^i_{(\mathsf{coin},\mathsf{ct})}$ is the projective measurement described in the second item above. $\mathcal{P}_{i,D}$ is the mixture of projective measurements associated to D and $\mathcal{E}_i$.
5. Run $\mathcal{TI}_{\frac{1}{2}+\gamma}(\mathcal{P}_{i,D})$ for $i \in \{1,2\}$ on quantum decryptor $\mathcal{D}_1 = (q[\mathsf{R}_1],\boldsymbol{U}_1)$ and $\mathcal{D}_2 = (q[\mathsf{R}_2],\boldsymbol{U}_2)$. Output 1 if both tests pass, otherwise output 0.

We prove the following.

Lemma 7.33. *If* SDE *is strong* γ*-anti-piracy secure with randomized function family* $\{\mathsf{PKFE.Enc}(\mathsf{fe.pk},\cdot;\cdot)\}_\lambda$*, it holds that* $\mathsf{Adv}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDFE},\mathcal{A}}(\lambda,\gamma(\lambda)) \leq \mathsf{negl}(\lambda)$.

Proof of Lemma 7.33. Suppose that $\mathsf{Adv}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDFE},\mathcal{A}}(\lambda,\gamma(\lambda))$ is non-negligible for a contradiction. We construct a QPT algorithm $\mathcal{B}$ for the strong γ-anti-piracy game with randomized function $\{\mathsf{PKFE.Enc}(\mathsf{fe.pk},\cdot;\cdot)\}_\lambda$ of SDE by using the adversary $\mathcal{A}$ of the strong γ-anti-piracy game of SDFE as follows.

1. $\mathcal{B}$ receives $\mathsf{sde.pk}$ and $\mathsf{sde}.\mathcal{dk}$ from its challenger.
2. $\mathcal{B}$ generates $(\mathsf{fe.pk},\mathsf{fe.msk}) \leftarrow \mathsf{PKFE.Setup}(1^\lambda)$, sets $\mathsf{pk} := (\mathsf{fe.pk},\mathsf{sde.pk})$, and sends it to $\mathcal{A}$.
3. When $\mathcal{A}$ sends the challenge query f^*, $\mathcal{B}$ generates $\mathsf{fe.sk}_{f^*} \leftarrow \mathsf{PKFE.KG}(\mathsf{fe.msk},f^*)$, sets $\mathcal{sk}_{f^*} := (\mathsf{fe.sk}_{f^*},\mathsf{sde}.\mathcal{dk})$, and sends $\mathcal{sk}_{f^*}$ to $\mathcal{A}$.
4. At some point, $\mathcal{B}$ receives (x_0,x_1) and two (possibly entangled) quantum decryptors $\mathcal{D}_1 = (q[\mathsf{R}_1],\boldsymbol{U}_1)$ and $\mathcal{D}_2 = (q[\mathsf{R}_2],\boldsymbol{U}_2)$, where $f^*(x_0) \neq f^*(x_1)$, from $\mathcal{A}$.
5. $\mathcal{B}$ sets quantum decryptors $\mathcal{D}^*_1$ and $\mathcal{D}^*_2$ as follows. $\mathcal{B}$ sets $\mathcal{D}^*_1 := (q[\mathsf{R}_1],\boldsymbol{U}_1)$ and $\mathcal{D}^*_2 := (q[\mathsf{R}_2],\boldsymbol{U}_2)$. $\mathcal{B}$ sets a randomized function $g(\cdot;\cdot) := \mathsf{PKFE.Enc}(\mathsf{fe.pk},\cdot;\cdot)$ and sends $(x_0,x_1,g))$ and $\mathcal{D}^*_1 = (q[\mathsf{R}_1],\boldsymbol{U}_1)$ and $\mathcal{D}^*_2 = (q[\mathsf{R}_2],\boldsymbol{U}_2)$ to its challenger.

Recall that, for testing quantum decryptors $\mathcal{D}_1$ and $\mathcal{D}_2$ in $\mathsf{Adv}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDFE},\mathcal{A}}(\lambda,\gamma(\lambda))$, we use the following: For $i \in \{1,2\}$, let $\mathcal{P}_{i,D}$ be the following mixture of projective measurements acting on some quantum state q':

- Sample a uniform coin $\leftarrow \{0,1\}$. Compute $\mathsf{fe.ct} \leftarrow \mathsf{PKFE.Enc}(\mathsf{fe.pk}, x_{\mathsf{coin}})$ and $\mathsf{sde.ct} \leftarrow \mathsf{SDE.Enc}(\mathsf{sde.pk}, \mathsf{fe.ct})$ and set $\mathsf{ct} := \mathsf{sde.ct}$.
- Run the quantum decryptor $(q', \boldsymbol{U}_i)$ on input ct. If the outcome is coin, output 1. Otherwise, output 0.

The challenger of SDE runs the test by using the following. Let $\mathcal{P}_{i,D'}$ be the following mixture of projective measurements acting on some quantum state q_{SDE}:

- Sample a uniformly random coin $\leftarrow \{0,1\}$ and $r \leftarrow \mathcal{R}$, and compute $m'_{\mathsf{coin}} := \mathsf{PKFE.Enc}(\mathsf{fe.pk}, x_{\mathsf{coin}}; r)$. Note that $\mathcal{B}$ sets $g(\cdot,\cdot) := \mathsf{PKFE.Enc}(\mathsf{fe.pk}, \cdot; \cdot)$. Then, compute $\mathsf{ct} \leftarrow \mathsf{SDE.Enc}(\mathsf{sde.pk}, m'_{\mathsf{coin}})$.
- Run $\mathsf{coin}' \leftarrow \mathcal{D}(\mathsf{ct})$. If $\mathsf{coin}' = \mathsf{coin}$, output 1, otherwise output 0.

The distribution is the same as D, which $\mathcal{B}$ simulates.

We assumed that $\mathsf{Adv}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDFE},\mathcal{A}}(\lambda, \gamma(\lambda))$ is non-negligible at the beginning of this proof. That is, applying $\mathcal{TI}_{\frac{1}{2}+\gamma}(\mathcal{P}_{i,D})$ on $q[\mathsf{R}_i]$ results in two outcomes 1 with non-negligible probability:

$$\mathrm{Tr}\left[\mathcal{TI}_{\frac{1}{2}+\gamma}(\mathcal{P}_{1,D}) \otimes \mathcal{TI}_{\frac{1}{2}+\gamma}(\mathcal{P}_{2,D}) q\right] > \mathsf{negl}(\lambda),$$

where $\mu > \mathsf{negl}(\lambda)$ means μ is non-negligible. This means that $q[\mathsf{R}_i]$ is a γ-good distinguisher with respect to ciphertexts generated according to D. It is easy to see that $(\mathcal{D}_1^*, \mathcal{D}_2^*)$ is a γ-good distinguisher for randomized function $\mathsf{PKFE.Enc}(\mathsf{fe.pk}, \cdot; \cdot)$ since D' and D are the same distribution. Thus, if $\mathcal{A}$ outputs γ-good distinguisher, $\mathcal{B}$ can also outputs γ-good distinguisher for randomized function $\mathsf{PKFE.Enc}(\mathsf{fe.pk}, \cdot; \cdot)$. This completes the proof of Lemma 7.33. ■

By Lemma 7.33, we complete the proof of Theorem 7.32. ■

We obtain the following corollary from Theorems 7.26, 7.31 and 7.32.

Corollary 7.34. *Assuming the existence of sub-exponentially secure IO for* $\mathsf{P/poly}$ *and OWFs, and the hardness of QLWE, there exists an SDFE scheme for* $\mathsf{P/poly}$ *that satisfies adaptive single-key security and challenge-only strong γ-anti-piracy security for any inverse polynomial γ.*

Remark 7.35 (On approximation version of threshold implementation). We do not need to use approximate version of $\mathcal{TI}$ that runs in polynomial time (denoted by $\mathcal{ATI}$ by previous works [ALL+21, CLLZ21]) because we do not extract information from pirate decryptors $\mathcal{D}_1$ and $\mathcal{D}_2$. We use the decision bits output by them for breaking the security of cryptographic primitives in security reductions. This is the same for the construction in Section 7.5.

7.5 Collusion-Resistant Single-Decryptor Functional Encryption

Construction 7.36. We use the following tools:

- A single-key SDFE $\mathsf{1SDFE} = (\mathsf{1SDFE.Setup}, \mathsf{1SDFE.KG}, \mathsf{1SDFE}.\mathcal{QKG}, \mathsf{1SDFE.Enc}, \mathsf{1SDFE}.\mathcal{D}ec)$.
- An IO $i\mathcal{O}$.
- A puncturable PRF $\mathsf{PPRF} = (\mathsf{PRF.Gen}, \mathsf{F}, \mathsf{Puncture})$, where $\mathsf{F} : \{0,1\}^{\lambda} \times \{0,1\}^{\ell} \to \mathcal{R}_{\mathsf{Setup}}$ and $\mathcal{R}_{\mathsf{Setup}}$ is the randomness space of 1SDFE.Setup.
- A puncturable PRF $\mathsf{PPRF}' = (\mathsf{PRF.Gen}', \mathsf{F}', \mathsf{Puncture}')$, where $\mathsf{F}' : \{0,1\}^{\lambda} \times \{0,1\}^{\ell} \to \mathcal{R}_{\mathsf{Enc}}$ and $\mathcal{R}_{\mathsf{Enc}}$ is the randomness space of 1SDFE.Enc.
- A (standard) PRF $\mathsf{F}^{(1)} : \{0,1\}^{\lambda} \times \mathcal{C}_{\mathsf{1SDFE}} \to \mathcal{R}_{\mathsf{PRF.Gen}'}$, $\mathcal{C}_{\mathsf{1SDFE}}$ is the ciphertext space of 1SDFE and $\mathcal{R}_{\mathsf{PRF.Gen}'}$ is the randomness space of PRF.Gen′.

Setup Circuit $\mathsf{S}_{\mathsf{1fe}}[\mathsf{K}](\tau)$

Hardwired: puncturable PRF key K.
Input: tag $\tau \in \{0,1\}^{\ell}$.
Padding: circuit is padded to size $\mathsf{pad}_{\mathsf{S}} := \mathsf{pad}_{\mathsf{S}}(\lambda, n, s, \ell)$, which is determined in analysis.

1. Compute $r_\tau \leftarrow \mathsf{F}_{\mathsf{K}}(\tau)$.
2. Compute $(\mathsf{pk}_\tau, \mathsf{msk}_\tau) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; r_\tau)$ and output pk_τ.

Figure 2: Description of $\mathsf{S}_{\mathsf{1fe}}[\mathsf{K}]$.

Encryption Circuit $\mathsf{E}_{\mathsf{1fe}}[\widehat{\mathsf{pk}}, \mathsf{K}', x](\tau)$

Hardwired: circuit $\widehat{\mathsf{pk}}$, puncturable PRF key K', and message x.
Input: tag $\tau \in \{0,1\}^{\ell}$.
Padding: circuit is padded to size $\mathsf{pad}_{\mathsf{E}} := \mathsf{pad}_{\mathsf{E}}(\lambda, n, s, \ell)$, which is determined in analysis.

1. Evaluate the circuit $\widehat{\mathsf{pk}}$ on input τ, that is $\mathsf{pk}_\tau \leftarrow \widehat{\mathsf{pk}}(\tau)$.
2. Compute $r'_\tau \leftarrow \mathsf{F}'_{\mathsf{K}'}(\tau)$ and output $\mathsf{ct}_\tau \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_\tau, x; r'_\tau)$.

Figure 3: Description of $\mathsf{E}_{\mathsf{1fe}}[\widehat{\mathsf{pk}}, \mathsf{K}', x]$.

- A (standard) PRF $\mathsf{F}^{(2)} : \{0,1\}^\lambda \times \mathcal{C}_{\mathsf{1SDFE}} \rightarrow \mathcal{R}_{i\mathcal{O}}$, $\mathcal{C}_{\mathsf{1SDFE}}$ is the ciphertext space of 1SDFE and $\mathcal{R}_{i\mathcal{O}}$ is the randomness space of $i\mathcal{O}$.

Note that we use $\mathsf{F}^{(1)}$ and $\mathsf{F}^{(2)}$ only in security proofs.

$\mathsf{SDFE.Setup}(1^\lambda)$:

- Generate $\mathsf{K} \leftarrow \mathsf{PRF.Gen}(1^\lambda)$ and $\mathsf{S}_{\mathsf{1fe}}[\mathsf{K}]$ defined in Figure 2.
- Return $(\widehat{\mathsf{pk}}, \widehat{\mathsf{msk}}) := (i\mathcal{O}_1(\mathsf{S}_{\mathsf{1fe}}), \mathsf{K})$.

$\mathsf{SDFE}.\mathcal{QKG}(\widehat{\mathsf{msk}}, f)$:

- Parse $\mathsf{K} = \widehat{\mathsf{msk}}$ and choose $\tau \leftarrow \{0,1\}^\ell$.
- Compute $r_\tau \leftarrow \mathsf{F}_{\mathsf{K}}(\tau)$ and $(\mathsf{msk}_\tau, \mathsf{pk}_\tau) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; r_\tau)$.
- Generate $\mathcal{sk}_{f,\tau} \leftarrow \mathsf{1SDFE}.\mathcal{QKG}(\mathsf{msk}_\tau, f)$.
- Output $\widehat{\mathcal{sk}}_{f,\tau} := (\tau, \mathcal{sk}_{f,\tau})$.

$\mathsf{SDFE.Enc}(\widehat{\mathsf{pk}}, x)$:

- Generate $\mathsf{K}' \leftarrow \mathsf{PRF.Gen}'(1^\lambda)$ and $\mathsf{E}_{\mathsf{1fe}}[\widehat{\mathsf{pk}}, \mathsf{K}', x]$ defined in Figure 3.
- Return $\widehat{\mathsf{ct}} \leftarrow i\mathcal{O}_2(\mathsf{E}_{\mathsf{1fe}}[\widehat{\mathsf{pk}}, \mathsf{K}', x])$.

$\mathsf{SDFE}.\mathcal{Dec}(\widehat{\mathcal{sk}}_f, \widehat{\mathsf{ct}})$:

- Parse $(\tau, \mathcal{sk}_{f,\tau}) = \widehat{\mathcal{sk}}_f$.
- Evaluate the circuit $\widehat{\mathsf{ct}}$ on input τ, that is $\mathsf{ct}_\tau \leftarrow \widehat{\mathsf{ct}}(\tau)$.
- Return $y \leftarrow \mathsf{1SDFE}.\mathcal{Dec}(\mathcal{sk}_{f,\tau}, \mathsf{ct}_\tau)$.

