

THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Consistency for Functional Encryption

Citation for published version:
Badertscher, C, Kiayias, A, Kohlweiss, M & Waldner, H 2021, Consistency for Functional Encryption. in *2021 IEEE 34th Computer Security Foundations Symposium (CSF).* IEEE, pp. 1-16, 34th IEEE Computer Security Foundations Symposium, 21/06/21. https://doi.org/10.1109/CSF51468.2021.00045

Digital Object Identifier (DOI):
10.1109/CSF51468.2021.00045

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
2021 IEEE 34th Computer Security Foundations Symposium (CSF)

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.



Download date: 28. Mar. 2024

Consistency for Functional Encryption

Christian Badertscher
IOHK
christian.badertscher@iohk.io

Aggelos Kiayias
University of Edinburgh and IOHK
aggelos.kiayias@ed.ac.uk

Markulf Kohlweiss
University of Edinburgh and IOHK
markulf.kohlweiss@ed.ac.uk

Hendrik Waldner
University of Edinburgh
hendrik.waldner@ed.ac.uk

***Abstract*—In functional encryption (FE) a sender, Alice, encrypts plaintexts for which a receiver, Bob, can obtain functional evaluations, while Charlie is responsible for initializing the encryption keys and issuing the decryption keys. Standard notions of security for FE deal with a malicious Bob and guarantee the confidentiality of Alice's messages despite the leakage that occurs due to the functional keys that are revealed to the adversary via various forms of indistinguishability experiments that correspond to IND-CPA, IND-CCA and simulation-based security.**

In this work we provide a complete and systematic investigation of *Consistency*, a natural security property for FE, that deals with attacks that can be mounted by Alice, Charlie or a collusion of the two against Bob. We develop three main types of consistency notions according to which set of parties is corrupted and investigate their relation to the standard security properties of FE. To validate our different consistency types, we extend the universally composable framework for FE by Matt and Maurer (CSF 2015) and we show that our consistency notions naturally complement FE security by proving how they imply (and are implied by) UC security depending on which set of parties is corrupted; in this way we demonstrate a complete characterization of consistency for FE. Finally, we provide explicit constructions that achieve consistency efficiently either directly via a construction based on MDDH for specific function classes of inner products over a modulo group or generically for all the consistency types via compilers using standard cryptographic tools.

I. Introduction

Functional encryption (FE) [22], [56] has emerged as an important and general purpose cryptographic primitive, extending and generalizing earlier more specialized encryption concepts that include Identity-Based Encryption [21], Attribute-Based Encryption [44], [62] and Predicate Encryption [49]. Similar to these earlier primitives, in FE, there exists a setup algorithm that produces a master public-key mpk and a master secret-key msk, and a key-generation algorithm that receives as input msk and a function f and produces a function-specific secret-key sk_f. Subsequently, using sk_f along with the decryption algorithm, the computation of the value $f(x)$ is facilitated given any ciphertext that encrypts x. The potential applications of FE are numerous and include any setting where there exist designated entities that are entitled to functional views of encrypted information that is described in the form of a function f for which an associated functional key sk_f is produced by the key-generation procedure.

In order to define correctness and security of FE it is helpful to identify three distinct entities associated with the algorithms that comprise any FE scheme. Alice is the sender, wishing to transmit data x, Bob is a recipient wishing to receive $f(x)$ for some function $f(\cdot)$ and Charlie is an authority that issues the (master) keys. Typically we think there are multiple Alice and Bob parties for any given setup instance created by Charlie. As one of these Bob parties can be corrupted, this also captures security against an eavesdropper that only observes the network. Correctness mandates the natural requirement that Bob receives the value $f(x)$ for properly encrypted ciphertexts prepared by Alice that contain x. Security on the other hand is typically captured as a game with an adversary who attempts to distinguish between two possible plaintexts x_0, x_1 for which it holds that $f_i(x_0) = f_i(x_1)$ for all functions f_i whose key is possessed by the adversary. A stronger notion of security puts forth a simulation-based formulation and asks that ciphertexts can be simulated in an indistinguishable way. Cf. [5], [6], [14], [16], [22], [32], [43], [47], [52], [56]. The adversary controls multiple different Bob sessions and typically interferes with the honest Alice only in the sense of chosen plaintext attacks, however chosen ciphertext attacks have also been considered [18] (in which case the adversary may e.g., manipulate Alice's ciphertext and submit it to Bob's decryption oracle). Our work further builds on the composable formalization of FE security by Matt and Maurer [52].[1]

1) Consistency problems in real-world applications of FE: A crucial problem for any cryptographic primitive is to identify the exact set of correctness and security properties that are necessary and sufficient for deploying the primitive

[1]Note that while the composable analysis in [52] is formally conducted in the Constructive Cryptography (CC) framework [53], the composable guarantees captured via an ideal system in the CC framework are very close to a UC formulation (via an ideal functionality). Our work makes this translation on the fly.

within an intended real world system. To see that there is a fundamental property of FE that is missing, it is helpful to recall the most well known applications of FE and showcase the problems that emerge when *consistent* behavior of an FE scheme is not guaranteed.

a) Processing Encrypted Data: In the original paper [22] the following motivating example for FE is presented: Alice encrypts a photograph x and uploads it to her cloud service provider, Charlie, while Bob, a law enforcement agent, wishes to check whether any photographs in the cloud match a specific face. Using FE, Bob can achieve his objective, taking advantage of a functional key which detects the matching encrypted photographs without revealing any other information. Given the above setting, it is in everyone's understanding that if a photograph matches a specific face being searched, the law enforcement agent will be able to detect it. Nevertheless, neither standard notions of security nor correctness of FE can rule out the possibility that a malicious Alice creates a ciphertext that will be misclassified by Bob, specifically a ciphertext that in fact decrypts to a photograph of the person being searched, and for which the employed face recognition algorithm f actually works, but which is not detected as such by Bob when implementing the task using the functional key sk_f. As an extreme case, the failure could be selective: Alice can profit from all other services by the cloud provider (for different functions f') (such recommendation systems or collaborative filtering), and just exclude that her photo is detected by the law-enforcement agent. Looking ahead, the property that rules out this case called *input consistency*. The above setting can be more adverse in that a coalition of Alice and Charlie can together fool the law enforcement agent, e.g., by creating subverted public keys. The property that ensures the correct functioning of the classification task is called *strong input consistency*. Finally, even if the input provider Alice is not malicious, a problem with subverted system parameters can occur: the cloud service provider could generate a key for the law enforcement agent that purposefully misclassifies some pictures that contain for example the specific face that Bob is looking for and in effect allow Charlie to protect certain people such as Alice from prosecution. Also the other direction is possible: Charlie could also wrongfully frame her by generating a functional key that wrongly detects a specific face in Alice's ciphertext. (Both of these attacks could even be eased in case it is possible to trick Alice and Bob into using different master-public keys.) Excluding this case requires the FE scheme to be *setup consistent*. Surprisingly, as we will see later, setup consistency is not implied by strong input consistency. Clearly, similar "misclassification" inconsistency issue applies to any setting where FE is used to classify ciphertexts in-transit or in-situ (e.g., for virus-detection, routing etc.).

b) Attribute Based Encryption: In an attribute-based encryption (ABE) scheme [44], which is a special case of FE, Alice encrypts a message together with a set of attributes γ. Subsequently, Bob, who possesses a key corresponding to an access structure $\mathbb{A}$ will be able to decrypt the message as long as $\gamma \in \mathbb{A}$. Consider now also another party, say Bob junior, possessing a key for the access structure $\mathbb{A}' \subsetneq \mathbb{A}$. Given the above setting, it is in everyone's understanding that whatever messages Bob junior is able to see, Bob should see as well. Nevertheless, neither standard notions of security nor correctness of FE can rule out the possibility that a malicious Alice crafts a ciphertext that Bob junior will be able to decrypt but Bob would not. In the context of access control systems, this would imply that ciphertexts that appear valid for some parties are not role-respecting and therefore not compatible with the policy. As above, the same inconsistent behavior could occur with corrupted setup parameters and/or functional keys even if Alice is honest.

2) Consistency as a fundamental property for FE: What do the above problems tell us? Similar to advanced properties of ordinary PKE (such as e.g., robustness [1]), advanced properties for FE are needed when using the primitive in a real world setting because such properties are implied by the way the primitive is understood in the real-world. Moreover, the level at which they should be defined is at the level of the basic definition and syntax of FE. We call the enhanced property the above issues point to *consistency*; it addresses, at minimum, the adversarial setting where a malicious Alice produces a specially crafted ciphertext that causes an honest Bob to misclassify it, or, perhaps even a malicious Charlie who tampers with the setup to cause further types of misclassification. Interestingly and somewhat surprisingly, such a consistency property has not been considered in the strict context of FE so far and enhanced FE schemes, departing from the standard syntax such as [11], do only consider certain consistency aspects (see below). We show that, as with the confidentiality of FE, the consistency of FE has several flavors, some of which are very efficient to ensure, while others require more sophisticated techniques.

A. Contributions of this Work

We roll out consistency as a fundamental property of FE scheme from first principles. We provide a number of constructions for various consistency and security notions either directly for specific function classes or generically via compilers that upgrade existing FE constructions to be consistent. To formally cross-check our new notion, we show that the defined properties are necessary and sufficient in realizing the UC characterization of an "ideally" secure and consistent FE-scheme abstraction derived from [52]. The modelling of all security properties as an ideal functionality assures that no important details were omitted and that our game-based definitions interoperate correctly. In more details we make the following contributions.

1) Formal definition of consistency: We identify three main types of consistency, each type naturally corre-

sponding to a particular set of corrupted parties. The formalization is given in Section III.

- *Input consistency* considers a malicious Alice who computes a ciphertext ct and candidate functions f_i. The ciphertext ct is decrypted under sk_{f_i} to obtain the values y_i. The adversary wins if there is no single x that can explain ct in the sense that $f_i(x) = y_i$.
- *Strong input consistency* couples the above goal with additional adversarial power. It considers the setting where both Alice and Charlie are corrupted. Therefore, subverted parameters can assist the adversary in breaking the scheme.
- *Setup consistency* is the consistency notion that deals with a malicious Charlie. In this setting the adversary issues two plaintexts x_1, x_2 as well as a secret-key and a function f. The plaintexts are honestly encrypted and subsequently their decryptions y_1, y_2 are evaluated. The adversary wins the game if exactly one of the decryptions fails or $y_i \neq f(x_i)$ for some i. While at first sight it seems that setup consistency is implied by strong input consistency, this is not the case. This is discussed further in Section III-C.

We highlight that consistency in the above sense complements security, as in the latter Alice and Charlie are honest and Bob is malicious. To show that our definitions do formally capture what they are intended for, we put forth in Section IV a complete treatment of consistent FE in the universal composition (UC) setting [25]. Specifically, we prove that input consistency/setup consistency/strong input consistency is sufficient and necessary for UC security when Alice/Charlie/Alice+Charlie are corrupted respectively. This pairs and complements the result of [52] which implies that CFE security is sufficient (and necessary) for UC security in the case Bob is corrupted. We thus position consistency as an important novel property of FE.

2) Systematic study of consistency vis-à-vis existing security properties: We carefully analyze the relations in-between the consistency notions and between consistency and security. We confirm our intuition that all notions define separate levels of consistency, the only exception being that strong input consistency implies input consistency. With respect to security, namely IND-CPA, IND-CCA and CFE, the composable security notion of [52], we show that strong input consistency does not imply IND-CPA security and therefore also not IND-CCA or CFE, since both of these notions are known to imply IND-CPA. Furthermore, we show that IND-CPA together with strong input consistency does not imply any of the other stronger security notions such as IND-CCA or CFE. Finally, IND-CCA and CFE individually do not imply input consistency. We refer to Figure 5 for a relation diagram. Thus, it follows that consistency is independent from existing notions of FE security. The proofs are given in Section C.

3) Realizing FE with consistency: We first describe, in Section V, concrete input-consistent constructions for an inner-product type of FE under the Matrix DDH assumption for two different functional classes. The first construction covers the modified inner-product functionality class over a modulo group and the second construction covers the function class of exponentiated inner-products over a modulo group. Both of these constructions are adapted from the construction of [7].

Interestingly, we observe and prove that previous efficient constructions for the function class of inner-product over the *integers* fail to provide input consistency. We present explicit attacks, that exploit one core step in DDH based inner product functional encryption schemes which is the discrete logarithm computation at the end of the decryption algorithm: it is possible to generate a malicious ciphertext which, on two different honestly generated functional keys, will behave inconsistently in that one decryption yields an error symbol for one key, and the correct result for the other functional key. In other words, the two decryptions are not explainable by an underlying value and hence an inconsistent behavior occurs.

Subsequently, in Section VI we present compilers that achieve consistency in a black-box manner from any FE scheme. Our work is inspired by that of [11] which dismisses several compiler constructions as too weak when both Alice and Charlie are corrupt. However, there are scenarios where this is not the case and for which no compilers have been analyzed. We give compilers that achieve input consistency and setup consistency with improved efficiency compared to the strong input consistency case. Additionally, we not only prove security preservation for CPA and CFE, but also present how to lift the security from CPA to CCA for each compiler using the twin encryption technique [55]. Finally, for strong input consistency, we show that the compiler in [11], which achieves *verifiable FE* can be used to achieve strong input consistency. We actually show the more general result that any VFE scheme can be turned into a strong input consistent FE scheme. The reverse, however is not true and we elaborate on this in Section VI as well as below in Section I-B.

B. Comparison with Related Work

Relation to VFE: Prior to our work, there is only one previous, very insightful work [11] which identified some of the above deficiencies. In more detail, it puts forth a cryptographic primitive which is substantially stronger and syntactically different than FE, called verifiable functional encryption (VFE) (and the compiler presented in [11] has recently been instantiated using pairing-based NIWIs and a perfectly correct functional encryption scheme for predicates over inner-products [63]). VFE extends the normal FE syntax by two additional predicates to check validity of keys and ciphertexts, respectively. As already mentioned above, VFE implies strong input-consistent FE. Interestingly, the reverse is not necessarily true as VFE requires the public verifiability of ciphertext and well-formedness of functional key whereas for strong input consistency a private-key based test, for instance, is sufficent.

Furthermore, VFE (as well as strong input-consistent FE) does not imply setup consistency, since it merely guarantees that an encryption c of plaintext x would consistently decrypt to something but *not necessarily* to functions of x that an honest sender has encrypted (i.e., the setting where genuinely generated ciphertexts may be mangled due to a subverted setup). This, however, seems rather crucial, as additional guarantees for the setting where only Charlie is dishonest are desirable (see Section III). As briefly mentioned above, we obtain efficiency improvements for the specific cases of input and setup consistency compared to the VFE-compiler of [11]. Our input-consistency compiler only relies on NIZKs (not NIWIs) and only requires a single instance of a functional encryption scheme instead of four instances that the VFE compiler employs. For the setup-consistency compiler, we also need NIWIs, but show that only three instances of the FE scheme are sufficient. As for security, we directly aim at full CPA/CCA security instead of selective security [11].

Relation to robustness notions: As a first approximation, input-consistency for FE can be thought of as a natural well-formedness property of FE ciphertexts which is the main reason why it is of relevance to the cryptographic investigation of the FE primitive, in a similar way to other types of consistency properties for regular public-key encryption, for instance, plaintext-awareness [31] and non-malleability [33] (which are independent of weaker notions of security such as IND-CPA while related to stronger formulations of security such as IND-CCA. Furthermore, a consistency-like property more closely related to FE is robustness of identity-based encryption (IBE) [1], [35]. In the strong robustness attack of Abdalla et al. [1] an adversary outputs two identities $\mathsf{id}_1 \neq \mathsf{id}_2$ and a ciphertext ct. The game derives decryption keys for both identities and the adversary wins if the decryption of ct is non-$\perp$ under both keys. IBE can be viewed as a special case of FE: encrypt the pair identity and message and let the user with identity id possess the key for the function "$f_{\mathsf{id}}(\mathsf{id}', m) = m$, if $\mathsf{id} = \mathsf{id}'$ else $\perp$." It is immediate that a robustness attack is a consistency attack against the above FE scheme, since it cannot be that two distinct identities $\mathsf{id}_1, \mathsf{id}_2$ equal the same id'. By this reduction, we see that our notion of consistency for general FE can be instantiated to yield such related robustness notions for special cases such as IBE directly. Besides strong robustness, the authors also introduce the notion of weak robustness, in which the adversary outputs a message and two identities and the challenger encrypts the message under the first identity and tries to decrypt it using the second identity. The authors show that weak robustness is implied by strong robustness, and since strong robustness mirrors our notion of input consistency, it follows immediately that weak robustness is also implied by input consistency notion when instantiated for IBE as above.

A related kind of work is the task of generalizing the traditional notions of robustness, which roughly captures that decryption with a secret key that is generated in some system A *must indicate a failure* when presented with a ciphertext that was generated using (different) parameters of some system B. These notions have been extended (also to FE) [35], [39], but none of them looks at the harder problem covered by our work, namely to ensure that decrypted values (within one system), make sense relatively to each other, especially in FE.

Relation to distributed setup-generation: Setup consistency focuses on the important question of setup subversion resistance. A different approach in this realm is decentralized setup generation [50], Our setup consistency notion can complement such an approach by giving guarantees even in case all the parties (or a number of parties exceeding a critical threshold) involved in the MPC are corrupted.

