Functional Encryption with Secure Key Leasing

Fuyuki Kitagawa[1] and Ryo Nishimaki[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 P/poly with secure key leasing based on post-quantum one-way functions since we can instantiate bounded collusion-resistant SKFE for 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 P/poly from post-quantum indistinguishability obfuscation and quantum hardness of the learning with errors problem.

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

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

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 $\mathsf{P/poly}$ with secure key leasing from OWFs since we can instantiate bounded collusion-resistant SKFE for $\mathsf{P/poly}$ 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 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 $\mathsf{P/poly}$ 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 $\mathsf{P/poly}$ into *single-key* SDFE for $\mathsf{P/poly}$ by using SDE. Then, we transform single-key SDFE $\mathsf{P/poly}$ into collusion-resistant SDFE for $\mathsf{P/poly}$ by using an IO-based key bundling technique [KNT21,BNPW20].

[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. Many-ciphertext means that SKFE is secure even if adversaries can send unbounded polynomially many queries to an encryption oracle.

We can instantiate SDE with IO and the QLWE assumption [CLLZ21,CV21] and single-key PKFE for $\mathsf{P/poly}$ 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 $(\mathcal{fsk}, \mathsf{vk}) \leftarrow \mathcal{KG}(\mathsf{msk}, f, 1^n)$, returns $\mathcal{fsk}$ 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 the full version, 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 2.2. 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 $\mathsf{zSKFE\text{-}sbSKL}$ 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 $\mathsf{zSKFE\text{-}sbSKL}$, 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 $\mathsf{zSKFE\text{-}sbSKL}$ 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 $(\mathsf{cd}.\mathit{ct}, \mathsf{vk}) \leftarrow \mathsf{CD}.\mathcal{Enc}(\mathsf{cd.sk}, \mathsf{fsk})$. The resulting decryption key is $\mathit{zfsk} := \mathsf{cd}.\mathit{ct}$ 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 $\mathsf{cd.ct}$ and $\mathsf{cd.sk}$, 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 $\mathsf{zSKFE\text{-}sbSKL}$ 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 $\mathit{zfsk} := \mathsf{cd}.\mathit{ct}$, where $\mathsf{fsk} \leftarrow \mathsf{KG}(\mathsf{msk}, f)$ and $(\mathsf{cd}.\mathit{ct}, \mathsf{vk}) \leftarrow \mathsf{CD}.\mathcal{Enc}(\mathsf{cd.sk}, \mathsf{fsk})$. 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 $\mathsf{cd.sk}$ 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 $\mathsf{zSKFE\text{-}sbSKL}$ 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 2.3. 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 $\mathsf{zSKFE\text{-}sbSKL}$ in parallel. Concretely, suppose we use $N = \alpha n$ instances of $\mathsf{zSKFE\text{-}sbSKL}$ to achieve a scheme with availability bound n, where $\alpha \in \mathbb{N}$. To generate a decryption key for f, we generate

$(\mathit{zfsk}_j, \mathsf{vk}_j) \leftarrow \mathit{zKG}(\mathsf{zmsk}_j, f)$ for every $j \in [N]$, and set the resulting decryption key as $(\mathit{zfsk}_j)_{j\in[N]}$ 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 $(\mathit{zfsk}_j)_{j\in[N]}$, we just compute $f(x) \leftarrow \mathit{zDec}(\mathit{zfsk}_j, \mathsf{zct}_j)$. 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 $\mathsf{zSKFE\text{-}sbSKL}$ 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 4 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 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

[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].

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 $(\mathit{fsk}_i)_{i\in[m]}$, we can obtain $f_i(s_i)$ by decrypting ct_i with 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 sk_{f^*}. More specifically, $\mathcal{A}$ is given sk_{f^*} and outputs two possibly entangled quantum distinguishers $\mathcal{D}_1$ 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, sk_{f^*} is copy-protected. If both $\mathcal{D}_1$ and $\mathcal{D}_2$ have 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

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

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 $\mathit{sk}_{f,\tau}$ under msk_τ. A functional decryption key of our collusion-resistant scheme consists of $(\tau, \mathit{sk}_{f,\tau})$. 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 $\mathit{sk}_{f,\tau}$ 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.[6] 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 we need the sub-exponential security for building blocks.