Theorem 7.37. *If* $\mathsf{1SDFE}$ *is sub-exponentially adaptively single-key secure,* $i\mathcal{O}$ *is a sub-exponentially secure IO for* P/poly*, and* PPRF *is a sub-exponentially secure PPRF,* SDFE *is adaptively secure.*

Proof. We define a sequence of hybrid games. For the hybrid games, we present some definitions. Let q be the total number of key queries by $\mathcal{A}$. Note that q could be any polynomial. When we start the adaptive security game, the adversary $\mathcal{A}$ is fixed, and q is also fixed. We choose tags $\tau_1, \ldots, \tau_q \leftarrow \{0,1\}^\ell$ for q key queries at the beginning of the game. We can interpret ℓ bit strings as integers and assume that there is no i, j such that $i \neq j$ and $\tau_i = \tau_j$ without loss of generality.

$\mathsf{Hyb}_0(\mathsf{coin})$**:** This is the original adaptive security game.

1. The challenger generates $(\widehat{\mathsf{pk}}, \widehat{\mathsf{msk}}) = (i\mathcal{O}_1(\mathsf{S}_{\mathsf{1fe}}), \mathsf{K}) \leftarrow \mathsf{SDFE.Setup}(1^\lambda)$ and sends $\widehat{\mathsf{pk}}$ to $\mathcal{A}$.
2. $\mathcal{A}$ sends key queries f_k to the challenger and the challenger generates $r_{\tau_k} \leftarrow \mathsf{F}_\mathsf{K}(\tau_k)$ and $(\mathsf{pk}_{\tau_k}, \mathsf{msk}_{\tau_k}) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; r_{\tau_k})$, and returns $\widehat{\mathcal{sk}}_{f_k} = (\tau_k, \mathcal{sk}_{f_k,\tau_k})$ where $\mathcal{sk}_{f_k,\tau_k} \leftarrow \mathsf{1SDFE}.\mathcal{QKG}(\mathsf{msk}_{\tau_k}, f_k)$.
3. At some point $\mathcal{A}$ sends a pair (x_0, x_1) to the challenger. If $f_i(x_0) = f_i(x_1)$, the challenger generates $\mathsf{K}' \leftarrow \mathsf{PRF.Gen}'(1^\lambda)$ and $\mathsf{E}_{\mathsf{1fe}}[\widehat{\mathsf{pk}}, \mathsf{K}', x_{\mathsf{coin}}]$ defined in Figure 3 and sends $\widehat{\mathsf{ct}} \leftarrow i\mathcal{O}_2(\mathsf{E}_{\mathsf{1fe}}[\widehat{\mathsf{pk}}, \mathsf{K}', x_{\mathsf{coin}}])$ to $\mathcal{A}$.
4. Again, $\mathcal{A}$ sends f_k to the challenger and the challenger returns $\widehat{\mathcal{sk}}_{f_k} \leftarrow \mathsf{SDFE}.\mathcal{QKG}(\widehat{\mathsf{msk}}, f_k)$ if $f_k(x_0) = f_k(x_1)$.
5. When $\mathcal{A}$ outputs coin', the game outputs coin'.

Let $i \in [2^\ell]$. In the following hybrid games, we gradually change $\mathsf{S}_{\mathsf{1fe}}$ and $\mathsf{E}_{\mathsf{1fe}}$ for the i-th tag τ'_i, which is the ℓ-bit string representation of $i \in [2^\ell]$. So, it could happen that $\tau'_i = \tau_j$ for some $i \in [2^\ell]$ and $j \in [q]$. Let $\mathsf{Adv}_0(\mathsf{coin})$ and Adv^i_x be the advantage of $\mathcal{A}$ in $\mathsf{Hyb}_0(\mathsf{coin})$ and Hyb^i_x, respectively.

Hyb^i_1**:** We generate $\widehat{\mathsf{pk}}$ as obfuscated $\mathsf{S}^*_{\mathsf{1fe}}$ described in Figure 4. In this hybrid game, we set $r_i \leftarrow \mathsf{F}_\mathsf{K}(\tau'_i)$, $\mathsf{F}_{\neq\tau'_i} \leftarrow \mathsf{Puncture}(\mathsf{K}, \tau'_i)$, $(\mathsf{pk}_{\tau'_i}, \mathsf{msk}_{\tau'_i}) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; r_i)$, and $\widehat{\mathsf{pk}} \leftarrow i\mathcal{O}_1(\mathsf{S}^*_{\mathsf{1fe}}[\tau'_i, \mathsf{F}_{\neq\tau'_i}, \mathsf{pk}_{\tau'_i}])$.

When $i = 1$, the behavior of $\mathsf{S}^*_{\mathsf{1fe}}$ is the same as that of $\mathsf{S}_{\mathsf{1fe}}$ since the hard-wired $\mathsf{pk}_{\tau'_1}$ in $\mathsf{S}^*_{\mathsf{1fe}}$ is the same as the output of $\mathsf{S}_{\mathsf{1fe}}$ on the input τ'_1. Their size is also the same since we pad circuit $\mathsf{S}_{\mathsf{1fe}}$ to have the same size as $\mathsf{S}^*_{\mathsf{1fe}}$. Then, we can use the indistinguishability of $i\mathcal{O}_1$ and it holds that $\left|\mathsf{Adv}_0(0) - \mathsf{Adv}^1_1\right| \leq \mathsf{negl}(\lambda)$.

Hyb^i_2**:** The challenge ciphertext is generated by obfuscating $\mathsf{E}^*_{\mathsf{1fe}}$ described in Figure 5. In this hybrid game, we set $r'_i \leftarrow \mathsf{F}'_{\mathsf{K}'}(\tau'_i)$, $\mathsf{F}'_{\neq\tau'_i} \leftarrow \mathsf{Puncture}(\mathsf{K}', \tau'_i)$, $\mathsf{ct}_i \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau'_i}, x_0; r'_i)$, $\mathsf{pk}_{\tau'_i} \leftarrow \widehat{\mathsf{pk}}(\tau'_i)$, and $\widehat{\mathsf{ct}} \leftarrow i\mathcal{O}_2(\mathsf{E}_{\mathsf{1fe}}[\tau', \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau'}, x_0, x_1, \mathsf{ct}_{\tau'}])$.

When $i = 1$, the behavior of $\mathsf{E}^*_{\mathsf{1fe}}$ is the same as that of $\mathsf{E}_{\mathsf{1fe}}$ since the hard-wired ct_1 in $\mathsf{E}^*_{\mathsf{1fe}}$ is the same as the output of $\mathsf{E}_{\mathsf{1fe}}$ on the input 1. Both circuits have the same size by padding pad_E.

In addition, for $i \geq 2$, the behavior of $\mathsf{E}^*_{\mathsf{1fe}}$ does not change between Hyb^i_1 and Hyb^i_2. Thus, it holds that $\left|\mathsf{Adv}^i_2 - \mathsf{Adv}^i_1\right| \leq \mathsf{negl}(\lambda)$ for every $i \in [2^\ell]$ due to the indistinguishability of $i\mathcal{O}_2$.

Hyb^i_3**:** We change $r_i = \mathsf{F}_\mathsf{K}(\tau'_i)$ and $r'_i = \mathsf{F}'_{\mathsf{K}'}(\tau'_i)$ into uniformly random r_i and r'_i. Due to the pseudorandomness at punctured points of puncturable PRF, it holds that $\left|\mathsf{Adv}^i_3 - \mathsf{Adv}^i_2\right| \leq \mathsf{negl}(\lambda)$ for every $i \in [2^\ell]$.

Setup Circuit $\mathsf{S}^*_{\mathsf{1fe}}[\tau', \mathsf{F}_{\neq\tau'}, \mathsf{pk}_{\tau'}](\tau)$

Hardwired: tag τ', puncturable PRF key $\mathsf{F}_{\neq\tau'}$, and 1SDFE public-key $\mathsf{pk}_{\tau'}$.

Input: tag $\tau \in \{0,1\}^\ell$.

Padding: circuit is padded to size $\mathsf{pad}_\mathsf{S} := \mathsf{pad}_\mathsf{S}(\lambda, n, s)$, which is determined in analysis.

1. If $\tau' = \tau$, output $\mathsf{pk}_{\tau'}$.
2. Else, compute $r \leftarrow \mathsf{F}_{\neq\tau'}(\tau)$.
3. Compute $(\mathsf{pk}_\tau, \mathsf{msk}_\tau) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; r)$ and output pk_τ.

Figure 4: Description of $\mathsf{S}^*_{\mathsf{1fe}}[\tau', \mathsf{F}_{\neq\tau'}, \mathsf{pk}_{\tau'}]$.

Encryption Circuit $\mathsf{E}^*_{\mathsf{1fe}}[\tau', \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau'}, x_0, x_1, \mathsf{ct}_{\tau'}](\tau)$

Hardwired: tag τ', public key $\widehat{\mathsf{pk}}$ (this is an obfuscated circuit), puncturable PRF key $\mathsf{F}'_{\neq\tau'}$, plaintexts x_0, x_1, and ciphertext $\mathsf{ct}_{\tau'}$.

Input: tag $\tau \in \{0,1\}^\ell$.

Padding: circuit is padded to size $\mathsf{pad}_\mathsf{E} := \mathsf{pad}_\mathsf{E}(\lambda, n, s)$, which is determined in analysis.

1. If $\tau' = \tau$, output $\mathsf{ct}_{\tau'}$.
2. Else, compute $r'_\tau \leftarrow \mathsf{F}'_{\neq\tau'}(\tau)$ and the circuit $\widehat{\mathsf{pk}}$ on input τ, that is, $\mathsf{pk}_\tau \leftarrow \widehat{\mathsf{pk}}(\tau)$,

   If $\tau > \tau'$: Output $\mathsf{ct}_\tau \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_\tau, x_0; r'_\tau)$.

   If $\tau < \tau'$: Output $\mathsf{ct}_\tau \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_\tau, x_1; r'_\tau)$.

Figure 5: Description of $\mathsf{E}^*_{\mathsf{1fe}}[\tau', \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau'}, x_0, x_1, \mathsf{ct}_{\tau'}]$.

Hyb^i_4**:** We change $\mathsf{ct}_{\tau'_i}$ from $\mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau'_i}, x_0)$ to $\mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau'_i}, x_1)$. In Hyb^i_3 and Hyb^i_4, we do not need randomness to generate $\mathsf{pk}_{\tau'_i}$ and $\mathsf{ct}_{\tau'_i}$. We just hardwire $\mathsf{pk}_{\tau'_i}$ and $\mathsf{ct}_{\tau'_i}$ into $\mathsf{S}^*_{\mathsf{1fe}}$ and $\mathsf{E}^*_{\mathsf{1fe}}$, respectively. Therefore, for every $i \in [2^\ell]$, $\left|\mathsf{Adv}^i_4 - \mathsf{Adv}^i_3\right| \leq \mathsf{negl}(\lambda)$ follows from the adaptive security of 1SDFE under the master public key $\mathsf{pk}_{\tau'_i}$.

Lemma 7.38. *It holds that* $\left|\mathsf{Adv}^i_4 - \mathsf{Adv}^i_3\right| \leq \mathsf{negl}(\lambda)$ *for all* $i \in [2^\ell]$ *if* 1SDFE *is adaptively single-key secure.*

Proof of Lemma 7.38. We construct an adversary $\mathcal{B}$ in the selective security game of 1SDFE as follows. To simulate the adaptive security game of SDFE, $\mathcal{B}$ runs $\mathcal{A}$ attacking 1SDFE. $\mathcal{A}$ adaptively sends key queries $f_1, \cdots, f_q$. $\mathcal{B}$ simulates the game of SDFE as follows.

Setup: $\mathcal{B}$ receives a public key $\mathsf{pk}_{\tau'_i}$. Then, $\mathcal{B}$ chooses $\tau_1, \ldots, \tau_q \in \{0,1\}^\ell$ and generates K and K' by using $\mathsf{PRF.Gen}(1^\lambda)$ and $\mathsf{PRF.Gen}'(1^\lambda)$, $\mathsf{F}_{\neq\tau'_i} \leftarrow \mathsf{Puncture}(\mathsf{K}, \tau'_i)$, $\mathsf{F}'_{\neq\tau'_i} \leftarrow \mathsf{Puncture}(\mathsf{K}', \tau'_i)$, and the public key $\widehat{\mathsf{pk}} := i\mathcal{O}_1(\mathsf{S}^*_{\mathsf{1fe}}[\tau'_i, \mathsf{F}_{\neq\tau'_i}, \mathsf{pk}_{\tau'_i}])$ for SDFE according to Figure 4 by using the given $\mathsf{pk}_{\tau'_i}$. $\mathcal{B}$ sends $\widehat{\mathsf{pk}}$ to $\mathcal{A}$.

Key Generation: When f_k (i.e., k-th query) is queried, $\mathcal{B}$ checks $f_k(x_0) = f_k(x_1)$ and outputs $\perp$ if it does not hold. Otherwise, $\mathcal{B}$ checks whether $\tau'_i = \tau_k$ (i.e., i-th tag in $[2^\ell]$ is the same as the tag for k-th key query). $\mathcal{B}$ passes f_k to its challenger, receives $\mathcal{sk}_{f_k}$, and sends $(\tau'_i, \mathcal{sk}_{f_k})$ to $\mathcal{A}$. If $\tau'_i \neq \tau_k$, $\mathcal{B}$ generates $\mathcal{sk}_{f_k} \leftarrow \mathsf{1SDFE}.\mathcal{QKG}(\mathsf{msk}_{\tau_k}, f_k)$ by using $(\mathsf{pk}_{\tau_k}, \mathsf{msk}_{\tau_k}) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; \mathsf{F}_{\neq\tau'_i}(\tau_k))$ and returns it. Note that we do not need $\mathsf{msk}_{\tau'_i}$ for this simulation since $\mathcal{B}$ receives $\mathcal{sk}_{f_k}$ from its challenger if $\tau'_i = \tau_k$.