II. Preliminaries

1) Notation: The security parameter is denoted by $\lambda \in \mathbb{N}$ and its unary encoding by 1^λ. We call a randomized algorithm $\mathcal{A}$ *probabilistic polynomial time* (PPT), if there exists a polynomial $p(\cdot)$ such that for every input x the running time of $\mathcal{A}(x)$ is bounded by $p(|x|)$. A function $\mathrm{negl} : \mathbb{N} \to \mathbb{R}^+$ is called *negligible* if for every positive polynomial $p(\lambda)$ a $\lambda_0 \in \mathbb{N}$ exists, such that for all $\lambda > \lambda_0 : \epsilon(\lambda) < 1/p(\lambda)$. For a function f with domain $\mathcal{X}$ and range $\mathcal{Y}$, we write $f^{-1}(y) := \{x \in \mathcal{X} \mid f(x) = y\}$ where $y \in \mathcal{Y}$.

The set $\{1, \ldots, n\}$ is denoted as $[n]$ for $n \in \mathbb{N}$. For the equality check of two elements, we use "=". The assign operator is denoted with ":=", whereas randomized assignment is denoted with $a \leftarrow A$, with a randomized algorithm A and where the randomness is not explicit. If the randomness is explicit we write $a := A(x; r)$ where x is the input and r is the randomness. For algorithms $\mathcal{A}$ and $\mathcal{B}$, we write $\mathcal{A}^{\mathcal{B}(\cdot)}(x)$ to denote that $\mathcal{A}$ gets x as an input and has oracle access to $\mathcal{B}$, that is, the response for an oracle query q is $\mathcal{B}(q)$. We use $\mathcal{A}(\cdot)[[s]]$ to denote that $\mathcal{A}$ gets an additional input s which it can update. In more detail, $y \leftarrow \mathcal{A}(x)[[s]]$ corresponds to the algorithm that invokes $(y, s) \leftarrow \mathcal{A}(x, s)$ and returns y and updates s.

We write $\boldsymbol{e}_i^\ell$ for the unit vector of length ℓ that is 1 at position i and 0 everywhere else. We omit the length when it is clear from the context.

For the generation of prime-order groups, let GGen be a PPT algorithm that on input 1^λ returns a description $\mathcal{G} = (\mathbb{G}, p, g)$ of a cyclic group $\mathbb{G}$ of order p for a λ-bit prime p, whose generator is g. We use the implicit representation $[x]_g$ for group elements of the form g^x with a generator $g \in \mathbb{G}$. This notation is also used in the case of matrices. In more detail, for a matrix $\mathbf{A} = (a_{i,j}) \in \mathbb{Z}_p^{n \times m}$, we define $[\mathbf{A}]_g$ as the implicit representation of $\mathbf{A}$ in $\mathbb{G}$:

$$[\mathbf{A}]_g := \begin{pmatrix} g^{a_{1,1}} & \ldots & g^{a_{1,m}} \\ & & \\ g^{a_{1n,1}} & \ldots & g^{a_{n,m}} \end{pmatrix}$$

For games (i.e., random experiments) G that model an interaction between a challenger and an adversary, we

refer to the winning probability of an adversary $\mathcal{A}$ by $\mathsf{Win}^{\mathsf{G}}_{\mathcal{A}}(\lambda) \coloneqq \Pr[\mathsf{G}(\lambda, \mathcal{A}) = 1]$, where the probability is taken over the random coins of G and $\mathcal{A}$.

2) Functional Encryption: We now introduce the relevant notation for functional encryption.

Definition 1. *We denote by $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ a family of sets $\mathcal{F}_\lambda$ of functions $f : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$. We call $\mathcal{F}_\lambda$ a functionality class where all $f \in \mathcal{F}_\lambda$ have the same domain and the same range. We omit λ when it is clear from the context.*

For notational convenience, we further define an extension for functions $f \in \mathcal{F}$ in order to develop a formal language that simplifies expressing decryption consistency later in this work. We introduce two additional error symbols $\perp$ (invalid ciphertext), $\diamond$ (invalid key) and formally include them in the domain or range of the functions as defined below. We note that both symbols do not have any influence on the behavior of the function f. Rather, we require that the symbol $\perp$ maps to $\perp$ and that symbol $\diamond$ has no preimage:

Definition 2 (Function Extension)**.** *Let $f : \mathcal{X} \to \mathcal{Y}$ be a function of the functionality class $\mathcal{F}$, we define a function $\tilde{f} : (\mathcal{X} \cup \{\perp\}) \to (\mathcal{Y} \cup \{\perp, \diamond\})$, with $\perp, \diamond \notin \mathcal{X}, \mathcal{Y}$. The function $\tilde{f}$ has the following behavior:*

$$\tilde{f}(x) = \begin{cases} f(x) & \text{if } x \in \mathcal{X} \\ \perp & \text{if } x = \perp \end{cases}$$

and

$$\tilde{f}^{-1}(y) = \begin{cases} f^{-1}(y) & \text{if } y \in \mathcal{Y} \\ \{\perp\} & \text{if } y = \perp \\ \emptyset & \text{if } y = \diamond \end{cases}.$$

For a (standard) functionality class $\mathcal{F}$, the induced extended class is the set of function extensions of all $f \in \mathcal{F}$. When clear from the context, we do not introduce a new symbol for the extended class.

A functional encryption scheme is defined in the following way, where we follow the syntax of [11].

Definition 3 (Functional Encryption)**.** *Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of sets $\mathcal{F}_\lambda$ of functions $f : \mathcal{X}_\lambda \to \mathcal{Y}_\lambda$, where $\mathcal{X}_\lambda$ and $\mathcal{Y}_\lambda$ are finite sets that represent domain and range, respectively, and let $f_0 \in \mathcal{F}_\lambda$ be a distinguished leakage function[2]. A functional encryption scheme (FE) for the functionality class $\mathcal{F}_\lambda$ is a tuple of four algorithms $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$:*

$\mathsf{Setup}(1^\lambda)$**:** *Takes as input a unary representation of the security parameter λ and outputs the master public key mpk and the master secret key msk.*

$\mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$**:** *Takes as input the master public mpk, the master secret key msk and a function $f \in \mathcal{F}_\lambda$, and outputs a functional key sk_f. The key for the leakage function f_0 is the empty string denoted by ε.*

$\mathsf{Enc}(\mathsf{mpk}, x)$**:** *Takes as input the master public key mpk and a string $x \in \mathcal{X}_\lambda$, and outputs a ciphertext ct or err (to denote an encryption error).*

$\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}_f, \mathsf{ct})$**:** *Takes as input a functional key sk_f and a ciphertext ct and outputs a function value $y \in \mathcal{Y}_\lambda$ or one of the special symbols of the function extension: $\perp$ indicates an invalid ciphertext and $\diamond$ invalid keys.*

A scheme FE is correct, if (for all $\lambda \in \mathbb{N}$), for all pairs $(\mathsf{mpk}, \mathsf{msk})$ in the support of $\mathsf{Setup}(1^\lambda)$ all functions $f \in \mathcal{F}_\lambda$ and input values $x \in \mathcal{X}_\lambda$, it holds that

$$\Pr[\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f), \mathsf{Enc}(\mathsf{mpk}, x)) = f(x)] = 1.$$

For notational simplicity, we omit certain input values when they are not required by a concrete scheme (such as the additional mpk or f when decrypting).

The security of functional encryption is formally captured by the CPA [22], CCA2 [18], as well as the CFE [52] (composable) security notions, that formalize, roughly speaking that an attacker does not learn anything beyond what he can anyway decrypt given the functional keys (and in the case of CCA, additional decryptions) he requested.

We review all the security notions in Section A-A. The main body of this work can be understood without them.

3) Standard Tools and Assumptions: Now, we recap the definition of a matrix distribution and the Matrix-Diffie-Hellman assumption as introduced in [34]. We begin with the definition for a matrix distribution.

Definition 4 (Matrix Distribution)**.** *Let $\ell, k \in \mathbb{N}$ with $\ell > k$. We call $\mathcal{D}_{\ell,k}$ a matrix distribution if it outputs matrices in $\mathbb{Z}_p^{\ell\times k}$ of full rank k in polynomial time. We define $\mathcal{D}_k \coloneqq \mathcal{D}_{k+1,k}$.*

We assume, wlog, that the first k rows of $\mathbf{A} \leftarrow \mathcal{D}_k$ form an invertible matrix. The $\mathcal{D}_k$-Matrix Diffie-Hellman problem is to distinguish the two distributions $([\mathbf{A}], [\mathbf{A}\boldsymbol{w}])$ and $([\mathbf{A}], [\boldsymbol{u}])$ where $\mathbf{A} \leftarrow \mathcal{D}_k, \boldsymbol{w} \leftarrow \mathbb{Z}_p^k$ and $\boldsymbol{u} \leftarrow \mathbb{Z}_p^{k+1}$.

Now, we state the $\mathcal{D}_k$-Matrix Diffie-Hellman Assumption ($\mathcal{D}_k$-MDDH).

Definition 5 ($\mathcal{D}_k$-Matrix Diffie-Hellman Assumption ($\mathcal{D}_k$-MDDH))**.** *Let $\mathcal{D}_k$ be a matrix distribution. The $\mathcal{D}_k$-Matrix Diffie-Hellman ($\mathcal{D}_k$-MDDH) assumption holds relative to GGen if for all PPT adversaries $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathcal{D}_k\text{-MDDH}}_{\mathsf{GGen},\mathcal{A}}(\lambda) := |\Pr[\mathcal{A}(\mathcal{G}, [\mathbf{A}], [\mathbf{A}\boldsymbol{w}]) = 1] - \Pr[\mathcal{A}(\mathcal{G}, [\mathbf{A}], [\boldsymbol{u}]) = 1]| \leq \mathrm{negl}(\lambda) \ ,$$

where the probability is taken over $\mathcal{G} = (\mathbb{G}, p, g), \mathbf{A} \leftarrow \mathcal{D}_k, \boldsymbol{w} \leftarrow \mathbb{Z}_p^k, \boldsymbol{u} \leftarrow \mathbb{Z}_p^{k+1}$ and the coin tosses of adversary $\mathcal{A}$.

Finally, we recall non-interactive witness indistinguishable (NIWI) proofs [13], [20], [46].

[2]The leakage function is a modelling technique adopted from [22] that can, e.g., reveal information about the plaintext length.

$\mathrm{WI}_{\beta}^{\mathsf{NIWI}}(1^{\lambda}, \mathcal{A})$ (for relation R)

$(x, w_1, w_2, \mathsf{st}) \leftarrow \mathcal{A}_1(1^{\lambda})$

$\pi \leftarrow \mathsf{NIWI.Prove}(1^{\lambda}, x, w_{\beta})$

$\alpha \leftarrow \mathcal{A}_2(\pi, \mathsf{st})$

Output: $\alpha \wedge (x, w_1) \in R \wedge (x, w_2) \in R$

Fig. 1: Witness-indistinguishability of a NIWI proof system. The output condition enforces the use of valid instance witness pairs.

Definition 6 (Non-Interactive Witness-Indistinguishable Proofs)**.** *Let R be an NP Relation and consider the language $L = \{x \mid \exists w \text{ with } (x, w) \in R\}$ (where x is called a statement or instance). A non-interactive witness-indistinguishable proof (NIWI) for the relation R is a tuple of PPT algorithms* $\mathsf{NIWI} = (\mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$*:*

$\mathsf{NIWI.Prove}(1^{\lambda}, x, w)$**:** *Takes as input the unary representation of the security parameter λ, a statement x and a witness w, and outputs a proof π.*

$\mathsf{NIWI.Verify}(1^{\lambda}, x, \pi)$**:** *Takes as input the unary representation of the security parameter λ, a statement x and a proof π, and outputs* 0 *or* 1*.*

A system NIWI *is complete, if for all statement-witness pairs in the relation $(x, w) \in R$, it holds that*

$$\Pr[\mathsf{NIWI.Verify}(1^{\lambda}, x, \mathsf{NIWI.Prove}(1^{\lambda}, x, w)) = 1] = 1.$$

A NIWI proof system fulfills additional properties besides completeness, namely soundness and witness-indistinguishability.

Definition 7 (Soundness)**.** *Let* $\mathsf{NIWI} = (\mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$ *be a NIWI proof system for a relation R and the corresponding language L. We define the advantage of an adversary $\mathcal{A}$ as the following probability:*

$$\mathsf{Adv}^{\mathrm{Sound}}_{\mathsf{NIWI},\mathcal{A}}(\lambda) := \Pr[(x, \pi) \leftarrow \mathcal{A}(1^{\lambda}) : \mathsf{NIWI.Verify}(1^{\lambda}, x, \pi) = 1 \wedge x \notin L].$$

A NIWI proof system NIWI *is called perfectly sound if $\mathsf{Adv}^{\mathrm{Sound}}_{\mathsf{NIWI},\mathcal{A}}(\lambda) = 0$ for all algorithms $\mathcal{A}$, and computationally sound, if $\mathsf{Adv}^{\mathrm{Sound}}_{\mathsf{NIWI},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ for all PPT algorithms $\mathcal{A}$.*

Definition 8 (Witness-Indistinguishability)**.** *Let* $\mathsf{NIWI} = (\mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$ *be a NIWI proof system for a relation R and the corresponding language L. For $\beta \in \{0, 1\}$, we define the experiment $\mathrm{WI}_{\beta}^{\mathsf{NIWI}}(1^{\lambda}, \mathcal{A})$ in Fig. 1. The associated advantage of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is defined as*

$$\mathsf{Adv}^{\mathrm{WI}}_{\mathsf{NIWI},\mathcal{A},\mathcal{S}}(\lambda) := |\Pr[\mathrm{WI}_0^{\mathsf{NIWI}}(1^{\lambda}, \mathcal{A}) = 1] - \Pr[\mathrm{WI}_1^{\mathsf{NIWI}}(1^{\lambda}, \mathcal{A}) = 1]|.$$

A NIWI proof system is called witness-indistinguishable, if $\mathsf{Adv}^{\mathrm{WI}}_{\mathsf{NIWI},\mathcal{A}}(\lambda) = 0$ for all algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, and computationally witness-indistinguishable, if $\mathsf{Adv}^{\mathrm{WI}}_{\mathsf{NIWI},\mathcal{A}}(\lambda) \leq \mathrm{negl}(\lambda)$ for all PPT algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$.

Regarding feasibility, the NIWI construction in [46] relies on the decisional linear (DLIN) assumption and provides perfectly sound non-interactive witness indistinguishability. In [13], the authors rely on a complexity theoretic assumption and also present (less efficient) perfectly sound proofs. For the construction in [20], the authors rely on one-way permutations and indistinguishability obfuscation.

III. Consistency for Functional Encryption Schemes

Recall that there are three distinct tasks in functional encryption: parameter/key generation, encryption and decryption. Following [52], they correspond to three entities in a system: the input provider, the setup/key manager, and the decryptor. Consistency must be seen as a guarantee that an honest decryptor can rely on even in the presence of other malicious entities. In contrast, confidentiality (in the sense of CPA/CCA or CFE) is a guarantee that an honest input provider relies on against a potentially dishonest decryptor (in the presence of honestly generated setup and keys). We summarize these combinations in Table I. We remark that aside from the informal justification that the games represent what we intend to capture, we cross-check the games against a constructive and composable model in Section IV. This shows that our consistency notions realize the intended *idealized UC-functionality* for FE.

	Entities		
Notions	Input Provider	Setup & Key Generator	Decryptor
Correctness	Honest	Honest	Honest
in-CONS	Corrupted	Honest	Honest
set-CONS	Honest	Corrupted	Honest
st-in-CONS	Corrupted	Corrupted	Honest
Confidentiality	Honest	Honest	Corrupted

TABLE I: The different consistency notions and the corrupted entities

In the following sections, we introduce three different consistency notions, each corresponding to a different corruption set of untrusted entities: input consistency (in-CONS), strong input consistency (st-in-CONS), and setup consistency (set-CONS).

In the rest of this section, whenever we refer to a function f, or a functionality class $\mathcal{F}$, we implicitly mean the induced function extension as defined in Definition 2.

A. Consistency with a dishonest Input Provider

An input consistency attack entails the malicious generation of a ciphertext ct, and the honest generation of several non-trivial functional keys $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_n}$ that interpret the ciphertext ct in an inconsistent way. We call a ciphertext

consistent, if there exists a plaintext x that can explain the decryption of the ciphertext ct under the different functional keys $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_n}$, and inconsistent otherwise.

We formalize input-consistency as an experiment.

$\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A})$

$(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda)$

$\mathsf{ct} \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{mpk},\mathsf{msk},\cdot)}(1^\lambda, \mathsf{mpk})$

Let $F := \{(\mathsf{sk}_{f_i}, f_i)\}_{i\in[n]}$ be the set of key-function pairs obtained by $\mathcal{A}$.

If $n < 1$ then output 0

$y_i := \mathsf{Dec}(\mathsf{mpk}, f_i, \mathsf{sk}_{f_i}, \mathsf{ct}),$ for all $i \in [n]$

If $\bigcap_{i\in[n]} f_i^{-1}(y_i) = \emptyset$

  Output 1

Output 0

Fig. 2: Input Consistency Experiment.

Definition 9 (Input Consistency)**.** *The functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *satisfies input consistency (or* in-CONS *for short), if for any polynomial-time adversary* $\mathcal{A}$ *interacting with experiment* $\text{in-CONS}^{\mathsf{FE}}$ *in Fig. 2, there exists a negligible function* negl *such that:*

$$\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A})] \leq \text{negl}(\lambda) \ .$$

a) Discussion: The game reflects (in-)consistency: After the adversary $\mathcal{A}$ asked the key generation oracle $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ for functional keys $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_n}$ for functions $\{f_1, \ldots, f_n\}$, it outputs a ciphertext ct trying to win the game. The challenger checks if there exist plaintext messages that explains the functional decryption behavior of ct under these keys. Formally, it computes the intersection $\bigcap_{i\in[n]} f_i^{-1}(\mathsf{Dec}(\mathsf{mpk}, f_i, \mathsf{sk}_{f_i}, \mathsf{ct}))$. If it is empty, there is no explanation for the decryption behavior of ct. This means that the adversary has caused an inconsistency and wins the game.