1.4 Organization

Due to the space limitation, we focus on SKFE-SKL in the rest of this version, and we provide results on single decryptor FE in the full version. For the high-level overview of our single-decryptor FE, please see Section 1.3. Also, please refer the full version for preliminaries including notations.

In Section 2, we provide the definition of SKFE-SKL, and its variants SKFE-sbSKL and index-based SKFE-sbSKL. In Section 3, we construct an index-based SKFE-sbSKL scheme. In Section 4, we show how to transform an index-based SKFE-sbSKL scheme into an SKFE-sbSKL scheme. In Section 5, we show how to construct an SKFE-SKL scheme from an SKFE-sbSKL scheme.

2 Definition of SKFE with Secure Key Leasing

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

2.1 SKFE with Secure Key Leasing

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

Definition 2.1 (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.*

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

$\mathcal{KG}(\mathsf{msk}, f, 1^n) \to (\mathit{fsk}, \mathsf{vk})$**:** *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*.*

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

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

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

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

$\mathsf{Vrfy}(\mathsf{vk}, \mathsf{cert}) \to \top/\bot$**:** *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 2.2 (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*

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

Remark 2.1 (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 the full version for the definition and transformation.

Remark 2.2 (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 the full version for the simulation-based definition and the transformation.

2.2 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. [7]

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 2.3 (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.*

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 $\mathsf{skl.msk} \coloneqq (\mathsf{msk}_k)_{k\in[\lambda]}$. To encrypt x by $\mathsf{SKFE\text{-}SKL}$, we encrypt it by all

[7] 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.

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 SKFE-SKL for a function f and an availability bound n, we first compute $k' \in [\lambda]$ such that $2^{k'-1} \leq n \leq 2^{k'}$. 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 SKFE-SKL are simply those of SKFE-SKL.

We now consider the security proof of SKFE-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 SKFE-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 SKFE-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 2.4 (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 2.2 , 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 2.3 (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 2.4.

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.

2.3 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 2.5 (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 2.6 (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*

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

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

We present our index-based SFFE-sbSKL scheme in this section.

Tool. First, we introduce the definitions for reusable SKE with certified deletion introduced by Hiroka et al. [HMNY21]

Definition 3.1 (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* $\mathsf{sk} \in \mathcal{K}$*.*
$\mathcal{Enc}(\mathsf{sk}, m) \to (\mathsf{vk}, \mathit{ct})$: *The encryption algorithm takes as input* sk *and a plaintext* $m \in \mathcal{M}$ *and outputs a verification key* vk *and a ciphertext* ct*.*
$\mathcal{Dec}(\mathsf{sk}, \mathit{ct}) \to m'$: *The decryption algorithm takes as input* sk *and* ct *and outputs a plaintext* $m' \in \mathcal{M}$ *or* $\bot$*.*
$\mathcal{Del}(\mathit{ct}) \to \mathsf{cert}$: *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* $m \in \mathcal{M}$*,*

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

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

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

We introduce a variant of certified deletion 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 this section.

Definition 3.2 (IND-CVA-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* $\mathsf{Exp}^{\mathsf{sk\text{-}cert\text{-}vo}}_{\Sigma,\mathcal{A}}(\lambda, b)$.

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* $(\mathsf{vk}, \mathsf{ct}) \leftarrow \mathcal{Enc}(\mathsf{sk}, m)$ *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* $(\mathsf{vk}_b, ct_b) \leftarrow \mathcal{Enc}(\mathsf{sk}, m_b)$ *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*

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

Theorem 3.1. *Known reusable SKE with certified deletion scheme [HMNY21] satisfies IND-CVA-CD security.*

We prove this theorem in the full version.