Encryption: When $\mathcal{A}$ sends (x_0, x_1) to $\mathcal{B}$, $\mathcal{B}$ passes (x_0, x_1) to its challenger and receives $\mathsf{ct}^*_{\tau'_i}$. Then, $\mathcal{B}$ generates the challenge ciphertext $\widehat{\mathsf{ct}} \leftarrow i\mathcal{O}_2(\mathsf{E}^*[\tau'_i, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau'_i}, x_0, x_1, \mathsf{ct}^*_{\tau'_i}])$ (obfuscated $\mathsf{E}^*_{1\mathsf{fe}}$ Figure 5).

The simulation is completed. If $\mathcal{B}$ receives $\mathsf{ct}^*_{\tau'_i} = \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau'_i}, x_0)$, it perfectly simulates Hyb^i_3. If $\mathcal{B}$ receives $\mathsf{ct}^*_{\tau'_i} = \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau'_i}, x_1)$, it perfectly simulates Hyb^i_4. This completes the proof of the lemma. ■

Hyb^i_5: We change r_i and r'_i into $r_i = \mathsf{F}_{\mathsf{K}}(\tau'_i)$ and $r'_i = \mathsf{F}'_{\mathsf{K}'}(\tau'_i)$. We can show $\left|\mathsf{Adv}^i_5 - \mathsf{Adv}^i_4\right| \leq \mathsf{negl}(\lambda)$ for every $i \in [2^\ell]$ by using the pseudorandomness at punctured point of puncturable PRF.

From the definition of $\mathsf{S}^*_{1\mathsf{fe}}$, $\mathsf{E}^*_{1\mathsf{fe}}$, and Hyb^i_1, the behaviors of $\mathsf{S}^*_{1\mathsf{fe}}$ and $\mathsf{E}^*_{1\mathsf{fe}}$ in Hyb^i_5 and Hyb^{i+1}_1 are the same. Thus, $\left|\mathsf{Adv}^{i+1}_1 - \mathsf{Adv}^i_5\right| \leq \mathsf{negl}(\lambda)$ holds for every $i \in [2^\ell - 1]$ due to the indistinguishability of $i\mathcal{O}_1$ and $i\mathcal{O}_2$. It also holds that $\left|\mathsf{Adv}_0(1) - \mathsf{Adv}^{2^\ell}_5\right| \leq \mathsf{negl}(\lambda)$ based on the security guarantee of $i\mathcal{O}_1$ and $i\mathcal{O}_2$.

There are $O(2^\ell)$ hybrid games. However, if $\mathsf{negl}(\lambda)$ is sub-exponentially small for all transitions, $|\mathsf{Adv}_0(0) - \mathsf{Adv}_0(1)|$ is negligible. Note that it is sufficient to set $\ell = \text{polylog}(\lambda)$.

Padding Parameter. The proof of security relies on the indistinguishability of obfuscated $\mathsf{S}_{1\mathsf{fe}}$ and $\mathsf{S}^*_{1\mathsf{fe}}$ defined in Figures 2 and 4, and that of obfuscated $\mathsf{E}_{1\mathsf{fe}}$ and $\mathsf{E}^*_{1\mathsf{fe}}$ defined in Figures 3 and 5. Accordingly, we set $\mathsf{pad}_\mathsf{S} := \max(|\mathsf{S}_{1\mathsf{fe}}|, |\mathsf{S}^*_{1\mathsf{fe}}|)$ and $\mathsf{pad}_\mathsf{E} := \max(|\mathsf{E}_{1\mathsf{fe}}|, |\mathsf{E}^*_{1\mathsf{fe}}|)$.

The circuits $\mathsf{S}_{1\mathsf{fe}}$ and $\mathsf{S}^*_{1\mathsf{fe}}$ compute a puncturable PRF over domain $\{0,1\}^\ell$ and a key pair of 1FE, and may have punctured PRF keys and a public key hardwired. The circuits $\mathsf{E}_{1\mathsf{fe}}$ and $\mathsf{E}^*_{1\mathsf{fe}}$ run the circuit $\widehat{\mathsf{pk}}$ and compute a puncturable PRF over domain $\{0,1\}^\ell$ and a ciphertext of 1SDFE, and may have punctured PRF keys, tags, plaintexts, and a hard-wired ciphertext. Note that ℓ is a polynomial of λ. Thus, it holds that

$$\mathsf{pad}_\mathsf{S} \leq \text{poly}(\lambda, n, s),$$
$$\mathsf{pad}_\mathsf{E} \leq \text{poly}(\lambda, n, s, \left|\widehat{\mathsf{pk}}\right|).$$

Therefore, we complete the proof of Theorem 7.37. ■

Theorem 7.39. *If* 1SDFE *is challenge-only strong* $\gamma/2$*-anti-piracy secure and sub-exponentially adaptively single-key secure,* $i\mathcal{O}$ *is a sub-exponentially secure IO for* P/poly*, and* PPRF *and* PPRF′ *are sub-exponentially secure puncturable PRFs, then* SDFE *is strong* γ*-anti-piracy secure.*

Proof of Theorem 7.39. Let $\mathcal{A}$ be an adversary attacking the strong γ-anti-piracy of SDFE. We define a sequence of hybrid games. For the hybrid games, we define the following values. Let q be the total number of key queries by $\mathcal{A}$ except for the challenge query f^*. Note that q could be any polynomial. When we start the strong γ-anti-piracy security game, the adversary $\mathcal{A}$ is fixed, and q is also fixed. We choose tags $\tau_1, \ldots, \tau_q \leftarrow \{0,1\}^\ell$ for q key queries and τ^* for the challenge query at the beginning of the game. We can interpret ℓ bit strings as integers and assume that there is no i, j such that $i \neq j$ and $\tau_i = \tau_j$ without loss of generality.

Hyb_0: The first game is the original strong γ-anti-piracy experiment, that is, $\mathsf{Exp}^{\mathsf{strong\text{-}anti\text{-}piracy}}_{\mathsf{SDFE},\mathcal{A}}(1^\lambda, \gamma)$.

1. The challenger generates $(\widehat{\mathsf{pk}}, \widehat{\mathsf{msk}}) = (i\mathcal{O}_1(\mathsf{S}_{1\mathsf{fe}}), \mathsf{K}) \leftarrow \mathsf{SDFE.Setup}(1^\lambda)$ and sends $\widehat{\mathsf{pk}}$ to $\mathcal{A}$.

2. $\mathcal{A}$ sends key queries f_k to the challenger and the challenger returns $\widehat{sk}_{f_k} = (\tau_k, sk_{f_k,\tau_k}) \leftarrow \mathsf{SDFE}.\mathcal{QKG}(\widehat{\mathsf{msk}}, f_k)$ where $sk_{f_k,\tau_k} \leftarrow \mathsf{1SDFE}.\mathcal{QKG}(\mathsf{msk}_{\tau_k}, f_k)$.
3. At some point $\mathcal{A}$ sends a challenge query f^* to the challenger and the challenger returns $\widehat{sk} = (\tau^*, sk_{f^*}) \leftarrow \mathsf{SDFE}.\mathcal{QKG}(\widehat{\mathsf{msk}}, f^*)$ where $sk_{f^*} \leftarrow \mathsf{1SDFE}.\mathcal{QKG}(\mathsf{msk}_{\tau^*}, f^*)$.
4. Again, $\mathcal{A}$ sends f_k to the challenger and the challenger returns $\widehat{sk}_{f_k} \leftarrow \mathsf{SDFE}.\mathcal{QKG}(\widehat{\mathsf{msk}}, f_k)$.
5. $\mathcal{A}$ outputs (x_0, x_1) and two (possibly entangled) quantum decryptors $\mathcal{D}_1 = (q[\mathsf{R}_1], \boldsymbol{U}_1)$ and $\mathcal{D}_2 = (q[\mathsf{R}_2], \boldsymbol{U}_2)$, where $\forall i\ f_i(x_0) = f_i(x_1)$, $f^*(x_0) \neq f^*(x_1)$, q is a quantum state over registers R_1 and R_2, and $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are general quantum circuits.
6. For $i \in \{1,2\}$, let $\mathcal{P}_{i,D}$ be the following mixture of projective measurements acting on some quantum state q':

   - Sample a uniform $\mathsf{coin} \leftarrow \{0,1\}$. Compute $\widehat{\mathsf{ct}} \leftarrow \mathsf{SDFE.Enc}(\widehat{\mathsf{pk}}, x_{\mathsf{coin}})$ and set $\mathsf{ct} := \widehat{\mathsf{ct}}$.
   - Run the quantum decryptor $(q', \boldsymbol{U}_i)$ on input ct. If the outcome is coin, output 1. Otherwise, output 0.

   Let D be the distribution over pairs $(\mathsf{coin}, \mathsf{ct})$ defined in the first item above, and let $\mathcal{E}_i = \{\boldsymbol{E}^i_{(\mathsf{coin},\mathsf{ct})}\}_{\mathsf{coin},\mathsf{ct}}$ be a collection of projective measurements where $\boldsymbol{E}^i_{(\mathsf{coin},\mathsf{ct})}$ is the projective measurement described in the second item above. $\mathcal{P}_{i,D}$ is the mixture of projective measurements associated to D and $\mathcal{E}_i$.
7. The challenger runs the test for a γ-good FE decryptor with respect to pk, (x_0, x_1), and f^* on $\mathcal{D}_1$ and $\mathcal{D}_2$. The challenger outputs 1 if both tests pass, otherwise outputs 0.

In the following hybrid games, we gradually change $\mathsf{S}_{\mathsf{1fe}}$ and $\mathsf{E}_{\mathsf{1fe}}$ for the i-th tag τ'_i, which is the ℓ-bit string representation of $i \in [2^\ell]$, with the following exception. Let $i^* \in [2^\ell]$ is the integer representation of $\tau^* \in \{0,1\}^\ell$. Instead of going over indices $\{1, \ldots, i^*-1, i^*, i^*+1, \ldots, 2^\ell\}$, we go over indices $I_{i^*} := \{1, \ldots, i^*-1, i^*+1, \ldots, 2^\ell, i^*\}$. That is, we skip the tag τ_{i^*} and go to τ_{i^*} after we finish the 2^ℓ-th tag τ_{2^ℓ}. It could happen that $\tau'_i = \tau_j$ for some $i \in [2^\ell]$ and $j \in [q]$. Let Adv^i_x be the advantage of $\mathcal{A}$ in Hyb^i_x.

Hyb^i_1: This is defined for $i \in [I_{i^*}]$ and the same as Hyb^{i-1}_5 except that we generate $\widehat{\mathsf{pk}}$ as obfuscated $\mathsf{S}^*_{\mathsf{1fe}}$ described in Figure 6. Note that we define $\mathsf{Hyb}^0_5 = \mathsf{Hyb}_0$. In this hybrid game, we set $r_{\tau'_i} \leftarrow \mathsf{F}_\mathsf{K}(\tau'_i)$, $\mathsf{F}_{\neq\tau'_i} \leftarrow \mathsf{Puncture}(\mathsf{K}, \tau'_i)$, and $(\mathsf{pk}_{\tau'_i}, \mathsf{msk}_{\tau'_i}) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; r_{\tau'_i})$.

The behavior of $\mathsf{S}^*_{\mathsf{1fe}}$ is the same as that of $\mathsf{S}_{\mathsf{1fe}}$ since the hard-wired $\mathsf{pk}_{\tau'_1}$ in $\mathsf{S}^*_{\mathsf{1fe}}$ is the same as the output of $\mathsf{S}_{\mathsf{1fe}}$ on the input τ'_1. Their size is also the same since we pad circuit $\mathsf{S}_{\mathsf{1fe}}$ to have the same size as $\mathsf{S}^*_{\mathsf{1fe}}$. Then, we can use the indistinguishability guarantee of $i\mathcal{O}_1$ and it holds that $i\mathcal{O}_1(\mathsf{S}_{\mathsf{1fe}}[\mathsf{K}])$ is computationally indistinguishable from $i\mathcal{O}_1(\mathsf{S}^*_{\mathsf{1fe}}[\tau'_i, \mathsf{F}_{\neq\tau'_i}, \mathsf{pk}_{\tau'_i}])$. Let $D^{(1,i)}$ be the distribution over pairs $(\mathsf{coin}, \mathsf{ct})$ defined in Hyb_0 except that $i\mathcal{O}_1(\mathsf{S}^*_{\mathsf{1fe}}[\tau'_i, \mathsf{F}_{\neq\tau'_i}, \mathsf{pk}_{\tau'_i}])$ as $\widehat{\mathsf{pk}}$. Then, $D^{(1,1)}$ is computationally indistinguishable from D.

Hyb^i_2: This is defined for $i \in [I_{i^*}]$ and the same as Hyb^i_1 except that we change the distribution $D^{(1,i)}$ over pairs $(\mathsf{coin}, \mathsf{ct})$ into $D^{(2,i)}$ as follows.

- Sample a uniform $\mathsf{coin} \leftarrow \{0,1\}$.
- Compute $r'_{\tau'_i} \leftarrow \mathsf{F}'_{\mathsf{K}'}(\tau'_i)$, $\mathsf{F}'_{\neq\tau'_i} \leftarrow \mathsf{Puncture}'(\mathsf{K}', \tau'_i)$, $\mathsf{ct}_{\tau'_i} \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau'_i}, x_{\mathsf{coin}}; r'_{\tau'_i})$, and $\widehat{\mathsf{ct}} \leftarrow i\mathcal{O}_2(\mathsf{E}^*_{\mathsf{1fe}}[\tau'_i, \tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau'_i}, x_{\mathsf{coin}}, x_1, \mathsf{ct}_{\tau'_i}])$ where $\mathsf{E}^*_{\mathsf{1fe}}$ is described in Figure 7. Set $\mathsf{ct} := \widehat{\mathsf{ct}}$.

The behavior of $\mathsf{E}^*_{\mathsf{1fe}}$ is the same as that of $\mathsf{E}_{\mathsf{1fe}}$ since the hard-wired $\mathsf{ct}_{\tau'_i}$ in $\mathsf{E}^*_{\mathsf{1fe}}$ is the same as the output of $\mathsf{E}_{\mathsf{1fe}}$ on the input τ'_i. Moreover, both circuits have the same size by padding pad_E. Then, we

Setup Circuit $\mathsf{S}^*_{\mathsf{1fe}}[\tau', \mathsf{F}_{\neq\tau'}, \mathsf{pk}_{\tau'}](\tau)$

Hardwired: tag τ', puncturable PRF key $\mathsf{F}_{\neq\tau'}$, and 1SDFE public-key $\mathsf{pk}_{\tau'}$.