Note that in order for our experiment to be well-defined, we just need the element-of operator $x \in f^{-1}(y)$ be computable (where $\mathcal{X}_\lambda$, $\mathcal{Y}_\lambda$, and $f^{-1}(\cdot)$ are by definition finite sets in an experiment parameterized by λ). In terms of efficiency, it is clear that the entire consistency check might not always be *efficiently* computable, for example when the f_i's are one-way functions. Whether a restriction of the function class for example w.r.t. efficiently computable preimages is necessary depends on the bigger construction in which the FE scheme is employed—and in particular on their reduction proof.[3] Moreover, when used as an assumption in a proof, then the efficiency restriction is a simple way to make consistency a falsifiable assumption [54]. Our UC proof in Section IV uses such a restriction, all other sections hold irrespective of the exact efficiency assumptions.

b) The semantics of the special symbols: We introduced the symbols $\diamond$ and $\perp$ with the idea of modeling invalid keys and invalid ciphertexts respectively: If the decryption of ct outputs $\diamond$ at any time in the game, the adversary wins because the preimage of $\diamond$ under every function is empty, i.e. $f^{-1}(\diamond) = \emptyset$ (see Definition 2), which results in an empty intersection; in particular there exists no x in the message space $x \in \mathcal{X} \cup \{\perp\}$ such that $f(x) = \diamond$ due to the definition of the function extension (Definition 2), which makes the adversary win the game. This captures that when the public parameters and the functional keys are honestly generated, then the decryption algorithm is not allowed to output $\diamond$ (recall that the symbol indicates an invalid key).

Analogously, if one of the decryption algorithm invocations outputs $\perp$ and another decryption algorithm invocation outputs a value $y_i \neq \perp$ then the adversary wins the game, as the intersection must be empty since the preimage of $\perp$ is $\perp$ and cannot be equal to the preimage of y_i (Definition 2). This captures that the ciphertext cannot be honestly generated, as the keys disagree on its validity.

Remark 1 (On the leakage function). As noted earlier, we deliberately ignore the leakage function f_0 when defining consistency requirements, since we perceive f_0, as already noted in [52], as a modeling artifact specific to the confidentiality definitions that we do not need to port to our new definition: the information captured by f_0 models the general leakage that an adversary *might* learn just by observing an *honestly generated* ciphertext. However, it seems unreasonable to assume that this must be guaranteed to be available. For instance, in the case of standard encryption schemes, computing the length of the plaintext is not guaranteed by the scheme, but the definition does formally not require that this information must be hidden. This distinction is further clarified in our UC treatment, where f_0 is the leakage function for the adversary, but not an actual function evaluated by (honest) parties.

B. Consistency with a dishonest Input Provider and Key Generator

We turn our attention to a stronger coalition against an honest decryptor, namely the setting in which both the input provider and the parameter/key generation entities are dishonest. In the experiment in Fig. 3, the adversary aims to outputs a malicious master public key mpk, two ciphertexts ct_1, ct_2 and a set of functional keys $\{\mathsf{sk}_{f_i}\}_{i\in[n]}$ that decrypt the ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ in an inconsistent way. Contrary to input consistency, the game considers two ciphertexts. This is the minimal number of ciphertexts to formulate and require a consistent decryption behavior w.r.t. different keys, i.e., have consistent behavior regarding invalid keys as modeled by the special $\diamond$ symbol (in the third line of Fig. 3). Minimality follows from the UC treatment that proves equivalence of this notion with

[3]Note that similar thoughts apply, e.g., to extractor games in interactive zero-knowledge proofs where the experiment need not be bounded by a polynomial, or in complexity leveraging arguments.

st-in-CONS$^{\mathsf{FE}}(1^\lambda, \mathcal{A})$

1 $(\mathsf{mpk}, \mathsf{ct}_1, \mathsf{ct}_2, \{(\mathsf{sk}_j, f_j)\}_{j\in[n]}) \leftarrow \mathcal{A}(1^\lambda)$ (**Assume** $\mathsf{sk}_j \neq \varepsilon$)

2 $y_{i,j} := \mathsf{Dec}(\mathsf{mpk}, f_j, \mathsf{sk}_j, \mathsf{ct}_i)$, for all $j \in [n], i \in \{1,2\}$

3 If $y_{1,j} = \diamond \wedge y_{2,j} \neq \diamond$ or $y_{1,j} \neq \diamond \wedge y_{2,j} = \diamond$ for any $j \in [n]$

Output 1

4 Let $F := \{(\mathsf{sk}_j, f_j)\}_{j\in[n]\wedge(y_{1,j}\neq\diamond\vee y_{2,j}\neq\diamond)}$

5 If F is empty then output 0

6 For each $i \in \{1,2\}$ do:

If $\bigcap_{j\in[n],(\cdot,f_j)\in F} f_j^{-1}(y_{i,j}) = \emptyset$

Output 1

7 Output 0

Fig. 3: Strong Input Consistency Experiment. Note that the consistency check in line 3 is technically redundant due to the disjunction that appears in the definition of F in line 5. However, for better accessibility we choose to highlight this important property explicitly.

an ideal functionality that guarantees the detection of invalid keys. As for input consistency, an adversary breaks consistency, if there exists no plaintext x, for at least one of the challenge ciphertexts, that can explain the decryption of some ciphertext ct under the different functional keys $\mathsf{sk}_{f_1}, \ldots, \mathsf{sk}_{f_n}$. Formally:

Definition 10 (Strong Input-Consistency)**.** *The functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *satisfies strong input consistency (or* st-in-CONS *for short), if for any polynomial-time adversary* $\mathcal{A}$ *interacting with experiment* st-in-CONS$^{\mathsf{FE}}$ *in Fig. 3, there exists a negligible function* negl *such that:*

$$\Pr[\text{st-in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A})] \leq \text{negl}(\lambda) \ .$$

a) Discussion: The experiment above strengthens the attack capabilities of the input consistency experiment. Here, the adversary can output a master public key mpk, ciphertexts $\{\mathsf{ct}_i\}_{i\in[2]}$ and a set of functional keys $\{\mathsf{sk}_j, f_j\}_{j\in[n]}$ (again, note that we do not give any guarantees for f_0 and the empty key).

In contrast to the weaker notion of the previous section, not all keys are valid, and hence the set F is defined as the set of key-function pairs (sk_j, f_j) that yield a decryption $y_{i,j} \neq \diamond$, for at least one ciphertext ct_i. Only keys in F can provoke a consistency breach. As we detail below, this assigns the correct meaning of key validity to $\diamond$. The challenger checks for a common explanation, i.e. whether there exists a message in the intersection of the preimages under the different functions $\{f_j\}_{j\in[n],(\cdot,f_j)\in F}$. If the intersection is empty, the adversary has generated a ciphertext ct_i with a decryption behavior that cannot be explained. Again, the symbols $\diamond$ and $\perp$ of Definition 2 deserve a special observation. A key's invalidity only provoke an inconsistency if not all decryptions w.r.t. this key yield $\diamond$. If a key yields consistently the "decryption" $\diamond$, this key is detected as invalid; otherwise, if for some ciphertext ct_i we have that exactly one decryption $y_{i,j} \neq \diamond$, the performed intersection check must yield the empty set. This behavior captures our intention that the decryptor's believes about the invalidity of a key cannot vary depending on what is decrypted.

Analogously, if a ciphertext is deemed invalid, i.e., $y_{i,j} = \perp$ for some key sk_j, then all keys in F must consistently declare this ciphertext invalid and agree on the single possible preimage $\{\perp\}$ (since special symbol $\perp$ maps only to $\{\perp\}$). Otherwise, the adversary has won.

C. Consistency with a dishonest Parameter/Key Generator

We now define consistency for the setting with an untrustworthy parameter/key generator. A notion we call setup-consistency. At first sight it might seem that setup consistency is implied by strong input consistency. Perhaps surprisingly, it is not. Setup consistency captures the important case where an authority tampers with the system's parameters and hence captures consistency in the presence of subversion attacks [10], [15]. We model setup consistency by formalizing the capabilities of an adversary. The adversary produces the master public key mpk and a functional key sk and defines inputs (out of which honest ciphertexts are generated). Note that we allow the adversary to specify two master public-keys (one for the input provider and one for the decryptor). We see the need for this in our UC treatment: if there were only one master public-key in the experiment, this would imply that one assumes a broadcast channel between the dishonest setup generator and the honest input provider and decryptor. Such a stronger assumption about the agreement on the master public key among all parties might be justified in some settings where a reliable public-key infrastructure (PKI) is available. However, it makes sense that a consistent FE scheme takes care of it by design independently of whether a PKI is available.

An attack breaks consistency, if the functional key sk together with the function f yields inconsistent output values with respect to the ciphertexts, i.e. the decryption of the ciphertexts under the functional key sk reveals a mismatch with respect to the input values and the declared function f (unless sk is identified as invalid). As for strong input consistency, consistency of key validity as modeled by $\diamond$ is captured using a disjunction, this time in the outer "If" statement. In more detail, we define the following:

Definition 11 (Setup Consistency)**.** *The functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *satisfies setup consistency (or* set-CONS *for short), if for any polynomial-time adversary* $\mathcal{A}$ *interacting with experiment* set-CONS$^{\mathsf{FE}}$ *in Fig. 4, there exists a negligible function* negl *such that:*

$$\Pr[\text{set-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A})] \leq \text{negl}(\lambda) \ .$$

set-CONS$^{\mathsf{FE}}(1^\lambda, \mathcal{A})$

$(\mathsf{mpk}_1, \mathsf{mpk}_2, \mathsf{sk}, f, x_1, x_2) \leftarrow \mathcal{A}(1^\lambda)$

(**Assume** $x_i \in \mathcal{X}$ and $\mathsf{sk} \neq \varepsilon$)

$\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, x_i)$, for all $i \in \{1, 2\}$

$I := \{i \mid i \in \{1, 2\} \wedge \mathsf{ct}_i \neq \mathsf{err}\}$

($|I| \neq 1$ for universal encryption property)

$y_i := \mathsf{Dec}(\mathsf{mpk}_2, f, \mathsf{sk}, \mathsf{ct}_i)$ for all $i \in I$

If $y_i \neq \diamond$ for some $i \in I$

  If $y_i \neq f(x_i)$ for some $i \in I$

    Output 1

Output 0

Fig. 4: Setup Consistency Experiment.

In addition, we say that FE *satisfies the* universal encryption property*, if in the above experiment,* $|I| \in \{0, 2\}$ *with overwhelming probability (where* I *is defined by the game).*

a) Discussion: It is instructive to see the nature of consistency attacks that an adversary can mount against a scheme. After the adversary $\mathcal{A}$ outputted two master public keys mpk_1 and mpk_2, a functional key sk, a function f and two chosen messages x_1 and x_2, the challenger encrypts the messages under mpk_1 to generate $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{mpk}_1, x_i)$. Now, we are interested in the functional behavior of all valid encryptions that the input provider produces (i.e., that do not return an err symbol upon encryption because of an bogus mpk_1). Let us, for concreteness, discuss the case where both encryptions are valid: if both decryption invocations under sk return the special symbol $\diamond$ then the adversary does not win the game (in this case, the key is deemed invalid). However, if only one of the two outputs $\diamond$ the adversary immediately wins the game (as there can be no value x_i in the domain that yields $f(x_i) = \diamond$ (see Definition 2)). Recall that this behavior captures our intention that the decryptor's believes about a key's validity cannot vary depending on what is decrypted. Now, we consider the case where both decryption attempts yield values $y_i \neq \diamond$. In this case, to fulfill consistency, both of these values must satisfy $f(x_i) = y_i$, otherwise the attacker has broken consistency. If the decryption procedure would output $y_i = \perp$ a security breach happens. In more detail, considering that honestly generated ciphertexts are committed to a real message (otherwise the decryption must be considered inconsistent). By Definition 2 the adversary wins in this case since no message $x_i \neq \perp$ maps to $\perp$.

b) Universal encryption property: We also consider a stronger property that we term *universal encryption*. It requires that either both encryptions are valid or none is. While this is not a core consistency notion, which we deem to be about properties of decryption, universal encryption should be considered by applications if needed. If the property does not hold, a maliciously generated mpk_1 may only allow for the encryption of a subset of the plaintext space. When capturing ideal confidentiality guarantees in UC we see that this is in fact a side-channel that can provide additional leakage. In fact, it is easy to come up with a scheme, where the first bit of Alice's input is leaked due to this side-channel: take a secure base FE scheme and prefix the master public key by a global distinct identifier $\mathtt{id}$. The encryption algorithm of the modified scheme encrypts using mpk exactly as the base scheme does whenever the given master public key has the form $\mathtt{id}||\mathsf{mpk}$. However, if given a public key with a different identifier $\mathtt{id}'||\mathsf{mpk}$, the encryption algorithm throws an error if the input message starts with bit 0; if the input starts with bit 1, then the input message is encrypted just as in the base scheme. Hence, although the new scheme never encrypts differently than the base scheme, signaling the error perfectly correlates with Alice's input message starting with bit 0.

We note that universal encryption follows generically from an efficiently computable membership-test for the support of Setup and the perfect correctness of an FE scheme and refer to the UC treatment to quantify the gain in terms of additional security provided by the universal encryption property.

c) Strong robustness against subversion: Looking ahead to Section IV where we present the justification of the game by showing that it admits the realization of a natural ideal repository with access control, we see that in fact, we must insist that the inputs provided by Alice do functionally match the values that Bob decrypts. Otherwise, the guarantee for honest parties in this setting with subversion of parameters would be too weak, as it would merely enforce consistent decryption—but potentially with respect to a common preimage x' never intended by Alice! This is a form of robustness not implied by strong input consistency or verifiable functional encryption [11] and also goes beyond capturing key validity only. This shows that a separate notion for the case of subverted setup, namely setup-consistency, is indeed desirable.

D. Concluding remarks

To formally relate the new consistency notions, we investigate their relation to the CPA/CCA/CFE security notions and can conclude that the notion is orthogonal as depicted in Fig. 5. Furthermore, we verify the relations among the consistency notions. We note in passing that due to the disjoint corruption sets in the definitions of input consistency and setup consistency, the conjunction of the two notions is not guaranteed to yield protection against the collution of Alice and Charlie. We give the detailed proofs of the relations in Section C. Finally, for further discussion on how the properties can be applied to different forms of FE in the literature we refer to Section B.

IV. UC Consistency for Functional Encryption

Thanks to the foundational work of Matt and Maurer [52], we can accurately characterize the goal that

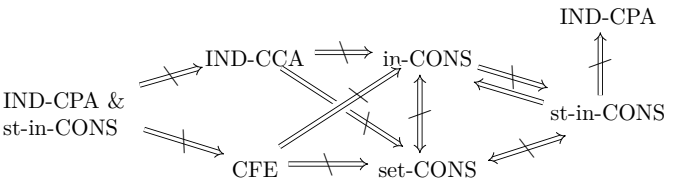


Fig. 5: Relations between consistency and confidentiality, where the crossed arrows indicate "does not imply" and "&" denotes both properties simultaneously.

functional encryption tries to achieve in a constructive sense: the goal is to realize (by means of a simple protocol using the FE scheme in the intended way) a repository, where an input provider can store a value x—and receive a unique handle h generated by an abstract function $\mathsf{getHandle}$ [52]—and where another entity can, using the handle h, obtain the values $f(x)$ if and only if the function f was explicitly granted for evaluation by a third-party (the setup manager). This model puts forth a form of access control: even a malicious receiver (or a coalition of malicious receivers) cannot obtain more information than what $f(x)$ reveals about the value stored under handle h. The goal of plain functional encryption is to realize this strong repository from a public one, where the receivers have read access to all values stored in the repository. The treatment in [52] focuses on confidentiality (in the sense that only the receiver is corrupted). Our work extends the treatment to all other cases, including malicious input providers and/or malicious setup generators. We thereby extend their functionality to precisely capture the realistic composable notion of consistency in functional encryption. In the following, $\mathcal{F}$ denotes the concrete functionality class of an FE scheme (dropping the index λ), f_0 the distinguished leakage function and $\mathcal{F}^+$ the set $\mathcal{F} \setminus \{f_0\}$, $\mathcal{X}$ the input domain and $\mathcal{Y}$ the range of functions. We assume three distinct party identifiers $\mathtt{A}, \mathtt{B}, \mathtt{C}$. Furthermore, if $\mathtt{p}$ is an identifier, $\mathtt{p}_i$ denotes the unique identifier derived from $\mathtt{p}$ that includes a (prefix-free) encoding of the number i.

1) The Ideal Functionality: Our ideal functionality $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ is a direct extension of the one given in [52] and is formally described below. In a nutshell, the functionality is defined to interact with three types of parties or roles, denoted by $\mathtt{A}$ for the input provider, $\mathtt{B}$ for the receiver or decryptor, and $\mathtt{C}$ for the third party that manages setup and key distribution. As long as $\mathtt{A}$ and $\mathtt{C}$ are honest, we enjoy the usual confidentiality guarantees for FE as described above. UC consistency on the other hand is a property that protects an *honest* receiver against malicious input providers and/or setup generator: if $\mathtt{A}$ is corrupt, we allow it—instead of providing an input from the domain—to specify that the input is still undefined ($x = \mathsf{unknown}$). For this case the functionality associates with the handle h a set of allowed values (initially the entire domain). The functions assigned to the receivers are steered by the setup manager $\mathtt{C}$. If this party is corrupt, it can stop the ideal functionality from accepting inputs for Alice. This is unavoidable as Alice has to wait for the public parameters. For an honest party $\mathtt{B}_i$ in this setting, the ideal functionality implements the following guarantee: when an output value for function f and handle h is requested, the functionality lets the adversary choose the return value y upon this request but enforces that the set of values associated with handle h is reduced to stay consistent with y. To ensure that this yields an efficient functionality, we assume an efficiently computable map $\mathsf{preMap}()$ satisfying $\mathsf{preMap}(\mathcal{M}, f, y) = \{x' \in \mathcal{M} \mid f(x') = y\}$. Hence, at any time, each handle h is committed to a set of possible values $\mathcal{M} \subseteq \mathcal{X}$ that are consistent with any output y generated by an honest receiver when requesting the output of the function value for an assigned function f for this handle h.

a) On Confidentiality and Subversion: The confidentiality guarantees are the same as in [52] except when additional parties beyond receivers are corrupted. In such cases of multiple corrupted roles, we allow the adversary to read out the stored value per handle since in such cases, confidentiality is generally lost (a malicious setup generator can collude with the corrupt receivers).