Scheme description. 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 $(\mathsf{cd}.\mathit{ct}_j, \mathsf{vk}_j) \leftarrow \mathsf{CD}.\mathcal{E}\mathit{nc}(\mathsf{cd.sk}_j, \mathsf{fsk}_j)$ for every $j \in [n]$.
– Output $\mathsf{skl}.\mathit{fsk} := (\mathsf{cd}.\mathit{ct}_j)_{j\in[n]}$ 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 $\mathsf{skl.ct} := (j, \mathsf{ct}_j, \mathsf{cd.sk}_j)$.

$\mathit{iDec}(\mathsf{skl}.\mathit{fsk}, \mathsf{skl.ct})$**:**

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

$\mathit{iCert}(\mathsf{skl}.\mathit{fsk})$**:**

– Parse $(\mathsf{cd}.\mathit{ct}_j)_{j\in[n]} \leftarrow \mathsf{skl}.\mathit{fsk}$.
– Compute $\mathsf{cert}_j \leftarrow \mathsf{CD}.\mathcal{D}\mathit{el}(\mathsf{cd.ct}_j)$ 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 3.2. *If* SKFE *is selective indistinguishability-secure,* CDSKE *is IND-CVA-CD secure,*[8] *and* F *is a secure PRF, then* iSKFE-sbSKL *satisfies selective strong lessor security.*

Proof of Theorem 3.2. 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 $(\mathsf{cd}.\mathit{ct}_j, \mathsf{vk}_j) \leftarrow \mathsf{CD}.\mathcal{E}\mathit{nc}(\mathsf{cd.sk}_j, \mathsf{fsk}_j)$ for every $j \in [n]$.

[8] See Definition 3.2 for the defition of IND-CVA-CD.

– Sets $\mathsf{skl}.\mathit{fsk} := (\mathsf{cd}.\mathit{ct}_j)_{j\in[n]}$ and $\mathsf{skl}.\mathsf{vk} := (\mathsf{vk}_j)_{j\in[n]}$.

It sends $\mathsf{skl}.\mathit{fsk}$ to $\mathcal{A}$ and adds $(f, \mathsf{skl}.\mathsf{vk}, \bot)$ 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 $\mathsf{skl}.\mathsf{ct}^* := (j^*, \mathsf{ct}_{j^*}^*, \mathsf{cd}.\mathsf{sk}_{j^*})$ 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 $\mathsf{cd}.\mathit{ct}_{j^*}$ included in $\mathsf{skl}.\mathit{fsk} := (\mathsf{cd}.\mathit{ct}_j)_{j\in[n]}$ as $(\mathsf{cd}.\mathit{ct}_{j^*}, \mathsf{vk}_{j^*}) \leftarrow \mathsf{CD}.\mathcal{E}\mathit{nc}(\mathsf{cd}.\mathsf{sk}_{j^*}, \mathbf{0})$.

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 $\mathsf{cd}.\mathsf{sk}_{j^*}$. If $\mathcal{A}$ is valid, $\mathcal{A}$ cannot obtain $\mathsf{cd}.\mathsf{sk}_{j^*}$ before $\mathcal{A}$ is given $\mathsf{skl}.\mathsf{ct}^*$, and $\mathcal{A}$ returns all ciphertexts under $\mathsf{cd}.\mathsf{sk}_{j^*}$ before it gets $\mathsf{cd}.\mathsf{sk}_{j^*}$. 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 $\mathsf{cd}.\mathsf{sk}_{j^*}$.

Hyb_3**:** This hybrid is the same as Hyb_2 except that the challenger generates $\mathsf{ct}_{j^*}^*$ included in $\mathsf{skl}.\mathsf{ct}^*$ 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. ■

4 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 SKFE-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 $\mathsf{sbskl.vk} := (\mathsf{vk}_i)_{i\in[m]}$.