Input: tag $\tau \in \{0,1\}^\ell$.

Padding: circuit is padded to size $\mathsf{pad}_\mathsf{S} := \mathsf{pad}_\mathsf{S}(\lambda, n, s)$, which is determined in analysis.

```
1. If $\tau' = \tau$, output $\mathsf{pk}_{\tau'}$.
2. Else, compute $r \leftarrow \mathsf{F}_{\neq\tau'}(\tau)$.
3. Compute $(\mathsf{pk}_\tau, \mathsf{msk}_\tau) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; r_i)$ and output $\mathsf{pk}_\tau$.
```

Figure 6: Circuit $\mathsf{S}^*_{\mathsf{1fe}}[\tau', \mathsf{F}_{\neq\tau'}, \mathsf{pk}_{\tau'}]$.

Encryption Circuit $\mathsf{E}^*_{\mathsf{1fe}}[\tau', \tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau'}, x_{\mathsf{coin}}, x_1, \mathsf{ct}_{\tau'}](\tau)$

Hardwired: tags τ', τ^*, public key $\widehat{\mathsf{pk}}$ (this is an obfuscated circuit), puncturable PRF key $\mathsf{F}'_{\neq\tau'}$, plaintexts x_{coin}, x_1, and ciphertext $\mathsf{ct}_{\tau'}$.

Input: tag $\tau \in \{0,1\}^\ell$.

Padding: circuit is padded to size $\mathsf{pad}_\mathsf{E} := \mathsf{pad}_\mathsf{E}(\lambda, n, s)$, which is determined in analysis.

```
1. If $\tau' = \tau$, output $\mathsf{ct}_{\tau'}$.
2. Else, compute $r'_\tau \leftarrow \mathsf{F}'_{\neq\tau'_i}(\tau)$ and the circuit $\widehat{\mathsf{pk}}$ on input $\tau$, that is, $\mathsf{pk}_\tau \leftarrow \widehat{\mathsf{pk}}(\tau)$,
   If $\tau = \tau^*$: Output $\mathsf{ct}_{\tau^*} \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau^*}, x_{\mathsf{coin}}; r'_{\tau^*})$
   If $\tau > \tau'$: Output $\mathsf{ct}_\tau \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_\tau, x_{\mathsf{coin}}; r'_\tau)$.
   If $\tau < \tau'$: Output $\mathsf{ct}_\tau \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_\tau, x_1; r'_\tau)$.
```

Figure 7: Circuit $\mathsf{E}^*_{\mathsf{1fe}}[\tau', \tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau'}, x_{\mathsf{coin}}, x_1, \mathsf{ct}_{\tau'}]$.

can use the indistinguishability of $i\mathcal{O}_2$, and it holds that $D^{(2,i)}$ is computationally indistinguishable from $D^{(1,i)}$.

Hyb^i_3: This is defined for $i \in [I_{i^*}]$ and the same as Hyb^i_2 except that we use distribution $D^{(3,i)}$, which is the same as $D^{(2,i)}$ except that setup randomness $r_{\tau^*} \leftarrow \mathcal{R}_{\mathsf{Setup}}$ and encryption randomness $r'_{\tau^*} \leftarrow \mathcal{R}_{\mathsf{Enc}}$ are uniformly random. By the punctured pseudorandomness of F and F', $D^{(3,i)}$ is computationally indistinguishable from $D^{(2,i)}$.

Hyb^i_4: This is defined for $i \in [I_{i^*}] \setminus \{i^*\}$ and the same as Hyb^i_3 except that we use distribution $D^{(4,i)}$, which is the same as $D^{(3,i)}$ except that the ciphertext $\mathsf{ct}_{\tau'_i}$ is $\mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau'_i}, x_1)$ instead of $\mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau'_i}, x_{\mathsf{coin}})$. In Hyb^i_3 and Hyb^i_4, we do not need randomness to generate $\mathsf{pk}_{\tau'_i}$ and $\mathsf{ct}_{\tau'_i}$. We just hardwire $\mathsf{pk}_{\tau'_i}$ and $\mathsf{ct}_{\tau'_i}$ into $\mathsf{S}^*_{\mathsf{1fe}}$ and $\mathsf{E}^*_{\mathsf{1fe}}$, respectively. In addition, it holds that $f_k(x_0) = f_k(x_1)$ for all $k \in [q]$ by the requirement of strong γ-anti-piracy security. For every $i \in [I_{i^*}] \setminus \{i^*\}$, by the adaptive security of 1SDFE under the public key $\mathsf{pk}_{\tau'_i}$, $D^{(4,i)}$ is computationally indistinguishable from $D^{(3,i)}$. Note that we do not need to generate a functional decryption key for $i \in [I_{i^*}]$ such that $\tau'_i \neq \tau_k$. The detail of this indistinguishability is almost the same as Lemma 7.38, and we omit it.

Hyb^i_5: This is defined for $i \in [I_{i^*}] \setminus \{i^*\}$ and the same as Hyb^i_4 except that we use distribution $D^{(5,i)}$, which is the same as $D^{(4,i)}$ except that we use $r_i = \mathsf{F}_\mathsf{K}(\tau'_i)$ and $r'_i = \mathsf{F}'_{\mathsf{K}'}(\tau'_i)$. By the punctured pseudorandomness of F and F', $D^{(5,i)}$ is computationally indistinguishable from $D^{(4,i)}$.

We finish describing one cycle of our hybrid games. We move from Hyb^i_5 to Hyb^{i+1}_1. From the definition of $\mathsf{S}^*_{\mathsf{1fe}}$ and Hyb^i_1, the behaviors of $\mathsf{S}^*_{\mathsf{1fe}}$ in Hyb^i_5 and Hyb^{i+1}_1 are the same. It holds that

Encryption Circuit $\mathsf{E}^{**}_{\mathsf{1fe}}[\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{ct}_{\tau^*}](\tau)$

Hardwired: tag τ^*, public key $\widehat{\mathsf{pk}}$ (this is an obfuscated circuit), puncturable PRF key $\mathsf{F}'_{\neq\tau^*}$, plaintext x_1, and ciphertext ct_{τ^*}.

Input: tag $\tau \in \{0,1\}^{\ell}$.

Padding: circuit is padded to size $\mathsf{pad}_{\mathsf{E}} := \mathsf{pad}_{\mathsf{E}}(\lambda, n, s)$, which is determined in analysis.

1. If $\tau^* = \tau$, output ct_{τ^*}.
2. Else, compute $r'_\tau \leftarrow \mathsf{F}'_{\neq\tau^*}(\tau)$ and the circuit $\widehat{\mathsf{pk}}$ on input τ, that is, $\mathsf{pk}_\tau \leftarrow \widehat{\mathsf{pk}}(\tau)$,
3. Output $\mathsf{ct}_\tau \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_\tau, x_1; r'_\tau)$.

Figure 8: Circuit $\mathsf{E}^{**}_{\mathsf{1fe}}[\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{ct}_{\tau^*}]$.

$D^{(1,i+1)}$ is computationally indistinguishable from $D^{(5,i)}$. So, we can arrive at $\mathsf{Hyb}_3^{i^*}$ by iterating the transitions so far.

$\mathsf{Hyb}_4^{i^*}$**:** This is the same as $\mathsf{Hyb}_3^{i^*}$ except that we change the distribution $D^{(3,i^*)}$ over pairs $(\mathsf{coin}, \mathsf{ct})$ into $D^{(4,i^*)}$ as follows.

- Sample a uniform $\mathsf{coin} \leftarrow \{0,1\}$.
- Sample $r'_{\tau^*} \leftarrow \mathcal{R}_{\mathsf{Enc}}$ and compute $\mathsf{ct}_{\tau^*} \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau^*}, x_{\mathsf{coin}}; r'_{\tau^*})$, $\mathsf{K}' \leftarrow \mathsf{PRF.Gen}'(1^\lambda)$, $\mathsf{F}'_{\neq\tau^*} \leftarrow \mathsf{Puncture}'(\mathsf{K}', \tau^*)$, and $\widehat{\mathsf{ct}} \leftarrow i\mathcal{O}_2(\mathsf{E}^{**}_{\mathsf{1fe}}[\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{ct}_{\tau^*}])$ where $\mathsf{E}^{**}_{\mathsf{1fe}}$ is described in Figure 8. Set $\mathsf{ct} := \widehat{\mathsf{ct}}$.

From the definitions of $\mathsf{E}^{*}_{\mathsf{1fe}}$, $\mathsf{E}^{**}_{\mathsf{1fe}}$, and $\mathsf{Hyb}_3^{i^*}$, the behaviors of $\mathsf{E}^{*}_{\mathsf{1fe}}$ in $\mathsf{Hyb}_3^{i^*}$ is the same as that of $\mathsf{E}^{**}_{\mathsf{1fe}}$ in $\mathsf{Hyb}_4^{i^*}$. Moreover, both circuits have the same size by padding $\mathsf{pad}_{\mathsf{E}}$. Then, we can use the indistinguishability of $i\mathcal{O}_2$, and it holds that $D^{(4,i^*)}$ is computationally indistinguishable from $D^{(3,i^*)}$. This change is for erasing information about coin from $\mathsf{E}^{*}_{\mathsf{1fe}}$.

Hyb_6**:** This is the same as $\mathsf{Hyb}_4^{i^*}$ except that we choose PRF keys $\mathsf{K}_1, \mathsf{K}_2 \leftarrow \{0,1\}^\lambda$ at the beginning of the game and change the distribution $D^{(4,i^*)}$ over pairs $(\mathsf{coin}, \mathsf{ct})$ into $D^{(6)}$ as follows.

- Sample a uniform $\mathsf{coin} \leftarrow \{0,1\}$.
- Sample $r'_{\tau^*} \leftarrow \mathcal{R}_{\mathsf{Enc}}$, compute $\mathsf{ct}_{\tau^*} := \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau^*}, x_{\mathsf{coin}}; r'_{\tau^*})$, $r_1 \leftarrow \mathsf{F}^{(1)}_{\mathsf{K}_1}(\mathsf{ct}_{\tau^*})$, $r_2 \leftarrow \mathsf{F}^{(2)}_{\mathsf{K}_2}(\mathsf{ct}_{\tau^*})$, $\mathsf{K}' \leftarrow \mathsf{PRF}'.\mathsf{Gen}(1^\lambda; r_1)$, $\mathsf{F}'_{\neq\tau^*} \leftarrow \mathsf{Puncture}'(\mathsf{K}', \tau^*)$, and $\widehat{\mathsf{ct}} := i\mathcal{O}_2(\mathsf{E}^{**}_{\mathsf{1fe}}[\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{ct}_{\tau^*}]; r_2)$ where $\mathsf{E}^{**}_{\mathsf{1fe}}$ is described in Figure 8. Set $\mathsf{ct} := \widehat{\mathsf{ct}}$.

That is, we de-randomize generation of K' and $\widehat{\mathsf{ct}}$. Note that the puncturing algorithm $\mathsf{Puncture}'$ is deterministic in the GGM construction based puncturable PRF [GGM86, BW13, KPTZ13, BGI14]. By the pseudorandomness of $\mathsf{F}^{(1)}$ and $\mathsf{F}^{(2)}$, $D^{(6)}$ is computationally indistinguishable from $D^{(4,i^*)}$.

There are $O(2^\ell)$ hybrid games between Hyb_0 and Hyb_6. However, if all building blocks are sub-exponentially secure, $D^{(6)}$ and D (the original distribution in Hyb_0) are computationally indistinguishable since we can set $\ell = \mathsf{polylog}(\lambda)$. Note that we do not need the sub-exponential security for the challenge-only strong $\gamma/2$-anti-piracy security of 1SDFE since we do not use the anti-piracy security so far.

Thus, by Theorem 7.10, it holds that

$$\begin{aligned}&\mathrm{Tr}\left[(\mathcal{TI}_{\frac{1}{2}+\gamma-\epsilon}(\mathcal{P}_{1,D^{(6)}}) \otimes \mathcal{TI}_{\frac{1}{2}+\gamma-\epsilon}(\mathcal{P}_{2,D^{(6)}}))q\right]\\ &\geq \mathrm{Tr}\left[(\mathcal{TI}_{\frac{1}{2}+\gamma}(\mathcal{P}_{1,D}) \otimes \mathcal{TI}_{\frac{1}{2}+\gamma}(\mathcal{P}_{2,D}))q\right] - \delta,\end{aligned}$$

where δ is some negligible function. That is, it holds that $\mathsf{Adv}_6 \geq \mathsf{Adv}_0 - \mathsf{negl}(\lambda)$.

We prove the following.

Lemma 7.40. *If* 1SDFE *is challenge-only strong* $\gamma/2$*-anti-piracy secure, it holds that* $\mathsf{Adv}_6 \leq \mathsf{negl}(\lambda)$.

Proof of Lemma 7.40. Suppose that Adv_6 is non-negligible for a contradiction. We construct a QPT algorithm $\mathcal{B}$ for the strong $\gamma/2$-anti-piracy game of 1SDFE by using the adversary $\mathcal{A}$ of the strong γ-anti-piracy game of SDFE as follows.