However, when *only* the setup manager is corrupted, we obtain an interesting special case: first, since the setup manager has no direct access to the ciphertexts in the real world [52], one would expect that in this case we still enjoy full confidentiality. Quite surprisingly, this is not the case in general: for a subverted setup, whether a ciphertext can be created ($\mathsf{ct}_i \neq \mathsf{err}$) potentially depends on which element of the plaintext space is encrypted which is a side-channel revealing information about the plaintext, which is the reason to admit leakage to the dishonest setup provider ($\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ generates a public-delayed output to $\mathtt{C}$). Looking ahead, by relying on the universal encryption property we can remove the side-channel and formally obtain the stronger repository which we capture by $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep}^*,(\mathcal{F}^+,f_0)}$ below, where the public-delayed output to $\mathtt{C}$ is replaced by a private-delayed output in this case. This assigns a clean composable semantic to this additional property introduced in the previous section.

2) The FE Protocol: We define the protocol $\pi^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathsf{FE}}$ for parties $\mathtt{A}$, $\mathtt{B}_i$ and $\mathtt{C}$, where party $\mathtt{A}$ acts as an input provider, parties $\mathtt{B}_i$ act as the receivers, and party $\mathtt{C}$ acts as the setup manager. The distribution of the public key to the input provider and the receivers is done via an authentic broadcast channel between $\mathtt{C}$ and $\mathtt{A}$ and between $\mathtt{C}$ and the receivers $\{\mathtt{B}_i\}_{i\in[t]}$. Note that we do not require broadcast from $\mathtt{C}$ to every participant, but only to the set of parties having the same role, which is the minimal assumption we have to make—to see this, note that otherwise, there could be two parties with the same role that operate with public parameters belonging to different schemes, among which clearly no consistency has to exist, e.g, when one scheme deems all legitimate encryptions w.r.t. to the other public key invalid under its own public key.

Functionality $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ / $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep}^*,(\mathcal{F}^+,f_0)}$

The functionality is parameterized by function class $\mathcal{F}$, the number t of decryptors/receivers, and by three distinct party identities $\mathcal{P} := \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$. These dummy parties interact with the functionality and identify particular roles.

Setup. Upon receiving input $(\text{SETUP}, \text{sid})$ via dummy party $\mathtt{C}$ (or from the adversary on behalf of corrupted $\mathtt{C}$), set $\mathtt{setup} \leftarrow \mathsf{true}$, $R_i \leftarrow \emptyset$, for each $i \in [t]$. Ignore the request if the party-id does not correspond to $\mathtt{C}$. Output $(\text{SETUP}, \text{sid})$ to the adversary to indicate that setup is completed.

Input: Upon receiving input $(\text{WRITE}, \text{sid}, x)$ via dummy party $\mathtt{A}$ (or from the adversary on behalf of corrupted $\mathtt{A}$), and if $\mathtt{setup} = \mathsf{true}$, do the following:

- If pid $\mathtt{A}$ is honest then verify that $x \in X$ (ignore request otherwise). If party $\mathtt{C}$ is honest, then compute handle $h \leftarrow \mathsf{getHandle}$ and store $M[h] \leftarrow (x, \{x\})$ and return $(\text{WRITTEN}, \text{sid}, h)$ to the calling party.
  If party $\mathtt{C}$ is corrupted, then do one of the following depending on the version of the repository:

  Rep: Provide *public* delayed-output to the adversary and do the previous actions only upon receiving ACK for this operation.

  Rep*: Provide *private* delayed-output to the adversary and do the previous actions only upon receiving ACK for this operation.

- If pid $\mathtt{A}$ is marked as corrupted, verify that $x \in X \cup \{\perp, \mathsf{unknown}\}$ (and ignore the request otherwise). Then choose $h \leftarrow \mathsf{getHandle}$ and store $M[h] \leftarrow (x, \{x\})$. Output $(\text{WRITTEN}, \text{sid}, h)$ to the adversary.

Access Management: Upon receiving input $(\text{ASSIGN}, \text{sid}, f, i)$ via dummy party $\mathtt{C}$ (or from the adversary on behalf of $\mathtt{C}$), do the following: if $f \in \mathcal{F}^+$, then update $R_i \leftarrow R_i \cup \{f\}$ and output $(\text{ASSIGNED}, \text{sid}, f, i)$ to the adversary.

Output: Upon receiving $(\text{READ}, \text{sid}, h, f)$ from some caller via dummy party $\mathtt{B}_i$ (or from the adversary on behalf of corrupted $\mathtt{B}$), first parse $M[h]$ as $(x, \mathcal{M})$. In case $M[h] = \perp$, return noData.

- If $\mathtt{B}_i$ is honest do:
  1) If $f \notin R_i$ then give up activation. Otherwise, if $x \in X$ and $f \in R_i$, then return $(\text{READ}, \text{sid}, f(x))$ to the caller; else if $x = \perp$ then return $(\text{READ}, \text{sid}, \perp)$ to the calling party. $(\star\star\star)$
  2) Otherwise, if $x = \mathsf{unknown}$, output $(\text{READ}, \text{sid}, h, f)$ to the adversary. Upon receiving $(\text{READ}, \text{sid}, h, f, (x', y))$ from the adversary, do the following:
     a) If $x' \in \mathcal{M}$, set $M[h] \leftarrow (x', \{x'\})$. Output$(\text{READ}, \text{sid}, f(x'))$ to the calling party.
     b) Else compute $\mathcal{M}_{\text{new}} \leftarrow \mathsf{preMap}(\mathcal{M}, f, y)$.
        i) If $\mathcal{M}_{\text{new}} = \emptyset$ then pick some x'' at random from $\mathcal{M}$ and store $M[h] \leftarrow (x'', \{x''\})$. Output $(\text{READ}, \text{sid}, f(x''))$ to the calling party. $(\star\star)$
        ii) Otherwise, update the entry either by $M[h] \leftarrow (\mathsf{unknown}, \mathcal{M}_{\text{new}})$ if $\mathcal{M}_{\text{new}}$ is not a singleton set or by $M[h] \leftarrow (x^*, \mathcal{M}_{\text{new}})$ in case $\mathcal{M}_{\text{new}} = \{x^*\}$ for some x^*. Output $(\text{READ}, \text{sid}, y)$ to the calling party.
- If $\mathtt{B}_i$ is marked as corrupted but none of $\mathtt{A}$ or $\mathtt{C}$, then do the following: If $f \in R_i$ then return $(\text{READ}, \text{sid}, f(x))$ (for the x guaranteed to exist since the input provider is honest) and if $f = f_0$ then return $(\text{READ}, \text{sid}, f_0(x))$ to the adversary. Otherwise, give up activation.
- If $\mathtt{B}_i$ is corrupted alongside $\mathtt{A}$ or $\mathtt{C}$, then output $(\text{READ}, \text{sid}, h, f)$ to the adversary and upon receiving $(\text{READ}, \text{sid}, h, f, (x, y))$ from the adversary output $(\text{READ}, \text{sid}, y)$ to the calling party.

Additional adversarial interaction (aside of corruption):

- On top of the standard pid-wise corruption mechanism of UC, the following additional capability is given to the adversary: If and only if some $\mathtt{B}_i$ and at least one more party among $\{\mathtt{A}, \mathtt{C}\}$ is corrupted, then the adversary is allowed to query $(\text{REVEAL}, \text{sid}, h)$ upon which $M[h]$ is revealed to the adversary.

Fig. 6: The ideal repository for consistency. We assume standard corruption handling as defined in [25] and do not describe it specifically.

As in [52], the protocol requires a point-to-point secret channel between $\mathtt{C}$ and each of the receivers $\mathtt{B}_i$, and we assume a basic storage repository, where the input provider (and only the input provider) can store messages of its choice (and only the receivers $\mathtt{B}_i$ can access them). Note that in UC, these hybrid functionalities are defined and invoked by the protocol. Hence, if a scheme would require the random oracle model, $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ would additionally invoke a random-oracle functionality (which is needed to achieve CFE security for example). The channel functionalities and the basic real-world repository are given in Section H-A for completeness. In a nutshell, the protocol works as follows: Party $\mathtt{C}$ generates the public keys (and sends them to the other parties) and assigns functions to parties $\mathtt{B}_i$ by sending the functional keys. Party $\mathtt{A}$ does provide the input to the real-world repository by encrypting the input $x \in \mathcal{X}$, and storing valid ciphertexts in the repository. Using the obtained handle h, the ciphertext can be accessed by some party $\mathtt{B}_i$ and decrypted using a (valid) sk corresponding to an assigned function f and the result is provided as output. The protocol is specified in Section H-B.

3) UC Realization: We provide a detailed security analysis with respect to the different corruption sets possible in the system and conclude that each of our consistency games captures exactly what we intended. The theorem therefore also gives guarantees for a scheme that does only achieve a subset of the properties (such as CFE and setup or input consistency): in this case, the scheme can only be safely used in contexts, where certain people are trusted.[4]

Theorem 1. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a functional encryption scheme for functionality class* $\mathcal{F}$*, and let* $\mathtt{A}, \mathtt{B}, \mathtt{C}$ *be three identifiers. Protocol* $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *UC-realizes* $\mathsf{Func}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *(under static corruption) under the following conditions:*

- *If party* $\mathtt{A}$ *is corrupted, and* $\mathtt{C}$ *is honest (and potentially a subset of receivers is corrupted), then* in-CONS *is a sufficient requirement on* FE *such that* $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *realizes* $\mathsf{Func}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$.
- *If party* $\mathtt{C}$ *is corrupted and party* $\mathtt{A}$ *is honest (and possibly a subset of receivers is corrupted), then* set-CONS *is a sufficient requirement on* FE *such that* $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *realizes* $\mathsf{Func}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$.
- *If parties* $\mathtt{A}$ *and* $\mathtt{C}$ *are corrupted (and possibly a subset of receivers is corrupted), then* st-in-CONS *is a sufficient requirement on* FE *such that* $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *realizes* $\mathsf{Func}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$.
- *If both* $\mathtt{A}$ *and* $\mathtt{C}$ *are honest, and only a subset of receivers is corrupted, then CFE security is a sufficient requirement on* FE *such that* $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *realizes* $\mathsf{Func}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$.

The above statements hold for the repository $\mathsf{Func}_{\mathrm{Rep}^*,(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *if* FE *has the universal encryption property.*

[4]Such trust assumptions could be formally modeled in UC by defining certain parties to be incorruptible, that is, the corresponding protocol machine would ignore corruption requests.

Conversely, the consistency notions in-CONS, set-CONS, *and* st-in-CONS *are the respectively necessary requirements on the scheme* FE *in order for* $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *to realize the specified security guarantees by* $\mathsf{Func}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ *w.r.t. a given corruption set in the above listing.*

Note that the second part of the theorem justifies our game-based notions for consistency. We refer to Section H for the proof.

V. Consistency Analysis of Selected Functional Encryption Schemes

In this section, we analyze the single-input functional encryption schemes for the inner product functionality based on the MDDH assumption regarding input consistency. These schemes have been initially introduced for the DDH assumption in [7] and extended to the MDDH assumption in [4]. This analysis contains of two parts: The analysis for the bounded-norm functionality class $\mathcal{F}^{m,X,Y}$ and the functionality class $\mathcal{F}_p^m$ over $\mathbb{Z}_p^m$ which we define here for completeness:

Inner Product (IP) over $\mathbb{Z}_P$. Let $\mathcal{F} = \{\mathcal{F}_{P_\lambda}^m\}_{\lambda\in\mathbb{N}}$ be a family (indexed by λ) of sets $\mathcal{F}_{P_\lambda}^m$, where P_λ is a modulus of length λ. Omitting the index λ, the set $\mathcal{F}_P^m = \{f_{\boldsymbol{y}} : \mathbb{Z}_P^m \to \mathbb{Z}_P, \text{ for } \boldsymbol{y} \in \mathbb{Z}_P^m\}$ where

$$f_{\boldsymbol{y}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle \bmod P$$

defines the inner-product operation over $\mathbb{Z}_P$.

Bounded-Norm IP over $\mathbb{Z}$. Let $\mathcal{F} = \{\mathcal{F}^{m,X_\lambda,Y_\lambda}\}_{\lambda\in\mathbb{N}}$ be a family (indexed by λ) of sets $\mathcal{F}^{m,X_\lambda,Y_\lambda}$. Omitting the index λ, the set $\mathcal{F}^{m,X,Y} = \{f_{\boldsymbol{y}} : \mathbb{Z}_X^m \to \mathbb{Z}, \text{ with } \boldsymbol{y} \in \mathbb{Z}_Y^m\}$, where $\mathbb{Z}_X^m := \{\boldsymbol{x} \in \mathbb{Z}^m, \text{ with } \|\boldsymbol{x}\|_\infty < X\}$, $\mathbb{Z}_Y^m := \{\boldsymbol{y} \in \mathbb{Z}^m, \text{ with } \|\boldsymbol{y}\|_\infty < Y\}$ and where $f_{\boldsymbol{y}}(\boldsymbol{x}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$, defines the bounded-norm inner-product over $\mathbb{Z}$.

A. Overview

We observe that some schemes for the inner product functionality seem to be input consistent, but without specific modifications they are not. Therefore, we analyze these schemes and the corresponding modifications for input consistency in this section. For both of the mentioned functionality classes we obtain negative results, i.e. the analyzed scheme is neither input consistent for the functionality class of bounded-norm inner products nor for the inner products calculated over $\mathbb{Z}_p^m$. To prove this, we present an attack for both cases in Section V-B.

Beside this, we introduce a natural modification of the above functionality class and denote it by $\mathcal{F}_{P,L}^m$ below w.r.t. which the MDDH scheme of Fig. 7 is an input consistent functional encryption scheme. This is formally defined and proven in Section V-C.

We then present a modification of the inner-product scheme described in Fig. 7, which covers a more restricted functionality class, $\mathcal{P}_p^m$ introduced below in Section V-D which is input consistent.

$\underline{\mathsf{Setup}(1^\lambda):}$
$\mathcal{G} := (\mathbb{G}, p, g) \leftarrow \mathsf{GGen}(1^\lambda)$,
$\mathbf{A} \leftarrow \mathcal{D}_k, \mathbf{W} \leftarrow \mathbb{Z}_p^{m\times(k+1)}$
$\mathsf{mpk} := (\mathcal{G}, [\mathbf{A}]_g, [\mathbf{WA}]_g), \mathsf{msk} := \mathbf{W}$
Return $(\mathsf{mpk}, \mathsf{msk})$

$\underline{\mathsf{Enc}(\mathsf{mpk}, \boldsymbol{x} \in \mathbb{Z}_X^m \; \boldsymbol{x} \in \mathbb{Z}_p^m):}$
$\boldsymbol{r} \leftarrow \mathbb{Z}_p^k, \begin{pmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c}_1 \end{pmatrix} := \begin{pmatrix} -\mathbf{A}\boldsymbol{r} \\ \boldsymbol{x} + \mathbf{WA}\boldsymbol{r} \end{pmatrix}$
Return $\mathsf{ct} := [\begin{pmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c}_1 \end{pmatrix}]_g \in \mathbb{G}^{k+m+1}$

$\underline{\mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, \boldsymbol{y} \in \mathbb{Z}_Y^m \; \boldsymbol{y} \in \mathbb{Z}_p^m):}$
Return $\mathsf{sk}_{\boldsymbol{y}} := \mathbf{W}^\top \boldsymbol{y} \in \mathbb{Z}_p^{k+m+1}$

$\underline{\mathsf{Dec}(\mathsf{mpk}, \boldsymbol{y}, \mathsf{sk}_{\boldsymbol{y}}, \mathsf{ct}):}$
Parse $\mathsf{ct} := [\begin{pmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c}_1 \end{pmatrix}]_g$
$C := [\boldsymbol{c}_1^\top \boldsymbol{y} + \boldsymbol{c}_0^\top \mathsf{sk}_{\boldsymbol{y}}]_g$
Return C
Return $\log(C)$

Fig. 7: FE for the standard classes $\mathcal{F}^{m,X,Y}$, $\mathcal{F}_p^m$, and for classes $\mathcal{F}_{p,L}^m$, $\mathcal{P}_p^m$ based on the $\mathcal{D}_k$-MDDH assumption.