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

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

$\mathsf{sbSKL}.\mathcal{Dec}(\mathsf{sbskl}.\mathit{fsk}, \mathsf{sbskl.ct})$:

- Parse $(\mathit{fsk}_i)_{i\in[m]} \leftarrow \mathsf{sbskl}.\mathit{fsk}$ and $(\mathsf{ct}_i)_{i\in[m]} \leftarrow \mathsf{sbskl.ct}$.
- Compute $y_{\boldsymbol{I}} \leftarrow i\mathcal{Dec}(\mathit{fsk}_i, \mathsf{ct}_i)$ for every $i \in [m]$.
- Output $y \leftarrow \mathsf{Decode}((y_i)_{i\in[m]})$.

$\mathsf{sbSKL}.\mathcal{Cert}(\mathsf{sbskl}.\mathit{fsk})$:

- Parse $(\mathit{fsk}_i)_{i\in[m]} \leftarrow \mathsf{sbskl}.\mathit{fsk}$.
- Compute $\mathsf{cert}_i \leftarrow i\mathcal{Cert}(\mathit{fsk}_i)$ 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 SKFE-sbSKL. Let $\mathsf{sbskl}.\mathit{fsk} := (\mathit{fsk}_i)_{i\in[m]}$ 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 iSKFE-sbSKL, we obtain $f_i(s_i)$ by decrypting ct_i with fsk_i for every $i \in [m]$, where $(f_i)_{i\in[m]} \leftarrow \mathsf{FuncEncode}(\mathsf{params}, f)$ and $(s_i)_{i\in[m]} \leftarrow$

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 $f_i(s_i)$.

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

$\mathsf{InpEncode}(\mathsf{params}, x)$. Thus, we obtains $f(x) \leftarrow \mathsf{Decode}((f_i(s_i))_{i\in[m]})$ from the correctness of SetHSS. It is clear that SKFE-sbSKL also satisfies verification correctness.

Also, the weak optimal efficiency of SKFE-sbSKL easily follows from that of iSKFE-sbSKL since the running time of algorithms of SetHSS is independent of n. Note that 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 4.1. *If* iSKFE-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* SKFE-sbSKL *is selectively strong lessor secure.*

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

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

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 $k_i \leftarrow [N]$ for every $i \in [m]$.
   - Generate $\mathsf{ct}_i^k \leftarrow \mathsf{iEnc}(\mathsf{msk}, i, k_i, (s_i^k, \mathbf{0}, 0))$ 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 $(\mathit{fsk}_i, \mathsf{vk}_i) \leftarrow \mathit{i}\mathcal{KG}(\mathsf{msk}_i, F[f_i, \mathsf{sct}_i])$ 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 $*_i \leftarrow [N]$ 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 $i^* \neq i^k$ 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 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 i^k for every $i \in [m]$ and $k \in [n]$ and 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 $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, *_i, (\mathbf{0}, K, 1))$ 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 4.1.

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 4.2.

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

Proof of Proposition 4.1. 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 $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, *_i, (\mathbf{0}, K, 1))$ for every i such that $i \in S_{\mathtt{secure}}$ and $i \le i'$.

Then, we have

$$\begin{aligned}&|\Pr[\mathsf{Hyb}_2 = 1] - \Pr[\mathsf{Hyb}_3 = 1]| \\ &\le \sum_{i' \in m} \left|\Pr\left[\mathsf{Hyb}_{2,i'-1} = 1 \wedge i' \in S_{\mathtt{secure}}\right] - \Pr\left[\mathsf{Hyb}_{2,i} = 1 \wedge i' \in S_{\mathtt{secure}}\right]\right|, \quad (1)\end{aligned}$$