1. $\mathcal{B}$ chooses $\tau_1, \ldots, \tau_q, \tau^* \leftarrow \{0,1\}^\ell$ and $\mathsf{K}_1, \mathsf{K}_2 \leftarrow \{0,1\}^\lambda$, and generates $\mathsf{K} \leftarrow \mathsf{PRF.Gen}(1^\lambda)$.
2. $\mathcal{B}$ generates $\mathsf{F}_{\neq\tau^*} \leftarrow \mathsf{Puncture}(\mathsf{K}, \tau^*)$.
3. $\mathcal{B}$ receives pk^* from its challenger, sets $\mathsf{pk}_{\tau^*} := \mathsf{pk}^*$, and constructs $\mathsf{S}^*_{1\mathsf{fe}}[\tau^*, \mathsf{F}_{\neq\tau^*}, \mathsf{pk}_{\tau^*}]$ described in Figure 6 by choosing τ^* for the challenge query f^*. $\mathcal{B}$ sets $\widehat{\mathsf{pk}} := i\mathcal{O}_1(\mathsf{S}^*_{1\mathsf{fe}}[\tau^*, \mathsf{F}_{\neq\tau^*}, \mathsf{pk}_{\tau^*}])$ and sends it to $\mathcal{A}$.
4. When $\mathcal{A}$ sends a key query f_i, $\mathcal{B}$ generates $(\mathsf{pk}_{\tau_i}, \mathsf{msk}_{\tau_i}) \leftarrow \mathsf{1SDFE.Setup}(1^\lambda; \mathsf{PRF}_{\mathsf{K}_{\neq\tau^*}}(\tau_i))$ and $\mathcal{sk}_{f_i,\tau_i} \leftarrow \mathsf{1SDFE}.\mathcal{QKG}(\mathsf{msk}_{\tau_i}, f_i)$, sets $\widehat{\mathcal{sk}}_{f_i,\tau_i} := (\tau_i, \mathcal{sk}_{f_i,\tau_i})$, and sends $\widehat{\mathcal{sk}}_{f_i}$ to $\mathcal{A}$.
5. When $\mathcal{A}$ sends the challenge query f^*, $\mathcal{B}$ passes it to its challenger and receives $\mathcal{sk}_{f^*} \leftarrow \mathsf{1SDFE}.\mathcal{QKG}(\mathsf{msk}^*, f^*)$, and sends $\widehat{\mathcal{sk}}_{f^*} := (\tau^*, \mathcal{sk}_{f^*})$ to $\mathcal{A}$.
6. Again, $\mathcal{B}$ can answer key queries f_i from $\mathcal{A}$ as the fourth item.
7. At some point, $\mathcal{B}$ receives (x_0, x_1) and two (possibly entangled) quantum decryptors $\mathcal{D}_1 = (q[\mathsf{R}_1], \boldsymbol{U}_1)$ and $\mathcal{D}_2 = (q[\mathsf{R}_2], \boldsymbol{U}_2)$, where $f^*(x_0) \neq f^*(x_1)$, from $\mathcal{A}$. For $i \in \{1,2\}$, let $\mathcal{P}_{i,D^{(6)}}$ be the following mixture of projective measurements acting on some quantum state q':
   - Sample a uniform $\mathsf{coin} \leftarrow \{0,1\}$.
   - Sample $r'_{\tau^*} \leftarrow \mathcal{R}_{\mathsf{Enc}}$, $\mathsf{ct}_{\tau^*} \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau^*}, x_{\mathsf{coin}}; r'_{\tau^*})$.
   - Generate $r_1 \leftarrow \mathsf{F}^{(1)}_{\mathsf{K}_1}(\mathsf{ct}_{\tau^*})$, $r_2 \leftarrow \mathsf{F}^{(2)}_{\mathsf{K}_2}(\mathsf{ct}_{\tau^*})$, $\mathsf{K}' \leftarrow \mathsf{PRF.Gen}'(1^\lambda; r_1)$, and $\mathsf{F}'_{\neq\tau^*} = \mathsf{Puncture}'(\mathsf{K}', \tau^*)$.
   - Generate $\widehat{\mathsf{ct}} \leftarrow i\mathcal{O}_2(\mathsf{E}^{**}_{1\mathsf{fe}}[\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{ct}_{\tau^*}]; r_2)$ where $\mathsf{E}^{**}_{1\mathsf{fe}}$ is described in Figure 8.
   - Run the quantum decryptor $(q', \boldsymbol{U}_i)$ on input $\widehat{\mathsf{ct}}$. If the outcome is coin, output 1. Otherwise, output 0.
8. $\mathcal{B}$ sets quantum decryptors $\mathcal{D}^*_1$ and $\mathcal{D}^*_2$ as follows.

   In this item, we denote pure state $|x\rangle\langle x|$ by $|x\rangle$ for ease of notation. First, $\mathcal{B}$ constructs $(q^*[\mathsf{R}_i], \boldsymbol{U}^*_i)$ such that

   - Set $q^*[\mathsf{R}_i] := q[\mathsf{R}_i] \otimes \left|0^{|\widehat{\mathsf{ct}}|}\right\rangle \otimes \left|\tau^*, \widehat{\mathsf{pk}}, 0^\ell, x_1, \mathsf{K}_1, \mathsf{K}_2, 0^{|r_1|}, 0^{|r_2|}, 0^\ell\right\rangle \otimes |\mathsf{ct}_{\tau^*}\rangle$. Note that $|\mathsf{ct}_{\tau^*}\rangle$ is the input for $(q^*[\mathsf{R}_i], \boldsymbol{U}^*_i)$.
   - Unitary $\boldsymbol{U}^*_i$ acts on $q^*[\mathsf{R}_i]$ and the result is
   $$q'[\mathsf{R}_i] := q[\mathsf{R}_i] \otimes |\widehat{\mathsf{ct}}\rangle \otimes \left|\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{K}_1, \mathsf{K}_2, r_1, r_2, \mathsf{K}'\right\rangle \otimes |\mathsf{ct}_{\tau^*}\rangle,$$
   where $r_1 := \mathsf{F}^{(1)}_{\mathsf{K}_1}(\mathsf{ct}_{\tau^*})$, $r_2 := \mathsf{F}^{(2)}_{\mathsf{K}_2}(\mathsf{ct}_{\tau^*})$, $\mathsf{K}' := \mathsf{PRF.Gen}'(1^\lambda; r_1)$, $\mathsf{F}'_{\neq\tau^*} := \mathsf{Puncture}'(\mathsf{K}', \tau^*)$, and $\widehat{\mathsf{ct}} := i\mathcal{O}_2(\mathsf{E}^{**}_{1\mathsf{fe}}[\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{ct}_{\tau^*}]; r_2)$. That is, $\boldsymbol{U}^*_i$ constructs $\mathsf{E}^{**}_{1\mathsf{fe}}[\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{ct}_{\tau^*}]$ described in Figure 8 and generates $\widehat{\mathsf{ct}} := i\mathcal{O}_2(\mathsf{E}^{**}_{1\mathsf{fe}}[\tau^*, \widehat{\mathsf{pk}}, \mathsf{F}'_{\neq\tau^*}, x_1, \mathsf{ct}_{\tau^*}]; r_2)$. Note that $\mathcal{B}$ can construct it since $\mathcal{B}$ has τ^*, $\widehat{\mathsf{pk}}$, K_1, K_2, and x_1.

It is easy to see that if we apply $(\boldsymbol{U}_i \otimes \boldsymbol{I})$ to $q'[\mathsf{R}_i]$, we obtain an output of $\mathcal{D}_i$ for input $\widehat{\mathsf{ct}}$. Thus, $\mathcal{B}$ sets $\mathcal{D}_1^* := (q^*[\mathsf{R}_1], (\boldsymbol{U}_1 \otimes \boldsymbol{I})\boldsymbol{U}_1^*)$ and $\mathcal{D}_2^* := (q^*[\mathsf{R}_2], (\boldsymbol{U}_2 \otimes \boldsymbol{I})\boldsymbol{U}_2^*)$. $\mathcal{B}$ sends (x_0, x_1) and $\mathcal{D}_1^*$ and $\mathcal{D}_2^*$ to its challenger.

The challenger of 1SDFE runs the test by using the following. Let $\mathcal{P}_{i,D'}$ be the following mixture of projective measurements acting on some quantum state q_{1SDFE}:

- Sample a uniformly random coin $\leftarrow \{0,1\}$ and compute $\mathsf{ct}_{\tau^*} \leftarrow \mathsf{1SDFE.Enc}(\mathsf{pk}_{\tau^*}, x_{\mathsf{coin}})$.
- Run $\mathsf{coin}' \leftarrow \mathcal{D}^*(\mathsf{ct})$. If $\mathsf{coin}' = \mathsf{coin}$, output 1, otherwise output 0.

By the construction of $\boldsymbol{U}_i^*$, quantum decryptor $\mathcal{D}_i^*$ takes a ciphertext ct_{τ^*} under pk_{τ^*} as input, converts it into a ciphertext $\widehat{\mathsf{ct}}$ of SDFE, and apply the quantum decryptor $\mathcal{D}_i$ from $\mathcal{A}$. The converted ciphertext $\widehat{\mathsf{ct}}$ perfectly simulates a ciphertext in Hyb_6 if an input to $\mathcal{D}_i^*$ is a ciphertext under pk_{τ^*}.

We assumed that Adv_6 is non-negligible at the beginning of this proof. That is, applying $\mathcal{TI}_{\frac{1}{2}+\gamma-\epsilon}(\mathcal{P}_{i,D^{(6)}})$ on $q^*[\mathsf{R}_i]$ results in two outcomes 1 with non-negligible probability. That is, it holds that

$$\mathrm{Tr}\left[\mathcal{TI}_{\frac{1}{2}+\gamma-\epsilon}(\mathcal{P}_{1,D^{(6)}}) \otimes \mathcal{TI}_{\frac{1}{2}+\gamma-\epsilon}(\mathcal{P}_{2,D^{(6)}}) q^*\right] > \mathsf{negl}(\lambda).$$

This means that for $i \in \{1,2\}$, $q^*[\mathsf{R}_i]$ is a $(\gamma - \epsilon)$-good distinguisher with respect to ciphertexts generated according to $D^{(6)}$.

Thus, if $\mathcal{D}_i$ works as a $(\gamma - \epsilon)$-good quantum distinguisher for SDFE, $\mathcal{D}_i^*$ works as a $(\gamma - \epsilon)$-good quantum distinguisher for 1SDFE, where $\epsilon = \frac{\gamma}{2}$. This completes the proof of Lemma 7.40. ■

By using the relationships among $\mathsf{Adv}_{\mathsf{SDFE},\mathcal{A}}$, Adv_x^i for $x \in \{1, \cdots, 5\}$ and $i \in [I_{i^*}]$, $\mathsf{Adv}_x^{i^*}$ for $x \in \{1, \cdots, 4\}$, and Adv_6 that we show above, we obtain $\mathsf{Adv}_{\mathsf{SDFE},\mathcal{A}}^{\mathsf{strong\text{-}anti\text{-}piracy}}(\lambda, \gamma) \leq \mathsf{negl}(\lambda)$. We almost complete the proof of Theorem 7.39. Lastly, we need to set the padding parameters for IO security.

Padding Parameter. The proof of security relies on the indistinguishability of obfuscated $\mathsf{S}_{\mathsf{1fe}}$ and $\mathsf{S}_{\mathsf{1fe}}^*$ defined in Figures 2 and 6, and that of obfuscated $\mathsf{E}_{\mathsf{1fe}}$, $\mathsf{E}_{\mathsf{1fe}}^*$, and $\mathsf{E}_{\mathsf{1fe}}^{**}$ defined in Figures 3, 7 and 8. Accordingly, we set $\mathsf{pad}_{\mathsf{S}} := \max(|\mathsf{S}_{\mathsf{1fe}}|, |\mathsf{S}_{\mathsf{1fe}}^*|)$ and $\mathsf{pad}_{\mathsf{E}} := \max(|\mathsf{E}_{\mathsf{1fe}}|, |\mathsf{E}_{\mathsf{1fe}}^*|, |\mathsf{E}_{\mathsf{1fe}}^{**}|)$.

The circuits $\mathsf{S}_{\mathsf{1fe}}$ and $\mathsf{S}_{\mathsf{1fe}}^*$ compute a puncturable PRF over domain $\{0,1\}^\ell$ and a key pair of 1FE, and may have punctured PRF keys and a public key hardwired. The circuits $\mathsf{E}_{\mathsf{1fe}}$, $\mathsf{E}_{\mathsf{1fe}}^*$, and $\mathsf{E}_{\mathsf{1fe}}^{**}$ run the circuit $\widehat{\mathsf{pk}}$ and compute a puncturable PRF over domain $\{0,1\}^\ell$ and a ciphertext of 1SDFE, and may have punctured PRF keys, tags, plaintexts, and a hard-wired ciphertext. Note that ℓ is a polynomial of λ. Thus, it holds that

$$\begin{aligned}\mathsf{pad}_{\mathsf{S}} &\leq \mathrm{poly}(\lambda, n, s),\\ \mathsf{pad}_{\mathsf{E}} &\leq \mathrm{poly}(\lambda, n, s, \left|\widehat{\mathsf{pk}}\right|).\end{aligned}$$

Therefore, we complete the proof of Theorem 7.39. ■

We obtain the following corollary from Corollary 7.34 and Theorems 7.37 and 7.39.

Corollary 7.41. *Assuming the existence of sub-exponentially secure IO for* $\mathsf{P/poly}$*, and the sub-exponential hardness of QLWE, there exists an SDFE scheme for* $\mathsf{P/poly}$ *that satisfies adaptive security (Definition 7.28) and strong γ-anti-piracy security for any inverse polynomial γ.*

References

[Aar09] Scott Aaronson. Quantum copy-protection and quantum money. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009*, pages 229–242. IEEE Computer Society, 2009. (Cited on page 2, 8.)

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Heidelberg, March / April 2015. (Cited on page 9.)

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015. (Cited on page 6, 9, 16, 57, 59.)

[AGKZ20] Ryan Amos, Marios Georgiou, Aggelos Kiayias, and Mark Zhandry. One-shot signatures and applications to hybrid quantum/classical authentication. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd ACM STOC*, pages 255–268. ACM Press, June 2020. (Cited on page 8.)

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, August 2015. (Cited on page 9.)

[AJL$^+$19] Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 284–332. Springer, Heidelberg, August 2019. (Cited on page 2, 5, 9.)

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation from functional encryption for simple functions. Cryptology ePrint Archive, Report 2015/730, 2015. `https://eprint.iacr.org/2015/730`. (Cited on page 9.)

[AL21] Prabhanjan Ananth and Rolando L. La Placa. Secure software leasing. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 501–530. Springer, Heidelberg, October 2021. (Cited on page 1, 2, 8.)

[ALL$^+$21] Scott Aaronson, Jiahui Liu, Qipeng Liu, Mark Zhandry, and Ruizhe Zhang. New approaches for quantum copy-protection. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 526–555, Virtual Event, August 2021. Springer, Heidelberg. (Cited on page 1, 8, 29, 30, 31, 41.)

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016. (Cited on page 9.)

[AMVY21] Shweta Agrawal, Monosij Maitra, Narasimha Sai Vempati, and Shota Yamada. Functional encryption for turing machines with dynamic bounded collusion from LWE. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 239–269, Virtual Event, August 2021. Springer, Heidelberg. (Cited on page 4, 9.)