B. Inconsistency of the Plain Schemes

The final output computation in the decryption procedure of the MDDH based inner product schemes requires a discrete logarithm computation. To ensure that this computation is efficiently possible, using for example Pollard's kangaroo method [58], it is required that the computed inner product lies within a specific polynomial bounded interval $\{0, \ldots, L\}$, i.e. $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \{0, \ldots, L\}$ with a known L. By the correctness of the scheme, we assume that for every encrypted vector $\boldsymbol{x}$, with $\|\boldsymbol{x}\|_\infty < X$, and every functional key corresponding to $\boldsymbol{y}$, with $\|\boldsymbol{y}\|_\infty < Y$, the decryption gives us the right output $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and otherwise, it outputs $\perp$. Now, we can break the input consistency of the scheme by maliciously generating a ciphertext ct' such that, when decrypted using a functional key $\mathsf{sk}_{\boldsymbol{y}_1}$, the inner product does not lie within the polynomial bound and therefore outputs $\perp$, and for another functional key $\mathsf{sk}_{\boldsymbol{y}_2}$ it lies within the polynomial bound and therefore the decryption procedure outputs a valid inner product. For the case of the inner product computation over $\mathbb{Z}_p$, the attack works in the same way, since the polynomial bound L for the discrete logarithm computation in the last step must be significantly smaller than p. Therefore, it is still possible to find a value $L+1$ such that the decryption procedure outputs $\perp$ when used with one of the functional keys and a valid inner product when used with the other functional key.

Theorem 2. *The functional encryption scheme* FE *described in Fig. 7 for the functionality class* $\mathcal{F}^{m,X,Y}$ *and* $\mathcal{F}_p^m$*, with p prime, is not input consistent. Namely, there exists a PPT adversary $\mathcal{A}$ such that*

$$\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 1 \ .$$

The details of the proof for this theorem can be found in Section D-A.

C. Consistency for Inner-product Schemes

Now, we present the modified inner product functionality class and state the theorem that when instantiating the scheme in Fig. 7 for this functionality class it achieves input consistency. The main idea of the new functionality class is that we allow the decryption procedure to output a new error symbol oob in the case that it is not able to do the discrete logarithm computation in the last step. The preimage of the oob symbol is then defined as all the $\boldsymbol{x}$ such that $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ exceeds the polynomial bound necessary for the logarithm computation. This allows to prevent the input consistency attack described in the proof of Theorem 2.

Modified IP over $\mathbb{Z}_P$. Let $\mathcal{F} = \{\mathcal{F}_{P_\lambda, L_\lambda}^m\}_{\lambda\in\mathbb{N}}$ be a family (indexed by λ) of sets $\mathcal{F}_{P_\lambda, L_\lambda}^m$, where P_λ is a modulus of length λ and L_λ . Omitting the index λ, the set $\mathcal{F}_{P,L}^m = \{f_{\boldsymbol{y}} : \mathbb{Z}_P^m \to \mathbb{Z}_P, \text{ for } \boldsymbol{y} \in \mathbb{Z}_P^m\}$ where

$$f_{\boldsymbol{y}}(\boldsymbol{x}) = \begin{cases} \langle \boldsymbol{x}, \boldsymbol{y} \rangle \bmod P & \text{if } \langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \{0, \ldots, L\} \\ \mathsf{oob} & \text{if } \langle \boldsymbol{x}, \boldsymbol{y} \rangle > L \ . \end{cases}$$

defines the inner-product operation over $\mathbb{Z}_P$.

The out-of-bound symbol oob is thereby defined as the output of the function when the resulting inner product computation does not lie within a polynomial bound $\{0, \ldots, L\}$; consequently, its preimage is $f_{\boldsymbol{y}}^{-1}(\mathsf{oob}) = \{\boldsymbol{x} \in \mathbb{Z}_P^m : \langle \boldsymbol{x}, \boldsymbol{y} \rangle > L\}$. The preimage for all other outcomes is $f_{\boldsymbol{y}}^{-1}(z) = \{\boldsymbol{x} \in \mathbb{Z}_P^m : \langle \boldsymbol{x}, \boldsymbol{y} \rangle = z\}$. When we consider the functional encryption scheme for the modified inner-product functionality $\mathcal{F}_{p,L}^m$, it achieves input consistency.

Theorem 3. *The functional encryption scheme* FE *described in Fig. 7 for the functionality class* $\mathcal{F}_{p,L}^m$*, with p prime, is input consistent. Namely, for any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 0 \ .$$

The detailed proof is given in Section D-A.

D. Consistency of a related Exponential Inner Product Scheme

We can turn the inner product functional encryption scheme into a scheme that is consistent and allows the evaluation of exponentiated inner products. We can achieve this by omitting the discrete logarithm computation in the end of the decryption procedure and just output the value $g^{\langle \boldsymbol{x}, \boldsymbol{y} \rangle}$. More formally:

Exponential IP over $\mathbb{Z}_p$. Let $\mathcal{P} = \{\mathcal{P}_{p_\lambda}^m\}_{\lambda\in\mathbb{N}}$ be a family of sets $\mathcal{P}_{p_\lambda}^m$, where p is a prime of length λ and $\mathbb{G}$ a group

of size p and generator g.[5] The sets are defined by $\mathcal{P}_p^m = \{f_{g,\boldsymbol{y}} : (\mathbb{Z}_p^m) \to \mathbb{G},\ \text{with } \boldsymbol{y} \in \mathbb{Z}_p^m\}$ where

$$f_{g,\boldsymbol{y}}(\boldsymbol{x}) = g^{\langle \boldsymbol{x},\boldsymbol{y}\rangle} \ .$$

For the scheme in Fig. 7, the input consistency property for this class follows similarly to the proof of Theorem 3.

Theorem 4. *The functional encryption scheme* FE *for the functionality class* $\mathcal{P}_p^m$*, with p prime, described in Fig. 7 is input consistent. Namely, for any PPT adversary* $\mathcal{A}$ *it holds that* $\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 0$.

VI. Consistency Compilers

In this section, we present black-box compilers that achieve consistency under the different corruption sets. Depending on the trust model and the efficiency requirements of a given application, an FE scheme can thus be lifted to withstand certain types of corruptions. As a rule of thumb, protecting against input consistency is cheaper than strong-input consistency, whereas setup-consistency resides between the two. Also, rather intuitively, achieving CCA security instead of CPA security (in combination with consistency) is more expensive. More concretely, we show that input consistency is achievable using only NIZKs (instead of NIWIs in the other compilers) and a single instance of a functional encryption scheme (compared to three resp. four in the following compilers). For the setup-consistency compiler, which we base on NIWIs, we need to run three instances of the functional encryption scheme, which compared to the strong-consistency compiler is one instance less. For strong input consistency, we show a close relationship to VFE which complements [11] by showing that their compiler is UC-secure and implies strong-input consistency. That compiler uses NIWIs and four instances of the underlying FE scheme. Finally, and as a result of independent interest, we also show how to obtain generic security lifting from CPA to CCA on the fly using the Naor-Yung approach [55].

A. Input and Strong Input Consistency Compilers

1) Input consistency: To turn a functional encryption scheme into an input consistent functional encryption scheme, we make use of NIZKs proofs. In more detail, we augment the output of the encryption algorithm with a NIZK proof that an underlying plaintext for this ciphertext exists. The formal description of this compiler is presented in Fig. 17 of the supplementary material. The soundness of the NIZK proof ensures consistency, by preventing an adversary from generating ciphertexts in a dishonest manner. This is formally stated in Theorem 16 and the security preservation for CPA and CFE security are provided in Theorem 17 and Theorem 18, respectively.

[5] As mentioned in Section II, we omit λ for simplicity.

The advanced input consistency compiler works in a similar manner as the input consistency compiler, but with the main difference that to achieve CCA security, although the underlying scheme is CPA, we make use of the Naor-Yung approach [55] by executing two functional encryption instances in parallel and prove in zero-knowledge that ciphertexts generated under the different instances encrypt the same message. The proof that our compiler achieves the desired CCA security based on a CPA secure FE scheme works along the lines of [55], but with some technical differences. The formal compiler is presented in Fig. 20 of the supplementary material. We provide a formal proof in Theorem 19. The input consistency follows with the same arguments as for the compiler above. We state it formally in Theorem 20.

To conclude the treatment on input consistency, we provide, at the end of Section E, some ideas regarding instantiations of the compiler from (several) standard assumptions.

2) Strong Input consistency: To achieve strong input consistency, we provide a general statement that shows that the verifiability property of VFE, introduced in [11], can be understood as providing strong input consistency using a straightforward reduction. Since VFE schemes come with two algrithms for cihpertext and functional key verification, we can derive a (standard) FE scheme that, as part of the decryption procedure, verfies ciphertexts (and returns $\perp$ if the check fails) and also key-function pairs (and returns $\diamond$ if the check fails), and only decrypts ciphertexts that pass both of these tests. The transformation clearly preserves the confidentiality notion of the underlying VFE scheme and due to this modular reduction, we also directly inherit a strong input consistency compiler from a VFE scheme. The full treatment is given in Section F.

B. Setup Consistency Compilers

1) First Compiler: To achieve setup consistency, we need to prevent the generation of malicious functional keys under maliciously generated parameters. While we can still rely on honest encryption procedures, the parameters are chosen by the adversary beforehand and we cannot rely on a common-reference string generated by the adversary.

We replace the role of the NIZK proof in the previous section by a non-interactive witness indistinguishable (NIWI) proof. NIWI proofs allow us to achieve similar properties in terms of correctness and soundness, as provided by the NIZK proof, without relying on a common reference string. As a trade-off, we cannot rely on the zero-knowledge property but on witness-indistinguishability instead, which we prove to be sufficient. However, our compiler needs to run three different instances of the same functional encryption scheme in parallel. The relation R_{set} of the NIWI proof is formally defined in Fig. 9 and formalizes that (during the key generation procedure) two out of the three generated functional keys are faithfully generated (where the different random coins involved in key generation

$\underline{\mathsf{Setup}'(1^\lambda):}$
For $i \in [3]$:
$(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda; s_i)$
$\mathsf{mpk}' := \{\mathsf{mpk}_i\}_{i\in[3]}$
$\mathsf{msk}' := \{(\mathsf{msk}_i, s_i)\}_{i\in[3]}$
Return $(\mathsf{mpk}', \mathsf{msk}')$

$\underline{\mathsf{KeyGen}'(\mathsf{mpk}', \mathsf{msk}', f):}$
Parse $\mathsf{mpk}' := \{\mathsf{mpk}_i\}_{i\in[3]}$,
$\mathsf{msk}' := \{(\mathsf{msk}_i, s_i)\}_{i\in[3]}$
For $i \in [3]$:
  $r_i \in \{0,1\}^\lambda$
  $\mathsf{sk}_{f,i} = \mathsf{KeyGen}(\mathsf{msk}_i, f; r_i)$
Generate $\pi \leftarrow \mathsf{NIWI.Prove}(1^\lambda, z, w)$:
  $z = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3]}, f)$,
  $w = (\{\mathsf{msk}_i\}_{i\in[3]}, \{r_i\}_{i\in[3]}, \{s_i\}_{i\in[3]})$
and L defined corresponding to R_{set} (Fig. 9)
Return $\mathsf{sk}'_f = (\{\mathsf{sk}_{f,i}\}_{i\in[3]}, \pi)$

$\underline{\mathsf{Enc}'(\mathsf{mpk}', x):}$
Parse $\mathsf{mpk}' := \{\mathsf{mpk}_i\}_{i\in[3]}$
For $i \in [3]$:
$\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, x)$
If $\exists i \in [3] : \mathsf{ct}_i = \mathsf{err}$ then return err
Return $\mathsf{ct}' = (\mathsf{mpk}', \{\mathsf{ct}_i\}_{i\in[3]})$

$\underline{\mathsf{Dec}'(\mathsf{mpk}', f, \mathsf{sk}'_f, \mathsf{ct}'):}$
Parse $\mathsf{mpk}' := \{\mathsf{mpk}_i\}_{i\in[3]}$,
  $\mathsf{sk}'_f := (\{\mathsf{sk}_{f,i}\}_{i\in[3]}, \pi)$,
  $\mathsf{ct}' := (\mathsf{mpk}'', \{\mathsf{ct}_i\}_{i\in[3]})$
$z := (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3]}, f)$
If $\mathsf{mpk}' = \mathsf{mpk}''$
  If $\mathsf{NIWI.Verify}(1^\lambda, z, \pi) = 1$
    For $i \in [3]$:
      $y_{f,i} := \mathsf{Dec}(\mathsf{mpk}_i, f, \mathsf{sk}'_{f,i}, \mathsf{ct}_i)$,
      $y \leftarrow \mathrm{MajVal}(\{y_{f,i}\}_{i\in[3]})$
      Return y
Return $\diamond$

Fig. 8: Setup consistency compiler. MajVal$(\cdot)$ calculates and returns the majority value of the input values, if there is a clear majority and $\diamond$ otherwise.

$\underline{\text{Relation } R_{\mathsf{set}}:}$
Instance: $z = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3}, f)$
Witness: $w = (\{\mathsf{msk}_i\}_{i\in[3]}, \{r_i\}_{i\in[3]}, \{s_i\}_{i\in[3]})$
$R_{\mathsf{set}}(z, w) = 1$ if and only if for at least two indices $j_1, j_2 \in [3]$, $j_1 \neq j_2$ we have:
1. The key sk_{f,j_1} for function f is generated using msk_{j_1} with r_{j_1} and related to mpk_{j_1} which is generated using s_{j_1}. Formally:
   $\mathsf{sk}_{f,j_1} = \mathsf{KeyGen}(\mathsf{mpk}_{j_1}, \mathsf{msk}_{j_1}, f; r_{j_1})$
   $\wedge (\mathsf{mpk}_{j_1}, \mathsf{msk}_{j_1}) = \mathsf{Setup}(1^\lambda; s_{j_1})$
   (AND)
2. The key sk_{f,j_2} for function f is generated using msk_{j_2} with r_{j_2} and related to mpk_{j_2} which is generated using s_{j_2}. Formally:
   $\mathsf{sk}_{f,j_2} = \mathsf{KeyGen}(\mathsf{mpk}_{j_2}, \mathsf{msk}_{j_2}, f; r_{j_2})$
   $\wedge (\mathsf{mpk}_{j_2}, \mathsf{msk}_{j_2}) = \mathsf{Setup}(1^\lambda; s_{j_2})$

Fig. 9: Relation used in the setup consistency compiler

(master key or functional key) serve as the witness). The decryption procedure then computes the decryption under all of the three instances and outputs the majority of the decryptions. If no majority is reached, the algorithm outputs $\diamond$. We give a formal description of this compiler in Fig. 8. This compiler only preserves CPA and, under certain conditions, also CFE security. Afterwards we also present a compiler that achieves CCA security by relying on a CPA secure scheme.

The proof of the following theorem can be found in Section G-A.

Theorem 5. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a functional encryption scheme and* $\mathsf{NIWI} = (\mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$ *a NIWI proof system for* R_{set} *(Fig. 9), then the construction* $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ *defined in Figure 8 is setup consistent. Namely, for any PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{B}$ *such that:*

$$|\Pr[\text{set-CONS}^{\mathsf{FE}'}(1^\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{Sound}}_{\mathsf{NIWI},\mathcal{B}}(\lambda).$$

We again show that the security of the underlying functional encryption scheme is preserved. We refer to Section G-B for these proofs.

2) Second Advanced Compiler: For the advanced setup consistency compiler that takes a CPA secure scheme and achieves CCA security, we proceed in a similar way as in the input consistency case. This is possible since security (in the sense of confidentiality) is only required w.r.t. an honest setup generator. Therefore, as long as the stronger tools required by the Naor-Yung approach [55] do smoothly integrate and not interfere with the tools needed to obtain setup consistency as of Theorem 5, we can follow a similar path, but have to pay attention to the details regarding the interplay of the three FE instances. Due to space constraints, we refer the reader to Section G-C for the full treatment.

References

[1] M. Abdalla, M. Bellare, and G. Neven. Robust encryption. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 480–497. Springer, Heidelberg, Feb. 2010.

[2] M. Abdalla, F. Benhamouda, and R. Gay. From single-input to multi-client inner-product functional encryption. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 552–582. Springer, Heidelberg, Dec. 2019.

[3] M. Abdalla, F. Benhamouda, M. Kohlweiss, and H. Waldner. Decentralizing inner-product functional encryption. In D. Lin and K. Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 128–157. Springer, Heidelberg, Apr. 2019.

[4] M. Abdalla, R. Gay, M. Raykova, and H. Wee. Multi-input inner-product functional encryption from pairings. In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 601–626. Springer, Heidelberg, Apr. / May 2017.

[5] S. Agrawal, S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption: New perspectives and lower bounds. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 500–518. Springer, Heidelberg, Aug. 2013.

[6] S. Agrawal, V. Koppula, and B. Waters. Impossibility of simulation secure functional encryption even with random oracles. In A. Beimel and S. Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 659–688. Springer, Heidelberg, Nov. 2018.

[7] S. Agrawal, B. Libert, and D. Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, Aug. 2016.

[8] P. Ananth, D. Boneh, S. Garg, A. Sahai, and M. Zhandry. Differing-inputs obfuscation and applications. Cryptology ePrint Archive, Report 2013/689, 2013. https://eprint.iacr.org/2013/689.

[9] P. Ananth and A. Jain. Indistinguishability obfuscation from compact functional encryption. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Heidelberg, Aug. 2015.

[10] B. Auerbach, M. Bellare, and E. Kiltz. Public-key encryption resistant to parameter subversion and its realization from efficiently-embeddable groups. In M. Abdalla and R. Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 348–377. Springer, Heidelberg, Mar. 2018.

[11] S. Badrinarayanan, V. Goyal, A. Jain, and A. Sahai. Verifiable functional encryption. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 557–587. Springer, Heidelberg, Dec. 2016.

[12] S. Badrinarayanan, D. Gupta, A. Jain, and A. Sahai. Multi-input functional encryption for unbounded arity functions. In T. Iwata and J. H. Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 27–51. Springer, Heidelberg, Nov. / Dec. 2015.

[13] B. Barak, S. J. Ong, and S. P. Vadhan. Derandomization in cryptography. *SIAM J. Comput.*, 37(2):380–400, 2007.