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 $\mathsf{params} \coloneqq (p, \ell, (T_i)_{i\in[m]}) \leftarrow \mathsf{SetGen}(1^\lambda)$. $\mathcal{B}$ generates $i^k \leftarrow [N]$ for every $i \in [m]$ and $k \in [n]$ and $i^* \leftarrow [N]$ 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 $(1^q, N, i'^*, (s_{i'}^*, \mathbf{0}, 0), (\mathbf{0}, K, 1))$. $\mathcal{B}$ 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 $k \le n$, use $(i^k)_{i\in[m]}$ generated at the beginning. Otherwise, Generate $k_i \leftarrow [N]$ for every $i \in [m]$.
   - Query $(k_{i'}, (s_{i'}^k, \mathbf{0}, 0))$ to its encryption oracle and obtain $\mathsf{ct}_{i'}^k$.
   - Generate $\mathsf{ct}_i^k \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, i^k, (s_i^k, \mathbf{0}, 0))$ for every $i \in [m] \setminus \{i'\}$.
   - Set $\mathsf{sbskl.ct}^k \coloneqq (\mathsf{ct}_i^k)_{i\in[m]}$.

   $O_{\mathcal{KG}}(f)$**:** 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 $\mathsf{sbskl}.\mathit{fsk} \coloneqq (\mathit{fsk}_i)_{i\in[m]}$.

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

$O_{\mathsf{Vrfy}}(f, \mathsf{cert} \coloneqq (\mathsf{cert}_i)_{i\in[m]})$**:** Given $(f, \mathsf{cert} \coloneqq (\mathsf{cert}_i)_{i\in[m]})$, 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 $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, i^*, (\mathbf{0}, K, 1))$ for every $i \in S_{\mathtt{secure},<i'}$ and $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, i^*, (s_i^*, \mathbf{0}, 0))$ for every $i \in [m]\setminus(S_{\mathtt{secure},<i'} \cup \{i'\})$. $\mathcal{B}$ sends $\mathsf{sbskl.ct} \coloneqq (\mathsf{ct}_i^*)_{i\in[m]}$ 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 4.2. $|\Pr[\mathsf{Hyb}_3 = 1] - \Pr[\mathsf{Hyb}_4 = 1]| = \mathsf{negl}(\lambda)$ *holds if* SetHSS *is a set homomorphic secret sharing.*

Proof of Proposition 4.2. We construct the following adversary $\mathcal{B}$ that attacks the selective indistinguishability-security of SetHSS.

1. Given $\mathsf{params} \coloneqq (p, \ell, (T_i)_{i\in[m]})$, $\mathcal{B}$ executes $\mathcal{A}$ and obtains $(1^q, n, x_0^*, x_1^*)$. $\mathcal{B}$ generates $i^k \leftarrow [N]$ for every $i \in [m]$ and $k \in [n]$ and $i^* \leftarrow [N]$ 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 $(i^k)_{i\in[m]}$ generated at the beginning. Otherwise, Generate $k_i \leftarrow [N]$ for every $i \in [m]$.
   - Generate $\mathsf{ct}_i^k \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, k_i, (s_i^k, \mathbf{0}, 0))$ for every $i \in [m]$.
   - Set $\mathsf{sbskl.ct}^k \coloneqq (\mathsf{ct}_i^k)_{i\in[m]}$.

   $O_{\mathcal{KG}}(f)$**:** Given f, $\mathcal{B}$ returns $\mathsf{sbskl.}\mathit{fsk}$ computed as follows.
   - Queries f to its function encode oracle and obtain $(f_i, y_i \coloneqq f_i(s_i^*))_{i\in[m]})$ 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 $\mathsf{sbskl.}\mathit{fsk} \coloneqq (\mathit{fsk}_i)_{i\in[m]}$.

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

$O_{\mathsf{Vrfy}}(f, \mathsf{cert} \coloneqq (\mathsf{cert}_i)_{i\in[m]})$: Given $(f, \mathsf{cert} \coloneqq (\mathsf{cert}_i)_{i\in[m]})$, 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 $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, *_i, (\mathbf{0}, K, 1))$ for every $i \in S_{\mathtt{secure}}$ and $\mathsf{ct}_i^* \leftarrow \mathsf{iEnc}(\mathsf{msk}_i, *_i, (s_i^*, \mathbf{0}, 0))$ for every $i \in S_{\mathtt{insecure}}$, and $\mathcal{B}$ sends $\mathsf{sbskl.ct} \coloneqq (\mathsf{ct}_i^*)_{i\in[m]}$ 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 4.1 (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 4.1. 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.