[AP20] Shweta Agrawal and Alice Pellet-Mary. Indistinguishability obfuscation without maps: Attacks and fixes for noisy linear FE. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 110–140. Springer, Heidelberg, May 2020. (Cited on page 8, 33.)

[AR17] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 173–205. Springer, Heidelberg, November 2017. (Cited on page 9.)

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 174–198. Springer, Heidelberg, December 2019. (Cited on page 2, 5, 9, 17, 18.)

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2):6:1–6:48, 2012. (Cited on page 9, 32.)

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519. Springer, Heidelberg, March 2014. (Cited on page 31, 32, 48.)

[BGMZ18] James Bartusek, Jiaxin Guan, Fermi Ma, and Mark Zhandry. Return of GGH15: Provable security against zeroizing attacks. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 544–574. Springer, Heidelberg, November 2018. (Cited on page 8, 33.)

[BI20] Anne Broadbent and Rabib Islam. Quantum encryption with certified deletion. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 92–122. Springer, Heidelberg, November 2020. (Cited on page 2, 4, 8.)

[BJL$^+$21] Anne Broadbent, Stacey Jeffery, Sébastien Lord, Supartha Podder, and Aarthi Sundaram. Secure software leasing without assumptions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 90–120. Springer, Heidelberg, November 2021. (Cited on page 1, 8.)

[BNPW20] Nir Bitansky, Ryo Nishimaki, Alain Passelègue, and Daniel Wichs. From cryptomania to obfustopia through secret-key functional encryption. *Journal of Cryptology*, 33(2):357–405, April 2020. (Cited on page 3, 7.)

[BS17] Shalev Ben-David and Or Sattath. Quantum tokens for digital signatures. Cryptology ePrint Archive, Report 2017/094, 2017. `https://eprint.iacr.org/2017/094`. (Cited on page 8.)

[BS18] Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. *Journal of Cryptology*, 31(1):202–225, January 2018. (Cited on page 9, 17.)

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011. (Cited on page 1, 9, 16.)

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, Heidelberg, August 2011. (Cited on page 9.)

[BV18] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *Journal of the ACM*, 65(6):39:1–39:37, 2018. (Cited on page 9.)

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 280–300. Springer, Heidelberg, December 2013. (Cited on page 31, 32, 48.)

[CGO21] Michele Ciampi, Vipul Goyal, and Rafail Ostrovsky. Threshold garbled circuits and ad hoc secure computation. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 64–93. Springer, Heidelberg, October 2021. (Cited on page 6.)

[CHVW19] Yilei Chen, Minki Hhan, Vinod Vaikuntanathan, and Hoeteck Wee. Matrix PRFs: Constructions, attacks, and applications to obfuscation. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 55–80. Springer, Heidelberg, December 2019. (Cited on page 8, 33.)

[CLLZ21] Andrea Coladangelo, Jiahui Liu, Qipeng Liu, and Mark Zhandry. Hidden cosets and applications to unclonable cryptography. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 556–584, Virtual Event, August 2021. Springer, Heidelberg. (Cited on page 2, 3, 7, 8, 30, 33, 34, 35, 36, 41, 56.)

[CMP20] Andrea Coladangelo, Christian Majenz, and Alexander Poremba. Quantum copy-protection of compute-and-compare programs in the quantum random oracle model. Cryptology ePrint Archive, Report 2020/1194, 2020. `https://eprint.iacr.org/2020/1194`. (Cited on page 1, 8.)

[CS19] R. Joseph Connor and Max Schuchard. Blind bernoulli trials: A noninteractive protocol for hidden-weight coin flips. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019*, pages 1483–1500. USENIX Association, August 2019. (Cited on page 1.)

[CV21] Eric Culf and Thomas Vidick. A monogamy-of-entanglement game for subspace coset states. *arXiv (CoRR)*, abs/2107.13324, 2021. (Cited on page 3, 8, 34, 35.)

[DIJ$^+$13] Angelo De Caro, Vincenzo Iovino, Abhishek Jain, Adam O'Neill, Omer Paneth, and Giuseppe Persiano. On the achievability of simulation-based security for functional encryption. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 519–535. Springer, Heidelberg, August 2013. (Cited on page 16, 60, 61, 62.)

[DQV$^+$21] Lalita Devadas, Willy Quach, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Succinct LWE sampling, random polynomials, and obfuscation. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 256–287. Springer, Heidelberg, November 2021. (Cited on page 8, 33.)

[GGH$^+$16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016. (Cited on page 9.)

[GGHW17] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. *Algorithmica*, 79(4):1353–1373, 2017. (Cited on page 8.)

[GGLW21] Rachit Garg, Rishab Goyal, George Lu, and Brent Waters. Dynamic collusion bounded functional encryption from identity-based encryption. Cryptology ePrint Archive, Report 2021/847, 2021. https://eprint.iacr.org/2021/847. (Cited on page 4, 9, 16.)

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, 1986. (Cited on page 10, 31, 32, 48.)

[GKP$^+$13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 536–553. Springer, Heidelberg, August 2013. (Cited on page 8.)

[GKW17] Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th FOCS*, pages 612–621. IEEE Computer Society Press, October 2017. (Cited on page 36.)

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, August 2012. (Cited on page 2, 3, 5, 9, 11, 16, 17, 18, 32.)

[GZ20] Marios Georgiou and Mark Zhandry. Unclonable decryption keys. Cryptology ePrint Archive, Report 2020/877, 2020. https://eprint.iacr.org/2020/877. (Cited on page 8, 33.)

[HMNY21] Taiga Hiroka, Tomoyuki Morimae, Ryo Nishimaki, and Takashi Yamakawa. Quantum encryption with certified deletion, revisited: Public key, attribute-based, and classical communication. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part I*, volume 13090 of *LNCS*, pages 606–636. Springer, Heidelberg, December 2021. (Cited on page 2, 4, 8, 12, 13.)

[JKMS20] Aayush Jain, Alexis Korb, Nathan Manohar, and Amit Sahai. Amplifying the security of functional encryption, unconditionally. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 717–746. Springer, Heidelberg, August 2020. (Cited on page 2, 5, 6, 9, 15, 26.)

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC 2021*, pages 60–73. ACM, 2021. (Cited on page 8, 9.)

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Heidelberg, May / June 2022. (Cited on page 8, 9.)

[KLM$^+$18] Sam Kim, Kevin Lewi, Avradip Mandal, Hart Montgomery, Arnab Roy, and David J. Wu. Function-hiding inner product encryption is practical. In Dario Catalano and Roberto De Prisco, editors, *SCN 18*, volume 11035 of *LNCS*, pages 544–562. Springer, Heidelberg, September 2018. (Cited on page 1.)

[KNT21] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Simple and generic constructions of succinct functional encryption. *Journal of Cryptology*, 34(3):25, July 2021. (Cited on page 3, 7.)

[KNT22] Fuyuki Kitagawa, Ryo Nishimaki, and Keisuke Tanaka. Obfustopia built on secret-key functional encryption. *J. Cryptol.*, 35(3):19, 2022. (Cited on page 9.)

[KNY21] Fuyuki Kitagawa, Ryo Nishimaki, and Takashi Yamakawa. Secure software leasing from standard assumptions. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 31–61. Springer, Heidelberg, November 2021. (Cited on page 1, 8.)

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 669–684. ACM Press, November 2013. (Cited on page 31, 32, 48.)

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016. (Cited on page 6.)

[NNL01] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 41–62. Springer, Heidelberg, August 2001. (Cited on page 9.)

[NP01] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In Yair Frankel, editor, *FC 2000*, volume 1962 of *LNCS*, pages 1–20. Springer, Heidelberg, February 2001. (Cited on page 1, 9.)

[NWZ16] Ryo Nishimaki, Daniel Wichs, and Mark Zhandry. Anonymous traitor tracing: How to embed arbitrary information in a key. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 388–419. Springer, Heidelberg, May 2016. (Cited on page 1, 9.)

[RPB$^+$19] Théo Ryffel, David Pointcheval, Francis R. Bach, Edouard Dufour-Sans, and Romain Gay. Partially encrypted deep learning using functional encryption. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS 2019*, pages 4519–4530, 2019. (Cited on page 1.)

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 463–472. ACM Press, October 2010. (Cited on page 3, 9.)

[SSW12] Amit Sahai, Hakan Seyalioglu, and Brent Waters. Dynamic credentials and ciphertext delegation for attribute-based encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 199–217. Springer, Heidelberg, August 2012. (Cited on page 1.)

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005. (Cited on page 1.)

[SW21] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. *SIAM J. Comput.*, 50(3):857–908, 2021. (Cited on page 7.)

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th FOCS*, pages 600–611. IEEE Computer Society Press, October 2017. (Cited on page 36.)

[Zha20] Mark Zhandry. Schrödinger's pirate: How to trace a quantum decoder. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 61–91. Springer, Heidelberg, November 2020. (Cited on page 30, 31.)

A CPA-Style Anti-Piracy Security Notions

We introduce the CPA-style anti-piracy security for SDE.

Definition A.1 (Anti-Piracy Security, CPA-Style [CLLZ21]). *We consider the CPA-style anti-piracy game* $\mathsf{Exp}^{\mathsf{anti\text{-}piracy\text{-}cpa}}_{\mathsf{SDE},\mathcal{A}}(\lambda)$ *between the challenger and an adversary* $\mathcal{A}$ *below.*

1. *The challenger generates* $(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda)$.
2. *The challenger generates* $\mathcal{sk} \leftarrow \mathcal{QKG}(\mathsf{sk})$ *and sends* $(\mathsf{pk},\mathcal{sk})$ *to* $\mathcal{A}$.
3. $\mathcal{A}$ *outputs* (m_0,m_1) *and two (possibly entangled) quantum decryptors* $\mathcal{D}_1 = (q[\mathsf{R}_1],\boldsymbol{U}_1)$ *and* $\mathcal{D}_2 = (q[\mathsf{R}_2],\boldsymbol{U}_2)$, *where* $m_0 \neq m_1$, $|m_0| = |m_1|$, q *is a quantum state over registers* R_1 *and* R_2, *and* $\boldsymbol{U}_1$ *and* $\boldsymbol{U}_2$ *are general quantum circuits.*
4. *The challenger chooses* $\mathsf{coin}_1,\mathsf{coin}_2 \leftarrow \{0,1\}$, *and generates* $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{pk},m_{\mathsf{coin}_1})$ *and* $\mathsf{ct}_2 \leftarrow \mathsf{Enc}(\mathsf{pk},m_{\mathsf{coin}_2})$. *The challenger runs* $m'_i \leftarrow \mathcal{D}_i(\mathsf{ct}_i)$ *for* $i \in \{1,2\}$. *If* $m'_i = m_{\mathsf{coin}_i}$ *for* $i \in \{1,2\}$, *the challenger outputs* 1, *otherwise outputs* 0.

We say that SDE *is* γ*-anti-piracy secure if for any QPT adversary* $\mathcal{A}$, *it satisfies that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{anti\text{-}piracy\text{-}cpa}}_{\mathsf{SDE},\mathcal{A}}(\lambda) = 1\right] \leq \frac{1}{2} + \gamma(\lambda) + \mathsf{negl}(\lambda).$$

Theorem A.2 ([CLLZ21]). *If an SDE scheme satisfies strong* γ*-anti-piracy security, it also satisfies* γ*-anti-piracry security (in the CPA-style).*

We can define a similar security notion for SDFE.

Definition A.3 (Anti-Piracy Security for FE, CPA-Style). *We consider the CPA-style anti-piracy game for FE* $\mathsf{Exp}^{\mathsf{anti\text{-}piracy\text{-}cpa}}_{\mathsf{SDFE},\mathcal{A}}(\lambda)$ *between the challenger and an adversary* $\mathcal{A}$ *below.*

1. *The challenger generates* $(\mathsf{pk},\mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and sends* pk *to* $\mathcal{A}$.
2. $\mathcal{A}$ *sends key queries* f_i *to the challenger and receives* $\mathcal{sk}_{f_i} \leftarrow \mathcal{QKG}(\mathsf{sk}_{f_i})$ *where* $\mathsf{sk}_{f_i} \leftarrow \mathsf{KG}(\mathsf{msk},f_i)$.
3. *At some point* $\mathcal{A}$ *sends a challenge query* f^* *to the challenger and receives* $\mathcal{sk}_{f^*} \leftarrow \mathcal{QKG}(\mathsf{msk},f^*)$.
4. *Again,* $\mathcal{A}$ *sends* f_i *to the challenger and receives* $\mathcal{sk}_{f_i} \leftarrow \mathcal{QKG}(\mathsf{sk}_{f_i})$ *where* $\mathsf{sk}_{f_i} \leftarrow \mathsf{KG}(\mathsf{msk},f_i)$.
5. $\mathcal{A}$ *outputs* (x_0,x_1) *and two (possibly entangled) quantum decryptors* $\mathcal{D}_1 = (q[\mathsf{R}_1],\boldsymbol{U}_1)$ *and* $\mathcal{D}_2 = (q[\mathsf{R}_2],\boldsymbol{U}_2)$, *where* $\forall i\ f_i(x_0) = f_i(x_1)$, $f^*(x_0) \neq f^*(x_1)$, q *is a quantum state over registers* R_1 *and* R_2, *and* $\boldsymbol{U}_1$ *and* $\boldsymbol{U}_2$ *are general quantum circuits.*
6. *The challenger chooses* $\mathsf{coin}_1,\mathsf{coin}_2 \leftarrow \{0,1\}$, *and generates* $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{pk},x_{\mathsf{coin}_1})$ *and* $\mathsf{ct}_2 \leftarrow \mathsf{Enc}(\mathsf{pk},x_{\mathsf{coin}_2})$. *The challenger runs* $\mathsf{coin}'_k \leftarrow \mathcal{D}_k(\mathsf{ct}_k)$ *for* $k \in \{1,2\}$. *If* $\mathsf{coin}'_k = \mathsf{coin}_k$ *for* $k \in \{1,2\}$, *the challenger outputs* 1, *otherwise outputs* 0.

We say that SDFE *is* γ*-anti-piracy secure if for any QPT adversary* $\mathcal{A}$, *it satisfies that*

$$\Pr\left[\mathsf{Exp}^{\mathsf{anti\text{-}piracy\text{-}cpa}}_{\mathsf{SDFE},\mathcal{A}}(\lambda) = 1\right] \leq \frac{1}{2} + \gamma(\lambda) + \mathsf{negl}(\lambda).$$

If $\mathcal{A}$ *can send only the challenge query* f^* *during the experiment, we say that* SDFE *is challenge-only* γ*-anti-piracy secure.*

We can also define the CPA-style anti-piracy for the predictor-based quantum decryptors as Definition A.1. That is, $\mathcal{D}_i$ outputs $f^*(x_{\mathsf{coin}_i})$ for $i \in \{1,2\}$ instead of coin_i. We omit the variation.