[14] M. Barbosa and P. Farshim. On the semantic security of functional encryption schemes. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 143–161. Springer, Heidelberg, Feb. / Mar. 2013.

[15] M. Bellare, G. Fuchsbauer, and A. Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, Dec. 2016.

[16] M. Bellare and A. O'Neill. Semantically-secure functional encryption: Possibility results, impossibility results and the quest for a general definition. In M. Abdalla, C. Nita-Rotaru, and R. Dahab, editors, *CANS 13*, volume 8257 of *LNCS*, pages 218–234. Springer, Heidelberg, Nov. 2013.

[17] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In S. Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 37–56. Springer, Heidelberg, Aug. 1990.

[18] F. Benhamouda, F. Bourse, and H. Lipmaa. CCA-secure inner-product functional encryption from projective hash functions. In S. Fehr, editor, *PKC 2017, Part II*, volume 10175 of *LNCS*, pages 36–66. Springer, Heidelberg, Mar. 2017.

[19] D. J. Bernstein and T. Lange. Computing small discrete logarithms faster. In S. D. Galbraith and M. Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 317–338. Springer, Heidelberg, Dec. 2012.

[20] N. Bitansky and O. Paneth. ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In Y. Dodis and J. B. Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 401–427. Springer, Heidelberg, Mar. 2015.

[21] D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Heidelberg, Aug. 2001.

[22] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, Mar. 2011.

[23] E. Boyle, K.-M. Chung, and R. Pass. On extractability obfuscation. In Y. Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 52–73. Springer, Heidelberg, Feb. 2014.

[24] Z. Brakerski, I. Komargodski, and G. Segev. Multi-input functional encryption in the private-key setting: Stronger security from weaker assumptions. *Journal of Cryptology*, 31(2):434–520, Apr. 2018.

[25] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.

[26] P. Chaidos and G. Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 193–221. Springer, Heidelberg, Apr. / May 2018.

[27] Y. Chen, V. Vaikuntanathan, B. Waters, H. Wee, and D. Wichs. Traitor-tracing from LWE made simple and attribute-based. In A. Beimel and S. Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 341–369. Springer, Heidelberg, Nov. 2018.

[28] J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Decentralized multi-client functional encryption for inner product. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 703–732. Springer, Heidelberg, Dec. 2018.

[29] J. Chotard, E. Dufour Sans, R. Gay, D. H. Phan, and D. Pointcheval. Multi-client functional encryption with repetition for inner product. Cryptology ePrint Archive, Report 2018/1021, 2018. https://eprint.iacr.org/2018/1021.

[30] G. Couteau and D. Hofheinz. Designated-verifier pseudorandom generators, and their applications. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 562–592. Springer, Heidelberg, May 2019.

[31] I. Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In J. Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 445–456. Springer, Heidelberg, Aug. 1992.

[32] 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 R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 519–535. Springer, Heidelberg, Aug. 2013.

[33] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM Journal on Computing*, 30(2):391–437, 2000.

[34] A. Escala, G. Herold, E. Kiltz, C. Ràfols, and J. Villar. An algebraic framework for Diffie-Hellman assumptions. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Heidelberg, Aug. 2013.

[35] P. Farshim, B. Libert, K. G. Paterson, and E. A. Quaglia. Robust encryption, revisited. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 352–368. Springer, Heidelberg, Feb. / Mar. 2013.

[36] L. Fortnow. The complexity of perfect zero-knowledge (extended abstract). In A. Aho, editor, *19th ACM STOC*, pages 204–209. ACM Press, May 1987.

[37] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, Oct. 2013.

[38] S. Garg, C. Gentry, S. Halevi, and M. Zhandry. Functional encryption without obfuscation. In E. Kushilevitz and T. Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 480–511. Springer, Heidelberg, Jan. 2016.

[39] R. Géraud, D. Naccache, and R. Rosie. Robust encryption, extended. In M. Matsui, editor, *CT-RSA 2019*, volume 11405 of *LNCS*, pages 149–168. Springer, Heidelberg, Mar. 2019.

[40] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest

majority. In A. Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[41] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F.-H. Liu, A. Sahai, E. Shi, and H.-S. Zhou. Multi-input functional encryption. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Heidelberg, May 2014.

[42] S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In D. Boneh, T. Roughgarden, and J. Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[43] S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Heidelberg, Aug. 2012.

[44] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In A. Juels, R. N. Wright, and S. De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, Oct. / Nov. 2006. Available as Cryptology ePrint Archive Report 2006/309.

[45] J. Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In X. Lai and K. Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, pages 444–459. Springer, Heidelberg, Dec. 2006.

[46] J. Groth, R. Ostrovsky, and A. Sahai. Non-interactive zaps and new techniques for NIZK. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Heidelberg, Aug. 2006.

[47] V. Iovino and K. Zebrowski. Simulation-based secure functional encryption in the random oracle model. In K. E. Lauter and F. Rodríguez-Henríquez, editors, *LATINCRYPT 2015*, volume 9230 of *LNCS*, pages 21–39. Springer, Heidelberg, Aug. 2015.

[48] S. Katsumata, R. Nishimaki, S. Yamada, and T. Yamakawa. Designated verifier/prover and preprocessing NIZKs from Diffie-Hellman assumptions. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 622–651. Springer, Heidelberg, May 2019.

[49] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 146–162. Springer, Heidelberg, Apr. 2008.

[50] A. B. Lewko and B. Waters. Decentralizing attribute-based encryption. In K. G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 568–588. Springer, Heidelberg, May 2011.

[51] Y. Lindell. A simpler construction of cca2-secure public-key encryption under general assumptions. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 241–254. Springer, Heidelberg, May 2003.

[52] C. Matt and U. Maurer. A definitional framework for functional encryption. In C. Fournet and M. Hicks, editors, *CSF 2015Computer Security Foundations Symposium*, pages 217–231. IEEE Computer Society Press, 2015.

[53] U. Maurer. Constructive cryptography – a new paradigm for security definitions and proofs. In S. Moedersheim and C. Palamidessi, editors, *Theory of Security and Applications (TOSCA 2011)*, volume 6993 of *Lecture Notes in Computer Science*, pages 33–56. Springer-Verlag, 4 2011.

[54] M. Naor. On cryptographic assumptions and challenges (invited talk). In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 96–109. Springer, Heidelberg, Aug. 2003.

[55] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd ACM STOC*, pages 427–437. ACM Press, May 1990.

[56] A. O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. https://eprint.iacr.org/2010/556.

[57] C. Peikert and S. Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, Aug. 2019.

[58] J. M. Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of Cryptology*, 13(4):437–447, Sept. 2000.

[59] W. Quach, R. D. Rothblum, and D. Wichs. Reusable designated-verifier NIZKs for all NP from CDH. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 593–621. Springer, Heidelberg, May 2019.

[60] R. D. Rothblum, A. Sealfon, and K. Sotiraki. Towards non-interactive zero-knowledge for NP from LWE. In D. Lin and K. Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 472–503. Springer, Heidelberg, Apr. 2019.

[61] A. Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, Oct. 1999.

[62] A. Sahai and B. R. Waters. Fuzzy identity-based encryption. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Heidelberg, May 2005.

[63] N. Soroush, V. Iovino, A. Rial, P. B. Rønne, and P. Y. A. Ryan. Verifiable inner product encryption scheme. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 65–94. Springer, Heidelberg, May 2020.

Appendix A
Continued Preliminaries

A. Security Definitions

Definition 12 (CPA & CCA Security of FE)**.** *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a functional encryption scheme,* $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ *a function family and* $\beta \in \{0,1\}$*. We define the experiments* $\text{IND-CPA}^{\mathsf{FE}}_\beta(1^\lambda, \mathcal{A})$ *and* $\text{IND-CCA}^{\mathsf{FE}}_\beta(1^\lambda, \mathcal{A})$ *in Fig. 10. The associated advantage of an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *for* $\text{XX} \in \{\text{CPA}, \text{CCA}\}$ *is defined by*

$$\mathsf{Adv}^{\text{IND-XX}}_{\mathsf{FE},\mathcal{A}}(\lambda) = |\Pr[\text{IND-XX}^{\mathsf{FE}}_0(1^\lambda, \mathcal{A}) = 1] - \Pr[\text{IND-XX}^{\mathsf{FE}}_1(1^\lambda, \mathcal{A}) = 1]|.$$

An adversary $\mathcal{A}$ *is* valid *if for the two submitted challenges* x^0 *and* x^1 *and all keys* sk_f *the attacker obtained for* f *via calls to* KeyGen *(and including the empty key for* f_0*), it holds that* $f(x^0) = f(x^1)$*. For CCA security, the adversary* $\mathcal{A}$ *is additionally not allowed to query the decryption oracle* $\mathsf{QDec}(f, \mathsf{ct})$ *on the challenge ciphertext* $\mathsf{ct} = \mathsf{Enc}(\mathsf{mpk}, x^\beta)$*.*

A functional encryption scheme FE *is* IND-XX *secure, if for any valid PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a negligible function* negl*, such that* $\mathsf{Adv}^{\text{IND-XX}}_{\mathsf{FE},\mathcal{A}}(\lambda) \le \text{negl}(\lambda)$*.*

$\text{IND-CPA}^{\mathsf{FE}}_\beta(1^\lambda, \mathcal{A})$

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$

$(x^0, x^1, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{mpk},\mathsf{msk},\cdot)}(\mathsf{mpk})$

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^\beta)$

$\alpha \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{mpk},\mathsf{msk},\cdot)}(\mathsf{mpk}, \mathsf{ct}, \mathsf{st})$

Output: α

$\text{IND-CCA}^{\mathsf{FE}}_\beta(1^\lambda, \mathcal{A})$

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$

$(x^0, x^1, \mathsf{st}) \leftarrow \mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{mpk},\mathsf{msk},\cdot),\mathsf{QDec}(\cdot,\cdot)}(\mathsf{mpk})$

$\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^\beta)$

$\alpha \leftarrow \mathcal{A}_2^{\mathsf{KeyGen}(\mathsf{mpk},\mathsf{msk},\cdot),\mathsf{QDec}(\cdot,\cdot)}(\mathsf{mpk}, \mathsf{ct}, \mathsf{st})$

Output: α

Fig. 10: IND-CPA and IND-CCA security for functional encryption. The decryption oracle $\mathsf{QDec}(f, \mathsf{ct})$ in the CCA game generates secret key $\mathsf{sk}_f = \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$ and outputs $\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}_f, \mathsf{ct})$ for query (f, ct).

Beside the game based security definitions, we also recap a simulation based definition, composable functional encryption (CFE), introduced by Matt and Maurer in [52]. The notion of composable functional encryption (CFE) security.

$\text{Real}^{\mathsf{FE}}(1^\lambda, \mathcal{A})$

$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$

$(\ell, \tau) \leftarrow (0, 0)$

Repeat

$\ell \leftarrow \ell + 1$

$x_\ell \leftarrow \mathcal{A}_1^{\mathsf{KeyGen}(\mathsf{mpk},\mathsf{msk},\cdot)}(\mathsf{mpk})[[\tau]]$

$\mathsf{ct}_\ell \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_\ell)$

$t \leftarrow \mathcal{A}_2(\mathsf{ct}_\ell)[[\tau]]$

Until $t = \mathsf{true}$

Output: τ

$\text{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$

$(\mathsf{mpk}, s) \leftarrow \mathcal{S}_1(1^\lambda)$

$(\ell, \tau) \leftarrow (0, 0)$

Repeat

$\ell \leftarrow \ell + 1$

$x_\ell \leftarrow \mathcal{A}_1^{\mathcal{O}(\cdot, x_1, \ldots, x_{\ell-1})[[s]]}(\mathsf{mpk})[[\tau]]$

$(f_1, \ldots, f_q) \leftarrow$ queries by $\mathcal{A}_1$

$\mathsf{ct}_\ell \leftarrow \mathcal{S}_3(f_0(x_\ell), \ldots, f_q(x_\ell))[[s]]$

$t \leftarrow \mathcal{A}_2(\mathsf{ct}_\ell)[[\tau]]$

Until $t = \mathsf{true}$

Output: τ

Fig. 11: CFE security definition

Definition 13 (Composable Functional Encryption Security)**.** *Let* FE *be a functional encryption scheme,* $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ *a function family, define the experiments* $\text{Real}^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ *and* $\text{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$ *with a PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *and a PPT simulator* $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *respectively in Fig. 11, where the oracle* $\mathcal{O}$ *is defined as*

$$\mathcal{O}(f, x_1, \ldots, x_{\ell-1})[[s]] := \mathcal{S}_2(f, f(x_1), \ldots, f(x_{\ell-2}))[[s]] \ .$$

The advantage of the experiments is defined by:

$$\mathsf{Adv}^{\mathcal{D},\text{CFE}}_{\mathsf{FE},\mathcal{A},\mathcal{S}}(\lambda) = |\Pr[\mathcal{D}(\text{Real}^{\mathsf{FE}}(1^\lambda, \mathcal{A})) = 1] - \Pr[\mathcal{D}(\text{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})) = 1| \ ,$$

where $\mathcal{D}$ *is a PPT distinguisher.*

A functional encryption scheme FE *is* CFE *secure, if there exists a PPT simulator* $\mathcal{S}$*, such that for any PPT distinguisher* $\mathcal{D}$ *it holds that* $\mathsf{Adv}^{\mathcal{D},\text{CFE}}_{\mathsf{FE},\mathcal{A},\mathcal{S}}(\lambda) \le \text{negl}(\lambda)$ *for any PPT adversary* $\mathcal{A}$*, where* $\text{negl}(\cdot)$ *is a negligible function.*

Remark 2 (On the leakage function). As already noted in [52], the leakage function is a modeling artifact specific to the confidentiality definitions: the information captured by f_0 models the general leakage that *might* be possible to compute by an adversary by just observing an *honestly generated* ciphertext, for example the length of the underlying plaintext (which some works put in place by

default). Because this information is not guaranteed to be computable f_0 does actually not model a real function as opposed to $f_i, i > 0$. As we will see later, our consistency guarantees will only require that the guaranteed functions $f_i, i > 0$ yield consistent results.

B. Non-interactive Proofs

Now, we recapture the definition of non-interactive zero knowledge (NIZK) proofs [17], [36], [40] and non-interactive witness indistinguishable (NIWI) proofs [13], [20], [46].

Definition 14 (Non-Interactive Zero-Knowledge Proofs)**.** *Let R be an NP Relation and consider the language $L = \{x \mid \exists w \text{ with } (x,w) \in R\}$ (where x is called a statement or instance). A non-interactive zero-knowledge proof (NIZK) for the relation R is a triple of PPT algorithms* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$*:*

$\mathsf{NIZK.Setup}(1^\lambda)$**:** *Takes as input a security parameter λ and outputs the common reference string* CRS*.*

$\mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$**:** *Takes as input the common reference string* CRS*, a statement x and a witness w, and outputs a proof π.*

$\mathsf{NIZK.Verify}(\mathsf{CRS}, x, \pi)$**:** *Takes as input the common reference string* CRS*, a statement x and a proof π, and outputs 0 or 1.*

A system NIZK *is complete, if (for all $\lambda \in \mathbb{N}$), for all* CRS *in the support of* $\mathsf{Setup}(1^\lambda)$ *and all statement-witness pairs in the relation $(x,w) \in R$, it holds that*

$$\Pr[\mathsf{NIZK.Verify}(\mathsf{CRS}, x, \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)) = 1] = 1.$$

Besides completeness, a NIZK system also fulfills the notion of soundness and zero-knowledge, which we introduce in the following two definitions:

Definition 15 (Soundness)**.** *Given a proof system* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *for a relation R and the corresponding language L, we define the soundness advantage of an adversary $\mathcal{A}$ as the probability:*

$$\begin{aligned}\mathsf{Adv}^{\text{Sound}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) := \Pr[&\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda);\\ &(x,\pi) \leftarrow \mathcal{A}(\mathsf{CRS}) :\\ &\mathsf{NIZK.Verify}(\mathsf{CRS}, x, \pi) = 1 \wedge x \notin L].\end{aligned}$$

A NIZK proof system is called perfectly sound if $\mathsf{Adv}^{\text{Sound}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) = 0$ *for all algorithms $\mathcal{A}$, and computationally sound, if* $\mathsf{Adv}^{\text{Sound}}_{\mathsf{NIZK},\mathcal{A}}(\lambda) \leq \text{negl}(\lambda)$ *for all PPT algorithms $\mathcal{A}$.*

Definition 16 (Zero-Knowledge)**.** *Let* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *be a NIZK proof system for a relation R and the corresponding language L, $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ a pair of algorithms (the simulator), with $\mathcal{S}'(\mathsf{CRS}, \tau, x, w) = \mathcal{S}_2(\mathsf{CRS}, \tau, x)$ for $(x,w) \in R$, and $\mathcal{S}'(\mathsf{CRS}, \tau, x, w) = \mathsf{failure}$ for $(x,w) \notin R$. For $\beta \in \{0,1\}$, we define the experiment $\mathrm{ZK}^{\mathsf{NIZK}}_\beta(1^\lambda, \mathcal{A})$ in Fig. 12. The associated advantage of an adversary $\mathcal{A}$ is defined as*

$\mathrm{ZK}^{\mathsf{NIZK}}_0(1^\lambda, \mathcal{A}, \mathcal{S})$	$\mathrm{ZK}^{\mathsf{NIZK}}_1(1^\lambda, \mathcal{A}, \mathcal{S})$
$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$	$(\mathsf{CRS}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$
$\alpha \leftarrow \mathcal{A}^{\mathsf{NIZK.Prove}(\mathsf{CRS},\cdot,\cdot)}(\mathsf{CRS})$	$\alpha \leftarrow \mathcal{A}^{\mathcal{S}'(\mathsf{CRS},\tau,\cdot,\cdot)}(\mathsf{CRS})$
Output: α	**Output:** α

Fig. 12: Zero-knowledge property of a NIZK proof system.

$$\begin{aligned}\mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda) := |\Pr[&\mathrm{ZK}^{\mathsf{NIZK}}_0(1^\lambda, \mathcal{A}, \mathcal{S}) = 1]\\ &- \Pr[\mathrm{ZK}^{\mathsf{NIZK}}_1(1^\lambda, \mathcal{A}, \mathcal{S}) = 1]|.\end{aligned}$$

A NIZK proof system NIZK *is called perfect zero-knowledge, with respect to a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, if* $\mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda) = 0$ *for all algorithms $\mathcal{A}$, and computationally zero-knowledge, if* $\mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda) \leq \text{negl}(\lambda)$ *for all PPT algorithms $\mathcal{A}$.*

Furthermore, we say that a NIZK is one-time simulation-sound [61], if the following holds.