5 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 $\mathsf{skl.ct} := (\mathsf{ct}_k)_{k\in[\lambda]}$.

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

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

$\mathsf{SKL.Vrfy}(\mathsf{vk}, \mathsf{cert})$:
- 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 5.1. *If* SKFE-sbSKL *satisfies selective strong lessor security, then* SKFE-SKL *satisfies selective lessor security.*

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

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

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

Proof of Proposition 5.1. We construct the following adversary $\mathcal{B}$ that attacks selective lessor security of SKFE-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 $k \in [\lambda]\backslash\{k'\}$. $\mathcal{B}$ 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, SKFE-SKL satisfies selective lessor security. ■

By Theorems 3.2, 4.1 and 5.1 and the fact that all building blocks used to obtain these theorems can be based on OWFs, we obtain the following theorem.

Theorem 5.2. *If there exist OWFs, there exists selectively lessor secure SKFE-SKL for* P/poly *(in the sense of Definition 2.2).*

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.

References

Aar09. S. 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. 2009.

ABSV15. P. Ananth, Z. Brakerski, G. Segev, and V. Vaikuntanathan. From Selective to Adaptive Security in Functional Encryption. In *CRYPTO 2015, Part II*, pages 657–677. 2015.

AJL+19. P. Ananth, A. Jain, H. Lin, C. Matt, and A. Sahai. Indistinguishability Obfuscation Without Multilinear Maps: New Paradigms via Low Degree Weak Pseudorandomness and Security Amplification. In *CRYPTO 2019, Part III*, pages 284–332. 2019.

AL21. P. Ananth and R. L. La Placa. Secure Software Leasing. In *EUROCRYPT 2021, Part II*, pages 501–530. 2021.

ALL+21. S. Aaronson, J. Liu, Q. Liu, M. Zhandry, and R. Zhang. New Approaches for Quantum Copy-Protection. In *CRYPTO 2021, Part I*, pages 526–555, Virtual Event, 2021.

AMVY21. S. Agrawal, M. Maitra, N. S. Vempati, and S. Yamada. Functional Encryption for Turing Machines with Dynamic Bounded Collusion from LWE. In *CRYPTO 2021, Part IV*, pages 239–269, Virtual Event, 2021.

AV19. P. Ananth and V. Vaikuntanathan. Optimal Bounded-Collusion Secure Functional Encryption. In *TCC 2019, Part I*, pages 174–198. 2019.

BI20. A. Broadbent and R. Islam. Quantum Encryption with Certified Deletion. In *TCC 2020, Part III*, pages 92–122. 2020.

BJL+21. A. Broadbent, S. Jeffery, S. Lord, S. Podder, and A. Sundaram. Secure Software Leasing Without Assumptions. In *TCC 2021, Part I*, pages 90–120. 2021.

BNPW20. N. Bitansky, R. Nishimaki, A. Passelègue, and D. Wichs. From Cryptomania to Obfustopia Through Secret-Key Functional Encryption. *Journal of Cryptology*, 33(2):357–405, 2020.

BS18. Z. Brakerski and G. Segev. Function-Private Functional Encryption in the Private-Key Setting. *Journal of Cryptology*, 31(1):202–225, 2018.

BSW11. D. Boneh, A. Sahai, and B. Waters. Functional Encryption: Definitions and Challenges. In *TCC 2011*, pages 253–273. 2011.

CGO21. M. Ciampi, V. Goyal, and R. Ostrovsky. Threshold Garbled Circuits and Ad Hoc Secure Computation. In *EUROCRYPT 2021, Part III*, pages 64–93. 2021.