Theorem A.4. *If an SDFE scheme satisfies strong γ-anti-piracy security, it also satisfies γ-anti-piracy security in the CPA-style.*

The proof of this theorem is almost the same as that of Theorem A.2, so we omit it to avoid replication.

B Adaptive and Simulation-based Security for Secure Key Leasing

This section presents how to upgrade FE with secure key leasing security.

B.1 Adaptive Security for Secure Key Leasing

Ananth et al. [ABSV15] present transformations from selectively secure FE into adaptively secure FE. Almost the same transformation works for FE with secure key leasing. The adaptive lessor security for SKFE with SKL is almost the same as Definition 3.2 except that $\mathcal{A}$ declares (x_0^*, x_1^*) when $\mathcal{A}$ sends cert to the challenger and we separate the key oracle into two types. The formal definition is as follows.

Definition B.1 (Adaptive Lessor Security). *We say that* $\mathsf{SKFE\text{-}SKL}$ *is an adaptively lessor secure SKFE-SKL scheme for $\mathcal{X}, \mathcal{Y}$, and $\mathcal{F}$, if it satisfies the following requirement, formalized from the experiment* $\mathsf{Exp}^{\mathsf{ada\text{-}lessor}}_{\mathcal{A},\mathsf{SKFE\text{-}SKL}}(1^\lambda, \mathsf{coin})$ *between an adversary $\mathcal{A}$ and a challenger:*

1. *At the beginning, $\mathcal{A}$ sends 1^{q+q^*} to the challenger. The challenger runs* $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{q+q^*})$. *Throughout the experiment, $\mathcal{A}$ can access the following oracles.*

   $O_{\mathsf{Enc}}(x)$**:** *Given x, it returns* $\mathsf{Enc}(\mathsf{msk}, x)$.

   $O_{\mathcal{KG}}(f, 1^n)$**:** *Given $(f, 1^n)$, it generates* $(\mathit{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f, 1^n)$, *and sends fsk to $\mathcal{A}$. $\mathcal{A}$ can access this oracle at most q times.*

2. *$\mathcal{A}$ can access the following oracles before $\mathcal{A}$ declares the challenge plaintexts.*

   $O^*_{\mathcal{KG}}(f^*, 1^{n^*})$**:** *Given $(f^*, 1^{n^*})$, it generates* $(\mathit{fsk}^*, \mathsf{vk}^*) \leftarrow \mathcal{KG}(\mathsf{msk}, f^*, 1^{n^*})$, *adds* $(f^*, 1^{n^*}, \mathsf{vk}^*, \bot)$ *to $L_{\mathcal{KG}}$, and sends fsk^* to $\mathcal{A}$. $\mathcal{A}$ can access this oracle at most q^* times.*

   $O_{\mathsf{Vrfy}}(f^*, \mathsf{cert}^*)$**:** *Given* (f^*, cert^*), *it finds an entry* $(f^*, 1^{n^*}, \mathsf{vk}^*, M)$ *from $L_{\mathcal{KG}}$. (If there is no such entry, it returns $\bot$.) If* $\top = \mathsf{Vrfy}(\mathsf{vk}^*, \mathsf{cert}^*)$ *and the number of queries to O_{Enc} at this point is less than n^*, it returns $\top$ and updates the entry into* $(f^*, 1^{n^*}, \mathsf{vk}^*, \top)$. *Otherwise, it returns $\bot$.*

3. *When $\mathcal{A}$ sends (x_0^*, x_1^*) to receive the challenge ciphertext, if there exists* $(f^*, 1^{n^*}, \mathsf{vk}^*, \bot)$ *in $L_{\mathcal{KG}}$, or $O_{\mathcal{KG}}$ received $(f, 1^n)$ such that $f(x_0^*) \neq f(x_1^*)$, the challenger outputs 0 as the final output of this experiment. Otherwise, the challenger generates* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{msk}, x^*_{\mathsf{coin}})$ *and sends ct^* to $\mathcal{A}$. Hereafter, $O_{\mathcal{KG}}$ rejects a query $(f, 1^n)$ such that $f(x_0^*) \neq f(x_1^*)$ and $\mathcal{A}$ cannot access $O^*_{\mathcal{KG}}$ and O_{Vrfy}.*

4. *$\mathcal{A}$ outputs a guess coin' for coin. The challenger outputs coin' as the final output of the experiment.*

For any QPT $\mathcal{A}$, it holds that

$$\mathsf{Adv}^{\mathsf{ada\text{-}lessor}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathsf{Exp}^{\mathsf{ada\text{-}lessor}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}(1^\lambda, 0) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{ada\text{-}lessor}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}(1^\lambda, 1) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Remark B.2. The definition above considers two key oracles $O_{\mathcal{KG}}$ and $O^*_{\mathcal{KG}}$ unlike Definition 3.2 to distinguish a functional decryption key which is deleted from one which is not deleted. The oracle $O^*_{\mathcal{KG}}$ accepts a query $(f^*, 1^{n^*})$ such that $f^*(x_0^*) \neq f^*(x_1^*)$. We need to distinguish these two types of queries since we need to know whether $f(x_0^*) = f(x_1^*)$ or not when an adversary sends a query in the security proof of the construction below. It is not clear whether we can achieve a stronger adaptive security where adversaries access a single key generation oracle as Definition 3.2. We leave it as an open question.

We construct an adaptively secure SKFE scheme with secure key leasing $\mathsf{SKFE\text{-}SKL} = (\mathsf{Setup}, \mathcal{KG}, \mathsf{Enc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$ using the following tools:

- A selectively secure SKFE scheme with secure key leasing $\mathsf{Sel.SKFE} = \mathsf{Sel}.(\mathsf{Setup}, \mathcal{KG}, \mathsf{Enc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$.
- An adaptively single-ciphertext secure SKFE scheme $\mathsf{1CT.SKFE} = \mathsf{1CT}.(\mathsf{Setup}, \mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$.
- A pseudorandom-secure SKE scheme $\mathsf{SKE} = \mathsf{SKE}.(\mathsf{E}, \mathsf{D})$.
- A PRF F.

Let ℓ_1 and ℓ_2 be the length of SKE ciphertexts and the length of random tags, respectively. The definition of adaptive lessor security is a natural adaptive variant of Definition 3.2. The description of $\mathsf{SKFE\text{-}SKL}$ is as follows.

$\mathsf{Setup}(1^\lambda)$:

- Generate $\mathsf{sel.msk} \leftarrow \mathsf{Sel.Setup}(1^\lambda)$.
- Output $\mathsf{msk} := \mathsf{sel.msk}$.

$\mathcal{KG}(\mathsf{msk}, f, 1^n)$:

- Parse $\mathsf{sel.msk} \leftarrow \mathsf{msk}$.
- Generate $\mathsf{ct_{ske}} \leftarrow \{0,1\}^{\ell_1}$ and choose $\tau \leftarrow \{0,1\}^{\ell_2}$.
- Compute $(\mathsf{sel}.\mathcal{sk}_G, \mathsf{sel.vk}) \leftarrow \mathsf{Sel}.\mathcal{KG}(\mathsf{sel.msk}, G[f, \mathsf{ct_{ske}}, \tau], 1^n)$, where the circuit G is described in Figure 9.
- Output $\mathcal{sk}_f := \mathsf{sel}.\mathcal{sk}_G$ and $\mathsf{vk} := \mathsf{sel.vk}$.

$\mathsf{Enc}(\mathsf{msk}, x)$:

- Parse $\mathsf{sel.msk} \leftarrow \mathsf{msk}$.
- Generate $\mathsf{1ct.msk} \leftarrow \mathsf{1CT.Setup}(1^\lambda)$ and choose $\mathsf{K} \leftarrow \{0,1\}^\lambda$.
- Generate $\mathsf{1ct.ct} \leftarrow \mathsf{1CT.Enc}(\mathsf{1ct.msk}, x)$.
- Generate $\mathsf{sel.ct} \leftarrow \mathsf{Sel.Enc}(\mathsf{sel.msk}, (\mathsf{1ct.msk}, \mathsf{K}, 0^\lambda, 0))$.
- Output $\mathsf{ct}_x := (\mathsf{1ct.ct}, \mathsf{sel.ct})$.

$\mathcal{Dec}(\mathcal{sk}_f, \mathsf{ct}_x)$:

- Parse $\mathsf{sel}.\mathcal{sk}_G \leftarrow \mathcal{sk}_f$ and $(\mathsf{1ct.ct}, \mathsf{sel.ct}) \leftarrow \mathsf{ct}_x$.
- Compute $\mathsf{1ct.sk}_f \leftarrow \mathsf{Sel}.\mathcal{Dec}(\mathsf{sel}.\mathcal{sk}_G, \mathsf{sel.ct})$.
- Output $y \leftarrow \mathsf{1CT.Dec}(\mathsf{1ct.sk}_f, \mathsf{1ct.ct})$.

$\mathcal{Cert}(\mathcal{sk}_f)$:

- Parse $\mathsf{sel}.\mathcal{sk}_f \leftarrow \mathcal{sk}_f$.

Key Generation Circuit $G[f, \mathsf{ct}_{\mathsf{ske}}, \tau](1\mathsf{ct.msk}, \mathsf{K}, K_{\mathsf{ske}}, \beta)$

Hardwired: function f, SKE ciphertext $\mathsf{ct}_{\mathsf{ske}}$, tag τ.

Input: master secret key $1\mathsf{ct.msk}$, PRF key K, SKE secret key K_{ske}, bit β.

1. If $\beta = 0$, output $1\mathsf{ct.sk}_f \leftarrow 1\mathsf{CT.KG}(1\mathsf{ct.msk}, f; \mathsf{F_K}(\tau))$.
2. Else, output $\mathsf{SKE.D}(K_{\mathsf{ske}}, \mathsf{ct}_{\mathsf{ske}})$.

Figure 9: Description of $G[f, \mathsf{ct}_{\mathsf{ske}}, \tau]$.

- Output $\mathsf{cert} := \mathsf{sel.cert} \leftarrow \mathsf{Sel}.\mathcal{Cert}(\mathsf{sel}.\mathcal{sk}_f)$.

$\mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$:

- Parse $\mathsf{sel.vk} = \mathsf{vk}$ and $\mathsf{sel.cert} = \mathsf{cert}$.
- Output $\top/\bot \leftarrow \mathsf{Sel.Vrfy}(\mathsf{sel.vk}, \mathsf{sel.cert})$.

We emphasize that $1\mathsf{CT.SKFE}$ is a classical standard SKFE scheme, and $1\mathsf{CT.KG}$ is a classical algorithm. This does not spoil the lessor security of $\mathsf{Sel.SKFE}$ because $1\mathsf{ct.sk}_f$ generated by the circuit G depends on $1\mathsf{ct.msk}$ and K from the encryption algorithm and τ from the key generation algorithm. At the decryption phase, we can obtain $1\mathsf{ct.sk}_f$ under $1\mathsf{ct.msk}$ from $\mathsf{Sel}.\mathcal{KG}(\mathsf{sel.msk}, G, 1^n)$ and

$$(1\mathsf{CT.Enc}(1\mathsf{ct.msk}, x), \mathsf{Sel.Enc}(\mathsf{sel.msk}, (1\mathsf{ct.msk}, \mathsf{K}, 0^\lambda, 0))).$$

We can keep a copy of $1\mathsf{ct.sk}_f$. However, it does not help us to decrypt another ciphertext

$$(1\mathsf{CT.Enc}(1\mathsf{ct.msk}', x'), \mathsf{Sel.Enc}(\mathsf{sel.msk}, (1\mathsf{ct.msk}', \mathsf{K}', 0^\lambda, 0))).$$

Note that $1\mathsf{ct.msk}$ is freshly generated at the encryption phase. That is, a functional decryption key $1\mathsf{ct.sk}_f$ is specific to only one pair of $\mathcal{sk}_f$ and ct_x

Thus, after we securely delete $\mathcal{sk}_f$ by using the deletion algorithm $\mathsf{Sel}.\mathcal{Cert}$ of $\mathsf{Sel.SKFE}$, we can no longer decrypt a fresh ciphertext generated after the deletion. Since the deletion power is preserved, we can upgrade the selective security of $\mathsf{Sel.SKFE}$ to adaptive security by leveraging the adaptive security of $1\mathsf{CT.SKFE}$ as the transformation by Ananth et al. [ABSV15].

Theorem B.3. *If* $\mathsf{Sel.SKFE}$ *is selectively lessor secure,* $1\mathsf{CT.SKFE}$ *is adaptively single-ciphertext secure and collusion-resistant,* SKE *is pseudorandom-secure, and* F *is a pseudorandom function,* $\mathsf{SKFE\text{-}SKL}$ *above is adaptively lessor secure.*

Since the proof is almost the same as that by Ananth et al. [ABSV15, Theorem 4 in the eprint ver.], we omit it to avoid the replication. Note that the reduction needs to embed $1\mathsf{ct.sk}_f \leftarrow 1\mathsf{CT.KG}(1\mathsf{ct.msk}, f; \mathsf{F_K}(\tau))$ into $\mathsf{ct}_{\mathsf{ske}}$ for a key query such that $f(x_0^*) = f(x_1^*)$, but $\mathsf{ct}_{\mathsf{ske}}$ should be a junk ciphertext for a key query such that $f(x_0^*) \neq f(x_1^*)$. The reduction knows which simulation-way is correct since Definition B.1 distinguishes two types of queries.

B.2 Simulation-based Security for Secure Key Leasing

Although we present indistinguishability-based security definitions for SKFE with secure key leasing in Section 3, we can also consider simulation-based security definitions for SKFE with secure key leasing. We present the definition of adaptive lessor simulation-security.