Definition 17 (One-Time Simulation-Soundness)**.** *Given a proof system* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *for an NP relation R with corresponding language L and a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, we define the simulation-soundness advantage of an algorithm $\mathcal{A}$ by*

$$\begin{aligned}\mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda) := \Pr[&(\mathsf{CRS}, \tau) \leftarrow \mathcal{S}_1(1^\lambda);\\ &(x,\pi) \leftarrow \mathcal{A}^{\mathcal{S}_2(\mathsf{CRS},\tau,\cdot)}(\mathsf{CRS}) :\\ &(x,\pi) \notin Q \text{ and } x \notin L\\ &\text{and } \mathsf{NIZK.Verify}(\mathsf{CRS}, x, \pi) = 1] \ ,\end{aligned}$$

where Q is the set of all (x', π'), such that $\mathcal{A}$ queried x' to its oracle and π' is the matching response.

A NIZK proof system is called one-time simulation sound with respect to the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, if $\mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{A},\mathcal{S}}(\lambda) \leq \text{negl}(\lambda)$ *for all PPT algorithms $\mathcal{A}$ that make at most one query to oracle $\mathcal{S}_2$.*

C. Verifiable Functional Encryption

Now, we recap the definition of verifiable functional encryption as stated in [11].

Definition 18 (Verifiable Functional Encryption)**.** *A verifiable functional encryption scheme* $\mathsf{VFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{VerifyCT}, \mathsf{VerifySK})$ *extends a functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *by two algorithms* $\mathsf{VerifyCT}$ *and* $\mathsf{VerifySK}$ *which have the following behavior:*

$\mathsf{VerifyCT}(\mathsf{mpk}, \mathsf{ct})$**:** *Takes as input the master public key* mpk *and a ciphertext* ct *and outputs 1 if the ciphertext* ct *was correctly generated using the master public key* mpk *for some message x.*

$\mathsf{VerifySK}(\mathsf{mpk}, f, \mathsf{sk})$**:** *Takes as input the master public key* mpk*, a function f and a functional key* sk *and outputs*

1 *if the functional key* sk *was correctly generated as a functional key for the function* f.

Beside the correctness and security definition, a verifiable functional encryption scheme also needs to fulfill verifiability:

Definition 19 (Verifiability)**.** *A verifiable functional encryption scheme* VFE *for* $\mathcal{F}$ *is verifiable if, for all* $\mathsf{mpk} \in \{0,1\}^*$*, for all* $\mathsf{ct} \in \{0,1\}^*$*, there exists* $x \in \mathcal{X}$ *such that for all* $f \in \mathcal{F}$ *and* $\mathsf{sk} \in \{0,1\}^*$*, the following implication holds:*

$$\textit{If } \mathsf{VerifyCT}(\mathsf{mpk},\mathsf{ct}) = 1 \textit{ and } \mathsf{VerifySK}(\mathsf{mpk}, f, \mathsf{sk}) = 1$$

then

$$\Pr[\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}, \mathsf{ct}) = f(x)] = 1$$

D. Overview of the UC Framework

We use the universal composability (UC) framework introduced by Canetti [25] and provide a brief overview in this section. The goal of the UC framework is to capture what it means for a protocol to securely carry out a task. For this, we need to describe an ideal process and prove that no (efficient) environment can distinguish the real process and the ideal process, where the real-process is an execution of the protocol. Ideal processes are typically captured by ideal functionalities, which can be thought of as an incorruptible machine providing capabilities to different parties. These guarantees can depend on the corruption status of the parties in the system.

a) Protocol and protocol instances: Formally, a protocol π is an algorithm for a distributed system and formalized as an interactive Turing machine. An ITM has several tapes, for example an identity tape (read-only), an activation tape, or input/output tapes to pass values to its program and return values back to the caller. A machine also has a backdoor tape where (especially in the case of ideal functionalities) interaction with an adversary is possible or corruption messages are handled. While an ITM is a static object, UC defines the notion of an ITM instance (denoted ITI), which is defined by the extended identity $\mathrm{eid} = (M, id)$, where M is the description of an ITM and $id = (\mathrm{sid}, \mathrm{pid})$ is a string consisting of a session identifier sid and a party identifier $\mathrm{pid} \in \mathcal{P}$. An instance, also called a session, of a protocol π (represented as an ITM M_π) with respect to a session number sid is defined as a set of ITIs $\{(M_\pi, id_{\mathrm{pid}})\}_{\mathrm{pid}\in\mathcal{P}}$ where $id_{\mathrm{pid}} = (\mathrm{sid}, \mathrm{pid})$.

The real process can now be defined by an environment $\mathcal{Z}$ (a special ITI) that spawns exactly one session of the protocol in the presence of an adversary $\mathcal{A}$ (also a special ITI), where $\mathcal{A}$ is allowed to corrupt ITIs and gain their control. Which ITIs and in which form they can be corrupted is defined in a corruption model. In this work, we follow the static corruption model, which says that a party is either corrupted right from the beginning of the execution, or never. While static corruption is often needed when encryption schemes are involved, it also makes it possible to reason in a fine-grained fashion about the security of a system by looking at the specific set of corrupted parties. We note that this corruption set in the system is always known to the environment.

The output of the execution is the bit output by $\mathcal{Z}$ and is denoted by $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,r)$ where k is the security parameter, $z \in \{0,1\}^*$ is the input to the environment, and randomness r for the entire experiment. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)$ denote the random variable obtained by choosing the randomness r uniformly at random and evaluating $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,r)$. Let $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$.

b) Ideal-world process: The ideal process is formulated with respect to an ITM Func which is called an ideal functionality. In the ideal process, the environment $\mathcal{Z}$ interacts with Func, an ideal-world adversary (often called the simulator) $\mathcal{S}$ and a set of trivial, i.e., dummy ITMs representing the protocol machines that forward to the functionality whatever is provided as inputs to them by the environment (and return back whatever received from the functionality). In the ideal world, the ideal-world adversary (aka the simulator) can decide to corrupt parties. All corruptions are handled by the functionality which can assign more or less capabilities to the adversary depending on which parties are declared as corrupted in the system.

We denote the output of this ideal-world process by $\mathrm{EXEC}_{\mathsf{Func},\mathcal{A},\mathcal{Z}}(k,z,r)$ where the inputs are as in the real-world process. Let $\mathrm{EXEC}_{\mathsf{Func},\mathcal{S},\mathcal{Z}}(k,z)$ denote the random variable obtained by choosing the randomness r uniformly at random and evaluating $\mathrm{EXEC}_{\mathsf{Func},\mathcal{S},\mathcal{Z}}(k,z,r)$. Let $\mathrm{EXEC}_{\mathsf{Func},\mathcal{S},\mathcal{Z}}$ denote the ensemble $\{\mathrm{EXEC}_{\mathsf{Func},\mathcal{S},\mathcal{Z}}(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$.

c) Hybrid worlds: To model setup, the UC framework knows so-called hybrid worlds, which are worlds where the protocol under considerations invoke make use of ideal functionalities as subroutines (i.e., they invoke an ideal process as a subroutine). In this work, we use an authenticated repository and channel as assumed ideal functionalities.

d) Secure Realization and Composition: In a nutshell, a protocol securely realizes an ideal functionality Func if the real-world process (where the protocol is executed) is indistinguishable from the ideal-world process (relative to Func):

Definition 20. *Let us denote by* $\mathcal{X} = \{X(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *and* $\mathcal{Y} = \{Y(k,z)\}_{k\in\mathbb{N},z\in\{0,1\}^*}$ *two distribution ensembles over* $\{0,1\}$*. We say that* $\mathcal{X}$ *and* $\mathcal{Y}$ *are indistinguishable if for any* $c,d \in \mathbb{N}$ *there exists a* $k_0 \in \mathbb{N}$ *such that* $|\Pr[X(k,z)=1] - \Pr[Y(k,z)=1]| < k^{-c}$ *for all* $k > k_0$ *and all* $z \in \bigcup_{\kappa\le k^d}\{0,1\}^\kappa$*. We use the shorthand notation* $\mathcal{X} \approx \mathcal{Y}$ *to denote two indistinguishable ensembles.*

Definition 21. *Let* Func *be an ideal functionality and let* π

be a protocol. We say that π securely realizes Func *if for any (efficient) adversary $\mathcal{A}$ there exists an (efficient) ideal-world adversary (the simulator) $\mathcal{S}$ such that for every (efficient) environment $\mathcal{Z}$ it holds that $EXEC_{\pi,\mathcal{A},\mathcal{Z}} \approx EXEC_{\mathsf{Func},\mathcal{S},\mathcal{Z}}$, as defined above.*

Note that the definition in [25] allows to capture in a more fine-grained way the context in which a protocol is executed as a further condition on the environment. We do not need this in our work and the statement holds for all contexts. The realization notion is composable, which is roughly speaking the guarantee that whenever in a certain context, the ideal process is used (e.g. as setup in a hybrid world) then it can be replaced by the protocol realizing it.

Appendix B
Consistency for Different Types of Functional Encryption

In the present treatment we considered the standard public-key single-input FE setting. Nevertheless, our consistency notions can be relevant also when considering other FE settings. In particular it is easy to extend our treatment and results to give some meaningful guarantees for secret-key [8], [23], [37], [38], [42], [43] and multi-input/multi-client FE [9], [12], [24], [41]. It is worth noting that stronger (more tailored) notions of consistency might be conceivable in these cases, which depends on the intended applications. This may lead to a new modeling of these properties as an interesting further direction.

For secret-key FE, since the input provider and the setup generator is one party (e.g., consider medical record and a system that assigns different access rights to different doctors) strong input consistency seems to be the only reasonable formulation and is basically covered in the verifiable FE paper [11].

Regarding Multi-Client FE, our notion still ensures that it is not efficiently possible to output a ciphertext ct (now consisting of n components) such that ct would not be explainable by a vector of input values $(x_1, \ldots, x_n)$ given output values y_i (derived from ct) for functions f_i. However, a stronger notion could be derived and analyzed in the UC setting that captures consistency across the components of a ciphertext (while allowing also ciphertexts that can be "mixed"). In the Multi-client setting, our notion gives again similar guarantees as above where the master public key is a vector. In this setting, it must be impossible to generate a ciphertext that yields inconsistent output values in the sense that no input vector $(x_1, \ldots, x_n)$ would exist. As in the case of Multi-Client FE, it might be interesting to define stronger and more specific notions of consistency for this setting. Especially setup consistency remains important in the case of *Decentralized* Multi-Client FE [2], [3], [28], [29], where a part of Charlie's task, i.e., the functional key generation, is distributed among the clients. Although these schemes are proven with respect to a passive adversary, as soon as moving to the active case, consistency, as we define it in this work, is needed.

Appendix C
Relations (in)between Consistency and Security

If not otherwise quantified, we denote by $\mathcal{F}$ a functionality class, the members of $\mathcal{F}$ by f_i, and refer to the number of functions (not counting the distinguished leakage function f_0) as the size s of the functionality class.

A. Relations among the Consistency Notions

Let us first summarize the relations between the notions which can all be seen by simple arguments: strong input consistency implies input consistency since the attack model of input consistency is a strict subset of strong input consistency. Furthermore, since the schemes we present in Section V are input consistent but neither strong input consistent nor setup consistent. The only remaining non-implications are that strong input consistency does not imply setup consistency and that setup consistency does not imply strong or normal input consistency. Formally, both are easy to see: one can always take an input or strong-input consistent scheme and introduce a special master public key mpk' (that has probability zero of being generated by setup) which takes all messages to special ciphertext $\bar{\mathsf{ct}}$ that decrypts to $\perp$. Such a scheme is obviously not setup consistent but remains consistent because ct decrypts consistently. Along the same lines, one can introduce a new special ciphertext $\bar{\mathsf{ct}}$ in a setup consistent scheme, that decrypts to inconsistent outputs but clearly has probability 0 to be output by the encryption algorithm. This scheme remains setup consistent but is clearly not input consistent.

B. Consistency does not imply Confidentiality

To show that consistency does not imply confidentiality, we aim to construct a scheme that satisfies st-in-CONS but is not IND-CPA secure. The scheme is described in Fig. 13. It is easy to see that the scheme described in Fig. 13 does not provide any confidentiality guarantee since the ciphertext reveals the input message. We prove the consistency of the scheme more formally:

Theorem 6 (Strong input consistency)**.** *The functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *described in Fig. 13 is strongly input consistent for any functionality class $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$. Namely, for any PPT adversary $\mathcal{A}$, it holds that:*

$$\Pr[\text{st-in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A})] \leq \text{negl}(\lambda) \ .$$

Proof. After the challenger has received two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$ and some functional keys $\{\mathsf{sk}_{f_i}, f_i\}_{i\in[n]}$ from $\mathcal{A}$, it parses $\mathsf{ct}_1 = \tilde{x}_1, \mathsf{ct}_2 = \tilde{x}_2$ and sorts out the keys where $\mathsf{sk}_{f_i} \neq f_i \vee f_i \notin \mathcal{F}_\lambda$ as demanded by the decryption function. This results in the set $\{\mathsf{sk}_{f_i}, f_i\}_{i\in[m]}$ with $m \leq n$. In the next step, ciphertext validity is checked (for this scheme, this is just checking that x_i belongs to the domain) and

$\underline{\mathsf{Setup}(1^\lambda):}$
Return $(\mathsf{mpk},\mathsf{msk}) \leftarrow \{0,1\}^\lambda \times \{0,1\}^\lambda$
$\underline{\mathsf{KeyGen}(\mathsf{msk},f):}$
Return $\mathsf{sk}_f = f$
$\underline{\mathsf{Enc}(\mathsf{mpk},x):}$
Return $\mathsf{ct} = x$
$\underline{\mathsf{Dec}(\mathsf{mpk},f,\mathsf{sk}_f,\mathsf{ct}):}$
Parse $\mathsf{ct} \coloneqq x$
If $x \notin \mathcal{X}_\lambda$
  Return $\perp$
If $\mathsf{sk}_f \neq f \vee f \notin \mathcal{F}_\lambda$
  Return $\diamond$
Return $f(x)$

Fig. 13: A strongly input consistent FE scheme which is not IND-CPA secure.

$\underline{\mathsf{Setup}(1^\lambda):}$
For $i = 1,\ldots,s$ run $(\mathsf{pk}_i,\mathsf{sk}_i) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$
$(\mathsf{mpk},\mathsf{msk}) = (\{\mathsf{pk}_i\}_{i\in[s]},\{\mathsf{sk}_i\}_{i\in[s]})$
$\underline{\mathsf{KeyGen}(\mathsf{msk},f_i):}$
Parse $\mathsf{msk} \coloneqq \{\mathsf{sk}_i\}_{i\in[s]}$
Return $\mathsf{sk}_{f_i} = \mathsf{sk}_i$
$\underline{\mathsf{Enc}(\mathsf{mpk},x):}$
Parse $\mathsf{mpk} \coloneqq \{\mathsf{pk}_i\}_{i\in[s]}$
Compute $\mathsf{ct}_i = \mathsf{PKE.Enc}(\mathsf{pk}_i, f_i(x)), \forall i \in [s]$
Return $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[s]}$
$\underline{\mathsf{Dec}(\mathsf{mpk},\mathsf{sk}_{f_i},f_i,\mathsf{ct}):}$
Parse $\mathsf{mpk} \coloneqq \{\mathsf{pk}_i\}_{i\in[s]}, \mathsf{sk}_{f_i} \coloneqq \mathsf{sk}_i, \mathsf{ct} \coloneqq (\mathsf{ct}_i)_{i\in[s]}$
Return $y := \mathsf{PKE.Dec}(\mathsf{sk}_i,\mathsf{ct}_i)$

Fig. 14: An IND-CCA secure, but not consistent functional encryption scheme.

we distinguish between two cases. First, for each $j \in [2]$ s.t. $\tilde{x}_j \in \mathcal{X}_\lambda$ we have by definition

$$\begin{aligned}&\textstyle\bigcap_{i\in[m]} f_i^{-1}(\mathsf{Dec}(\mathsf{mpk},f_i,\mathsf{sk}_{f_i},\mathsf{ct}_j))\\ =&\textstyle\bigcap_{i\in[m]} f_i^{-1}(f_i(\tilde{x}_j)) \ni \tilde{x}_j \ .\end{aligned}$$

And for each $j \in [2]$ s.t. $\tilde{x}_j \notin \mathcal{X}_\lambda$ it holds that

$$\begin{aligned}&\textstyle\bigcap_{i\in[m]} f_i^{-1}(\mathsf{Dec}(\mathsf{mpk},f_i,\mathsf{sk}_{f_i},\mathsf{ct}_j))\\ =&\textstyle\bigcap_{i\in[m]} f_i^{-1}(\perp) = \perp \ .\end{aligned}$$

In both of these cases, the intersection remains non-empty and strong input-consistency follows. □

The scheme is trivially setup consistent since the encryptor ignores any setup values and Bob just evaluates the plain functions. Finally, input consistency follows, since it is implied by strong input consistency.