CLLZ21. A. Coladangelo, J. Liu, Q. Liu, and M. Zhandry. Hidden Cosets and Applications to Unclonable Cryptography. In *CRYPTO 2021, Part I*, pages 556–584, Virtual Event, 2021.

CMP20. A. Coladangelo, C. Majenz, and A. 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.

CS19. R. J. Connor and M. Schuchard. Blind Bernoulli Trials: A Noninteractive Protocol For Hidden-Weight Coin Flips. In *USENIX Security 2019*, pages 1483–1500. 2019.

CV21. E. Culf and T. Vidick. A monogamy-of-entanglement game for subspace coset states. *arXiv (CoRR)*, abs/2107.13324, 2021.

DIJ+13. A. De Caro, V. Iovino, A. Jain, A. O'Neill, O. Paneth, and G. Persiano. On the Achievability of Simulation-Based Security for Functional Encryption. In *CRYPTO 2013, Part II*, pages 519–535. 2013.

GGLW21. R. Garg, R. Goyal, G. Lu, and B. Waters. Dynamic Collusion Bounded Functional Encryption from Identity-Based Encryption. Cryptology ePrint Archive, Report 2021/847, 2021. https://eprint.iacr.org/2021/847.

GVW12. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional Encryption with Bounded Collusions via Multi-party Computation. In *CRYPTO 2012*, pages 162–179. 2012.

HMNY21. T. Hiroka, T. Morimae, R. Nishimaki, and T. Yamakawa. Quantum Encryption with Certified Deletion, Revisited: Public Key, Attribute-Based, and Classical Communication. In *ASIACRYPT 2021, Part I*, pages 606–636. 2021.

JKMS20. A. Jain, A. Korb, N. Manohar, and A. Sahai. Amplifying the Security of Functional Encryption, Unconditionally. In *CRYPTO 2020, Part I*, pages 717–746. 2020.

KLM+18. S. Kim, K. Lewi, A. Mandal, H. Montgomery, A. Roy, and D. J. Wu. Function-Hiding Inner Product Encryption Is Practical. In *SCN 18*, pages 544–562. 2018.

KNT21. F. Kitagawa, R. Nishimaki, and K. Tanaka. Simple and Generic Constructions of Succinct Functional Encryption. *Journal of Cryptology*, 34(3):25, 2021.

KNY21. F. Kitagawa, R. Nishimaki, and T. Yamakawa. Secure Software Leasing from Standard Assumptions. In *TCC 2021, Part I*, pages 31–61. 2021.

MW16. P. Mukherjee and D. Wichs. Two Round Multiparty Computation via Multi-key FHE. In *EUROCRYPT 2016, Part II*, pages 735–763. 2016.

NP01. M. Naor and B. Pinkas. Efficient Trace and Revoke Schemes. In *FC 2000*, pages 1–20. 2001.

NWZ16. R. Nishimaki, D. Wichs, and M. Zhandry. Anonymous Traitor Tracing: How to Embed Arbitrary Information in a Key. In *EUROCRYPT 2016, Part II*, pages 388–419. 2016.

RPB+19. T. Ryffel, D. Pointcheval, F. R. Bach, E. Dufour-Sans, and R. Gay. Partially Encrypted Deep Learning using Functional Encryption. In *NeurIPS 2019*, pages 4519–4530, 2019.

SS10. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS 2010*, pages 463–472. 2010.

SSW12. A. Sahai, H. Seyalioglu, and B. Waters. Dynamic Credentials and Ciphertext Delegation for Attribute-Based Encryption. In *CRYPTO 2012*, pages 199–217. 2012.

SW05. A. Sahai and B. R. Waters. Fuzzy Identity-Based Encryption. In *EUROCRYPT 2005*, pages 457–473. 2005.

SW21. A. Sahai and B. Waters. How to Use Indistinguishability Obfuscation: Deniable Encryption, and More. *SIAM J. Comput.*, 50(3):857–908, 2021.