Definition B.4 (Adaptive Lessor Simulation-security). *Let* $\mathsf{SKFE\text{-}SKL}$ *be an SKFE scheme with secure key leasing. For a stateful QPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *and a stateful QPT Simulator* $\mathcal{S} = (\mathcal{SEnc}, \mathcal{SKG})$*, we consider the two experiments described in Figure 10, where*

$\mathsf{Real}^{\textsf{ad-sim}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}(\lambda)$	$\mathsf{Sim}^{\textsf{ad-sim}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A},\mathcal{S}}(\lambda)$
1. $1^{q+q^*} \leftarrow \mathcal{A}_1(1^\lambda)$	1. $1^{q+q^*} \leftarrow \mathcal{A}_1(1^\lambda)$
2. $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{q+q^*})$	2. $\mathsf{msk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{q+q^*})$
3. $x^* \leftarrow \mathcal{A}_2^{\mathsf{Enc}(\mathsf{msk},\cdot),\mathcal{KG}(\mathsf{msk},\cdot,\cdot),O^*_{\mathcal{KG}}(\cdot,\cdot),O_{\mathsf{Vrfy}}(\cdot,\cdot)}$	3. $x^* \leftarrow \mathcal{A}_2^{\mathsf{Enc}(\mathsf{msk},\cdot),\mathcal{KG}(\mathsf{msk},\cdot,\cdot),O^*_{\mathcal{KG}}(\cdot,\cdot),O_{\mathsf{Vrfy}}(\cdot,\cdot)}$
4. Output 0 if there exists $(f^*, 1^{n^*}, \mathsf{vk}^*, \perp)$ in $L_{\mathcal{KG}}$. Otherwise, go to the next step.	4. Output 0 if there exists $(f^*, 1^{n^*}, \mathsf{vk}^*, \perp)$ in $L_{\mathcal{KG}}$. Otherwise, go to the next step.
5.	5. Let $\mathcal{Q}$ be the query/answer list for $\mathsf{Enc}(\mathsf{msk},\cdot)$
6.	6. Let $(f_i, 1^{n_i})_{i\in[q]}$ be the queries for $\mathsf{KG}(\mathsf{msk},\cdot,\cdot)$
7.	7. $y_i := f_i(x^*)$ for every $i \in [q]$
8. $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{msk}, x^*)$	8. $(\mathsf{ct}^*, \mathsf{st}) \leftarrow \mathcal{SEnc}(\mathsf{msk}, \mathcal{Q}, (f_i, 1^{n_i}, y_i)_{i\in[q]})$
9. Output $b \leftarrow \mathcal{A}_3(\mathsf{ct}^*)^{\mathsf{Enc}(\mathsf{msk},\cdot),\mathcal{KG}(\mathsf{msk},\cdot,\cdot)}$	9. Output $b \leftarrow \mathcal{A}_3(\mathsf{ct}^*)^{\mathsf{Enc}(\mathsf{msk},\cdot),O_{\mathcal{SKG}}(\cdot,\cdot)}$

Figure 10: Security experiments for adaptively lessor simulation-secure SKFE-SKL

- *$\mathcal{A}$ is allowed to make total at most q queries to $\mathcal{KG}(\mathsf{msk},\cdot,\cdot)$ (resp. $\mathcal{KG}(\mathsf{msk},\cdot,\cdot)$ and $O_{\mathcal{SKG}}(\cdot,\cdot)$) in $\mathsf{Real}^{\textsf{ad-sim}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}(\lambda)$ (resp. $\mathsf{Sim}^{\textsf{ad-sim}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A},\mathcal{S}}(\lambda)$),*
- *$\mathcal{A}$ is allowed to make total at most q^* queries to $O^*_{\mathcal{KG}}(\cdot,\cdot)$, which is the same as $\mathcal{KG}(\mathsf{msk},\cdot,\cdot)$ except that it also adds $(f^*, 1^{n^*}, \mathsf{vk}^*, \perp)$ to $L_{\mathcal{KG}}$ when it is invoked,*
- *$\mathcal{A}$ is allowed to make queries to $O_{\mathsf{Vrfy}}(\cdot,\cdot)$, which is given (f^*, cert^*), it finds an entry $(f^*, 1^{n^*}, \mathsf{vk}^*, M)$ from $L_{\mathcal{KG}}$, and does the following: If there is no such entry, it returns $\perp$. If $\top = \mathsf{Vrfy}(\mathsf{vk}^*, \mathsf{cert}^*)$ and the number of queries to O_{Enc} at this point is less than n^*, it returns $\top$ and updates the entry into $(f^*, 1^{n^*}, \mathsf{vk}^*, \top)$. Otherwise, it returns $\perp$,*
- *$O_{\mathcal{SKG}}(f, 1^n) = \mathcal{SKG}(\mathsf{st}, f, 1^n, f(x^*))$.*

We say that SKFE-SKL *is adaptively lessor simulation-secure if there exists a QPT simulator $\mathcal{S}$ such that for any QPT adversary $\mathcal{A}$, it satisfies that*

$$\left|\Pr\left[\mathsf{Real}^{\textsf{ad-sim}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A}}(\lambda) = 1\right] - \Pr\left[\mathsf{Sim}^{\textsf{ad-sim}}_{\mathsf{SKFE\text{-}SKL},\mathcal{A},\mathcal{S}}(\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

This is a natural simulation-based variant of adaptive lessor (indistinguishability-)security. The simulator $\mathcal{S} = (\mathcal{SEnc}, \mathcal{SKG})$ does not have $f_j^*(x^*)$, and the simulated ciphertext does not reveal information about $f_j^*(x^*)$.

Remark B.5. We consider two key oracles $\mathcal{KG}$ and $O^*_{\mathcal{KG}}$ as Definition B.1 to distinguish two types of queries. We can achieve this simulation-based security since we use adaptively lessor secure SKFE-SKL in the sense of Definition B.1 as a building block below. It is also an open question to achieve a stronger adaptive simulation-based security where adversaries access a single key generation oracle as Definition 3.2.

De Caro, Iovino, Jani, O'Neil, Paneth, and Persiano [DIJ+13] present a transformation from indistinguishability-based secure FE to simulation-based secure FE. The transformation works if the number of key queries before a challenge ciphertext is given is a-priori bounded. We can apply almost the same transformation to SKFE with secure leasing since we modify a plaintext in the ciphertext and a function embedded in a functional decryption key.

We construct an adaptively lessor simulation-secure SKFE scheme with secure key leasing $\mathsf{adaSKL} = \mathsf{adaSKL}.(\mathsf{Setup}, \mathcal{KG}, \mathsf{Enc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$ using the following tools:

- An adaptively lessor (indistinguishability-)secure SKFE scheme with secure key leasing $\mathsf{indSKL} = \mathsf{indSKL}.(\mathsf{Setup}, \mathcal{KG}, \mathsf{Enc}, \mathcal{Dec}, \mathcal{Cert}, \mathsf{Vrfy})$.
- A pseudorandom-secure SKE scheme $\mathsf{SKE} = \mathsf{SKE}.(\mathsf{E}, \mathsf{D})$.

Let ℓ_1, ℓ_2, and m be the length of SKE ciphertexts, the length of random tags, and the output length, respectively. The description of adaSKL is as follows.

$\mathsf{adaSKL}.\mathsf{Setup}(1^\lambda)$:

- Generate $\mathsf{ind.msk} \leftarrow \mathsf{indSKL}.\mathsf{Setup}(1^\lambda)$.
- Output $\mathsf{msk} := \mathsf{ind.msk}$.

$\mathsf{adaSKL}.\mathcal{KG}(\mathsf{msk}, f, 1^n)$:

- Parse $\mathsf{ind.msk} \leftarrow \mathsf{msk}$.
- Generate $\mathsf{ct}_{\mathsf{ske}} \leftarrow \{0,1\}^{\ell_1}$ and choose $\tau \leftarrow \{0,1\}^{\ell_2}$.
- Compute $(\mathsf{ind}.\mathcal{sk}_T, \mathsf{ind.vk}) \leftarrow \mathsf{indSKL}.\mathcal{KG}(\mathsf{sel.msk}, T[f, \mathsf{ct}_{\mathsf{ske}}, \tau], 1^n)$, where the circuit T is described in Figure 11.
- Output $\mathcal{sk}_f := \mathsf{ind}.\mathcal{sk}_T$ and $\mathsf{vk} := \mathsf{ind.vk}$.

$\mathsf{adaSKL}.\mathsf{Enc}(\mathsf{msk}, x)$:

- Parse $(\mathsf{ind.msk}, K_{\mathsf{ske}}) \leftarrow \mathsf{msk}$.
- Set $x' := (x, 0^\lambda, (0^{\ell_2}, 0^m), \ldots, (0^{\ell_2}, 0^m), 0)$, where m is the output length of functions.
- Generate $\mathsf{ind.ct} \leftarrow \mathsf{indSKL}.\mathsf{Enc}(\mathsf{ind.msk}, x')$.
- Output $\mathsf{ct}_x := \mathsf{ind.ct}$.

$\mathsf{adaSKL}.\mathcal{Dec}(\mathcal{sk}_f, \mathsf{ct}_x)$:

- Parse $\mathsf{ind}.\mathcal{sk}_T \leftarrow \mathcal{sk}_f$ and $\mathsf{ind.ct} \leftarrow \mathsf{ct}_x$.
- Output $y \leftarrow \mathsf{indSKL}.\mathcal{Dec}(\mathsf{ind}.\mathcal{sk}_T, \mathsf{ind.ct})$.

$\mathsf{adaSKL}.\mathcal{Cert}(\mathcal{sk}_f)$:

- Parse $\mathsf{ind}.\mathcal{sk}_f \leftarrow \mathcal{sk}_f$.
- Output $\mathsf{cert} := \mathsf{ind.cert} \leftarrow \mathsf{indSKL}.\mathcal{Cert}(\mathsf{ind}.\mathcal{sk}_f)$.

$\mathsf{adaSKL}.\mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert})$:

- Parse $\mathsf{ind.vk} = \mathsf{vk}$ and $\mathsf{ind.cert} = \mathsf{cert}$.
- Output $\top/\bot \leftarrow \mathsf{indSKL}.\mathsf{Vrfy}(\mathsf{ind.vk}, \mathsf{ind.cert})$.

As we see above, we expand the plaintext space and use the trapdoor circuit T as De Caro et al. [DIJ+13]. We can construct a QPT simulator as follows.

1. Generates $\mathsf{ind.msk} \leftarrow \mathsf{indSKL}.\mathsf{Setup}(1^\lambda)$ and $K_{\mathsf{ske}} \leftarrow \{0,1\}^\lambda$.
2. Receives $\mathcal{sk}_{f_i} = (\mathsf{ind}.\mathcal{sk}_{f_i}, \mathsf{ct}_{\mathsf{ske},i}, \tau_i)$ for $i \in [q_{\mathsf{pre}}]$ and outputs for pre-challenge key queries, that is, $f_1(x^*), \ldots, f_{q_{\mathsf{pre}}}(x^*)$.
3. Generates a challenge ciphertext $\mathsf{indSKL}.\mathsf{Enc}(\mathsf{ind.msk}, (0, K_{\mathsf{ske}}, (\tau_1, f_1(x^*)), \ldots, (\tau_{q_{\mathsf{pre}}}, f_{q_{\mathsf{pre}}}(x^*)), 1))$.

Trapdoor Circuit $T[f, \mathsf{ct}_{\mathsf{ske}}, \tau](x, K_{\mathsf{ske}}, (\tau_1, y_1), \ldots, (\tau_q, y_q), \beta)$

Hardwired: function f, SKE ciphertext $\mathsf{ct}_{\mathsf{ske}}$, tag τ.
Input: plaintext x, SKE secret key K_{ske}, q pairs of tag and output $(\tau_1, y_1), \ldots, (\tau_q, y_q)$, bit β.

```
1. If $\beta = 1$, do the following
    • If there exists $i$ such that $\tau = \tau_i$, output $y_i$.
    • Else output $y' \leftarrow \mathsf{SKE.D}(K_{\mathsf{ske}}, \mathsf{ct}_{\mathsf{ske}})$.
2. Else, output $f(x)$.
```

Figure 11: Description of $T[f, \mathsf{ct}_{\mathsf{ske}}, \tau]$.

4. Simulates functional decryption keys for post-challenge key queries as follows. When an output value $f_i'(x^*)$ for a key query f_i' is given, generates $(\mathsf{ind}.\mathcal{sk}_T, \mathsf{ind.vk}) \leftarrow \mathsf{Sel}.\mathcal{KG}(\mathsf{sel.msk}, T[f_i', \mathsf{ct}_{\mathsf{ske},i}', \tau_i'], 1^n)$ where $\mathsf{ct}_{\mathsf{ske},i}' \leftarrow \mathsf{SKE.E}(K_{\mathsf{ske}}, f_i'(x^*))$ and $\tau_i' \leftarrow \{0,1\}^{\ell_2}$.

Note that the simulator should not have $f_j^*(x^*)$ for any $j \in [q^*]$ since $\{\mathcal{sk}_{f_j^*}\}_{j \in [q^*]}$ are deleted and the challenge ciphertext should not reveal information about $\{f_j^*(x^*)\}_{j \in [q^*]}$, where $\{f_j^*\}_{j \in [q^*]}$ are the challenge functions, and the simulation-security is satisfied.

Since the flag $\beta = 1$ in the simulated challenge ciphertext, the trapdoor circuit T works at the trapdoor branch. For functional decryption keys before the challenge ciphertext, T works at the first item in the $\beta = 1$ branch since τ_i is a tag embedded in the functional decryption keys for pre-challenge key queries. So, the output of T is an embedded $f_i(x^*)$. This is consistent. For functional decryption keys after the challenge ciphertext, T works at the second item in the $\beta = 1$ branch since τ_i' is uniformly random and must be different from τ_i except negligible probability. In addition, $y' = f_i'(x^*)$ since $\mathsf{ct}_{\mathsf{ske}}' \leftarrow \mathsf{SKE.E}(K_{\mathsf{ske}}, f_i'(x^*))$. This is consistent. Thus, the simulated ciphertext does not include information about $\{f_j^*(x^*)\}_{j \in [q^*]}$.

In reducing to the adaptive lessor (indistinguishability)-security, the simulator can simulate the key generation and encryption oracles in the simulation-based game by using the oracles in the indistinguishability-based game.

Theorem B.6. *If* indSKL *is adaptively lessor (indistinguishability-)secure and* SKE *is pseuodrandom-secure,* adaSKL *is adaptively lessor simulation-secure.*

Since the proof is almost the same as that by De Caro et al. [DIJ+13, Theorem 14 in the eprint ver.], we omit it to avoid the replication.