C. Confidentiality does not imply Consistency

Next, we prove that the strongest confidentiality notions in use, i.e., IND-CCA and CFE, do not imply consistency with respect to dishonest input provider or parameter generator.

1) The IND-CCA case: At first glance, the notions of IND-CCA security and input consistency seem to be related. In both games, the scheme must tame the adversaries capabilities of generating malicious ciphertexts. We show however that there is no connection between IND-CCA security and input or setup consistency, by presenting a scheme that is IND-CCA secure, but not input or setup consistent. The scheme is described in Fig. 14, it is based on the brute-force construction of [22, Section 4].

Theorem 7. *Let* $\mathsf{PKE} = (\mathsf{PKE.Setup},\mathsf{PKE.Enc},\mathsf{PKE.Dec})$ *be an IND-CCA secure public-key encryption scheme, then the functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup},\mathsf{KeyGen},\mathsf{Enc},\mathsf{Dec})$ *in Fig. 14 is IND-CCA secure for any functionality class* $\mathcal{F}$ *of polynomial size* s *(in the security parameter). Namely, for any PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{FE},\mathcal{A}}(\lambda) \leq s \cdot \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{PKE},\mathcal{B}}(\lambda) \ .$$

Proof. To prove this statement, we use a hybrid argument over the games $\mathsf{G}_0,\ldots,\mathsf{G}_s$ as defined in Fig. 15. Note that G_0 corresponds to the game $\text{IND-CCA}^{\mathsf{FE}}_0$ and game G_s to the game $\text{IND-CCA}^{\mathsf{FE}}_1$. By using the triangle inequality, we get:

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{FE}',\mathcal{A}}(\lambda) \leq \sum_{k=1}^{s} |\mathsf{Win}^{\mathsf{G}_{0.k-1}}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_{0.k}}_{\mathcal{A}}(1^\lambda)| \ .$$

We conclude the proof by showing that for any $k \in [s]$, there exists an adversary $\mathcal{B}_k$ such that:

$$|\mathsf{Win}^{\mathsf{G}_{0.k-1}}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_{0.k}}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{PKE},\mathcal{B}_k}(\lambda) \ .$$

The adversary $\mathcal{B}$ of the statement is then defined as the monolithic adversary that first samples $k \leftarrow [s]$ uniformly at random and then runs the code of $\mathcal{B}_k$.

We build an adversary $\mathcal{B}_k$ that simulates $\mathsf{G}_{0.k-1+\beta}$ to $\mathcal{A}$, when interacting with the underlying $\text{IND-CCA}^{\mathsf{PKE}}_\beta$ experiment.

In the first step of the reduction, the adversary $\mathcal{B}_k$ receives the public key pk from the experiment. It sets $\mathsf{pk}_k = \mathsf{pk}$ and generates public key instances $(\mathsf{pk}_i,\mathsf{sk}_i) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$ for all $i \in [s]\backslash\{k\}$, defines the master public key as $\mathsf{mpk} \coloneqq \{\mathsf{pk}_i\}_{i\in[s]}$ and sends it to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks for a functional key sk_{f_i}, with $i \in [s]\backslash\{k\}$, $\mathcal{B}_k$ outputs sk_i to $\mathcal{A}$. If $\mathcal{A}$ asks for the functional key sk_{f_k} the adversary $\mathcal{B}_k$ outputs a random value $\alpha \leftarrow \{0,1\}$

Game	ct	justification/ remark
G_0	$\mathsf{Enc}(\mathsf{mpk}, f_i(x^0))$, for all $i \in [s]$	$\mathsf{G}_0 = \mathsf{G}_{0.0}$
$\mathsf{G}_{0.k}$	$\mathsf{Enc}(\mathsf{mpk}, \boxed{f_i(x^1)})$, for all $i \leq k$ $\mathsf{Enc}(\mathsf{mpk}, f_i(x^0))$, for all $i > k$	IND-CCA of PKE
G_1	$\mathsf{Enc}(\mathsf{mpk}, \boxed{f_i(x^1)})$, for all $i \in [s]$	$\mathsf{G}_1 = \mathsf{G}_{0.s}$

Fig. 15: Overview of the games to prove the IND-CCA security of the functional encryption scheme described in Fig. 14.

as its guess. Note that by definition, $\mathcal{A}$ is in this case restricted to submit identical challenge messages w.r.t. the public key pk for which case the behavior of $\mathsf{G}_{0.k-1}$ and $\mathsf{G}_{0.k}$ are identical (and thus independent of β).

When $\mathcal{A}$ submits its challenge messages (x^0, x^1), the adversary $\mathcal{B}_k$ computes $\mathsf{ct}_i = \mathsf{PKE.Enc}(\mathsf{pk}_i, f_i(x^1))$ for all $i < k$ and $\mathsf{ct}_i = \mathsf{PKE.Enc}(\mathsf{pk}_i, f_i(x^0))$ for all $i > k$. To generate the ciphertext ct_k, $\mathcal{B}_k$ creates the challenge $(f_k(x^0), f_k(x^1))$ and submits it as its own challenge. $\mathcal{B}_k$ receives $\mathsf{ct}_k = \mathsf{PKE.Enc}(\mathsf{pk}_k, f_k(x^\beta))$ as an answer and creates the ciphertext $\mathsf{ct} := (\mathsf{ct}_i)_{i \in [s]}$, which it sends to $\mathcal{A}$.

If $\mathcal{A}$ queries the decryption oracle $\mathsf{QDec}(f_i, \mathsf{ct})$ with $i \in [s] \setminus \{k\}$, $\mathcal{B}_k$ computes $f_i(x) = \mathsf{PKE.Dec}(\mathsf{sk}_i, \mathsf{ct})$ and sends $f_i(x)$ to $\mathcal{A}$. In the case $\mathcal{A}$ queries $\mathsf{QDec}(f_k, \mathsf{ct})$, $\mathcal{B}_k$ forwards ct to its own decryption oracle and sends the reply to $\mathcal{A}$.

In the last step, the adversary $\mathcal{B}_k$ outputs the same bit β' returned by $\mathcal{A}$. Since $\mathcal{B}_k$ perfectly emulates $\mathsf{G}_{0.k-1+\beta}$ to $\mathcal{A}$ as long as the public key pk_k is not asked, and since for the latter exception case, the advantage of $\mathcal{A}$ is zero, $\mathcal{B}_k$'s distinguishing advantage in the CCA game is at least the advantage of $\mathcal{A}$ distinguishing systems $\mathsf{G}_{0.k-1+\beta}$, for $\beta \in \{0, 1\}$. □

After showing the IND-CCA security of the scheme, we describe a successful attacker for the input consistency game.

Theorem 8. *Let* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an IND-CCA secure public-key encryption scheme, then the functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *in Fig. 14 is not input consistent for a concrete functionality class* $\mathcal{F}$ *of size* $s = 2$ *(as described in the proof). Namely, there exists a PPT adversary* $\mathcal{A}$ *such that*

$$\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 1 \ .$$

Proof. We consider a functionality class that contains two functions $(s = 2)$, i.e. $\mathcal{F} = \{f_1, f_2\}$, with $f_1 : \mathcal{X}_\lambda \to \{0, 1\}$ and $f_2(x) := \overline{f_1(x)}$, where $\overline{\ }$ denotes the bit complement. The adversary $\mathcal{A}$ generates a ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2) = (\mathsf{Enc}(\mathsf{pk}_1, 0), \mathsf{Enc}(\mathsf{pk}_2, 0))$, asks the KeyGen oracle for the two secret keys and sends (ct, f_1, f_2) to the challenger.

We observe that both decryptions will yield $y_i = 0$ as an output. Therefore, we have obtain in any case $f_1^{-1}(0) \cap f_2^{-1}(0) = f_1^{-1}(0) \cap \overline{f_1^{-1}(0)} = \emptyset$, which contradicts the input-consistency requirement. □

The scheme described in Fig. 14 is also not setup consistent.

Theorem 9. *Let* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an IND-CCA secure public-key encryption scheme, then the functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *in Fig. 14 is not setup consistent for a concrete function class* $\mathcal{F}$ *of size* $s = 2$ *(as described in the proof). Namely, there exists a PPT adversary* $\mathcal{A}$ *such that*

$$\Pr[\text{set-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 1 \ .$$

Proof. Similarly to the proof of input consistency, we consider a functionality class that contains two functions $(s = 2)$, i.e. $\mathcal{F} = \{f_1, f_2\}$, with $f_1 : \mathcal{X}_\lambda \to \{0, 1\}$ and $f_2(x) := \overline{f_1(x)}$. The adversary $\mathcal{A}$ executes the setup algorithm $\mathsf{Setup}(1^\lambda)$ to receive $(\mathsf{mpk}, \mathsf{msk})$ and generates a functional key $\mathsf{sk}_{f_1} \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f_1)$. It chooses a message $x \leftarrow \mathcal{X}_\lambda$ and sends $(\mathsf{mpk}_1, \mathsf{mpk}_2, \mathsf{sk}_{f_1}, f_2, x, x)$ to the challenger, which uses mpk and x to compute $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x)$. In the next step, the challenger computes $\mathsf{Dec}(\mathsf{sk}_{f_1}, f_2, \mathsf{ct}) = f_1(x)$ (note that the scheme by design does not aim at verifying sk_{f_1} vs. f_2) but then then verification tests whether $f_1(x) = f_2(x)$. This check is always false, due to the definition of f_2 $(f_2(x) = \overline{f_1(x)} \neq f_2(x), \forall x \in \mathcal{X}_\lambda)$, and gives us the consistency attack. □

2) The CFE case: The analysis presented in the last section can be adapted to the case of CFE security. More precisely, we show that CFE security does not imply consistency, by presenting a scheme that is CFE secure but not consistent. The scheme is presented in Fig. 16, it is the CFE secure version of the brute force scheme as introduced in [22, Section 5] and further analyzed in [52].

Theorem 10. *Let* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an IND-CPA secure public-key encryption scheme, then the functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *in Fig. 16 is CFE secure in the random oracle model for* H *for any functionality class* $\mathcal{F}$ *of polynomial size* s *(in the security parameter). Namely, for any PPT adversary* $\mathcal{A}$ *there is a PPT simulator* $\mathcal{S}$ *such that*

$$\mathsf{Adv}^{\text{CFE}}_{\mathsf{FE},\mathcal{A},\mathcal{S}}(\lambda) \leq \text{negl}(\lambda) \ .$$

Proof. We refer to [52] for a security proof of the construction. □

We show that this scheme does not imply input consistency.

Theorem 11. *Let* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an IND-CPA secure public-key encryption scheme,*

then the functional encryption scheme $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *in Fig. 16 is not input consistent for a concrete functionality class of size* $s = 2$*. Namely, there exists a PPT adversary* $\mathcal{A}$ *such that*

$$\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 1 \ .$$

Proof. The attack described in the proof of Theorem 8 also applies here, since, for sk_{f_i}, the scheme in Fig. 16 still performs a simple decryption at position i and hence will produce inconsistent outputs. □

The scheme described in Fig. 16 is also not setup consistent.

Theorem 12. *Let* $\mathsf{PKE} = (\mathsf{PKE.Setup}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ *be an IND-CPA secure public-key encryption scheme, then the functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *in Fig. 16 is not setup consistent for a concrete functionality class of size* $s = 2$*. Namely, there exists a PPT adversary* $\mathcal{A}$ *such that*

$$\Pr[\text{set-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 1 \ .$$

Proof. The attack described in the proof of Theorem 8 still applies here, since the scheme in Fig. 16 does not verify the claim on the function to be decrypted (and simply takes the matching dimension). □

$\underline{\mathsf{Setup}(1^\lambda):}$
For $i = 1, \ldots, s$ run $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$
$(\mathsf{mpk}, \mathsf{msk}) = (\{\mathsf{pk}_i\}_{i\in[s]}, \{\mathsf{sk}_i\}_{i\in[s]})$
$\underline{\mathsf{KeyGen}(\mathsf{msk}, f_i):}$
Return $\mathsf{sk}_{f_i} = \mathsf{sk}_i$
$\underline{\mathsf{Enc}(\mathsf{mpk}, x):}$
Parse $\mathsf{mpk} := \{\mathsf{pk}_i\}_{i\in[n]}$
Sample $r_i \leftarrow \mathcal{Y}_\lambda$, for all $i \in [s]$
Compute $\mathsf{ct}_i = (\mathsf{Enc}(\mathsf{mpk}, r_i), \mathsf{H}(r_i) \oplus f_i(x)), \forall i \in [s]$
Return $\mathsf{ct} = (\mathsf{ct}_i)_{i\in[s]}$
$\underline{\mathsf{Dec}(\mathsf{sk}_{f_i}, f_i, (\mathsf{ct}_i)_{i\in[s]}):}$
Parse $\mathsf{ct}_i := (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$
Compute $r_i = \mathsf{PKE.Dec}(\mathsf{sk}_{f_i}, \mathsf{ct}_{i,1})$
Return $y := \mathsf{H}(r_i) \oplus \mathsf{ct}_{i,2}$

Fig. 16: A CFE secure but inconsistent functional encryption scheme.

D. Consistency does not amplify Confidentiality

To conclude the relationship graph in Fig. 5, we analyze if consistency allows to lift security, i.e., whether consistency coupled with IND-CPA would directly yield (any of the) IND-CCA or CFE security notions. Both of these results are answered in the negative in this section by showing that in general, malleability and consistency are not contradicting requirements as can bee seen by existing ordinary PKE schemes (cast as special cases of FE).

We provide an explicit proof of this insight for strong input consistency. For concreteness, let R be an (efficiently computable) map on the plaintext space and let $\mathsf{maul}^R_{\mathsf{FE}}$ be (an efficiently computable) map such that for all plaintexts x and public parameters mpk in the range of Setup, and for any fixed randomness r, $\mathsf{maul}_{\mathsf{FE}}(\mathsf{Enc}(\mathsf{mpk}, x; r), \mathsf{mpk}) = \mathsf{Enc}(\mathsf{mpk}, R(x); r')$ for some randomness string r'. We call the map R separating for a function $f \in \mathcal{F}$ (or f-separating for short), if the composed map $f \circ R : \mathcal{X} \to \mathcal{Y}$ is an injective map.

Theorem 13. *If a functional encryption scheme* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *for a functionality class* $\mathcal{F}$ *admits an efficiently computable map* $\mathsf{maul}^R_{\mathsf{FE}}$ *for a plaintext map* R *that is* f*-separating (as defined above) for a given* $f \in \mathcal{F}$ *then the scheme cannot be CCA-secure. Furthermore, such CPA-secure schemes and concrete functionality classes exist in the standard model (under computational assumptions) which satisfy strong input consistency but are neither CFE-secure nor CCA-secure.*

Proof. To prove the first part we construct a generic attack given the assumptions on R: the adversary does never invoke its oracle KeyGen, picks two challenges $x_0 \neq x_1$ of the same length and obtains the challenge ciphertext ct^β as the encryption of m^β. The adversary can now query the decryption oracle for function f on $\mathsf{maul}^R_{\mathsf{FE}}(\mathsf{ct}^\beta, \mathsf{mpk})$ to obtain the function value y'^β (by the perfect correctness of the scheme). Since R is f-separating, $y'^\beta = f(R(x^\beta)) \neq f(R(x^{1-\beta}))$ and thus β can be guessed perfectly.

To prove the second part of the scheme we cast the El Gamal encryption scheme as an FE scheme with function class $\mathcal{F} = \{\mathsf{id}, f_0\}$ which is therefore CPA-secure under DDH [22]: Let $G = \langle g \rangle$ be a prime-order group (for a prime $2^{\lambda-1} < q < 2^\lambda$) with generator g. More concretely, we let $(\mathsf{mpk}, \mathsf{msk}) \leftarrow (g^a, a)$ for a random exponent a; an encryption of x is defined as $(g^r, g^{ra}x)$ for a random exponent r; and finally, define $\mathsf{Dec}(\mathsf{mpk}, \mathsf{sk}_{\mathsf{id}} = msk = a, \mathsf{ct}' = (\mathsf{ct}'_0, \mathsf{ct}'_1))$ to return $\diamond$ if $g^a \neq \mathsf{mpk}$, and otherwise to return $x' \leftarrow \mathsf{ct}_1 \cdot (\mathsf{ct}'_0)^{-a}$. The scheme satisfies strong input consistency, since given the ciphertext and the public-private key-pair (g^a, a), the underlying message is committed to. Furthermore, the scheme is malleable and the mapping $R : x \mapsto c \cdot x$ for a constant c is an injective mapping which is separating the identity function $\mathsf{id} \in \mathcal{F}$. We conclude the proof of the second part of the theorem by observing that CFE security for this scheme is impossible by the impossibility result given in [52, Theorem 5.1]. □

Appendix D
Details on the Inner-Product Schemes of Section V

A. Proofs of Section V-B

We prove Theorem 2 by separating it into two Lemmas. First, in Lemma 1, we show that the scheme described

in Fig. 7 is input inconsistent for the functionality class $\mathcal{F}^{m,X,Y}$. Second, in Lemma 2, we show that the same scheme is input inconsistent for the functionality class $\mathcal{F}_p^m$.

Lemma 1. *The functional encryption scheme* FE *for the functionality class* $\mathcal{F}^{m,X,Y}$ *described in Fig. 7 is not input consistent. Namely, there exists a PPT adversary* $\mathcal{A}$ *such that:*

$$\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 1 \ .$$

Proof. For the computation of the final output in the decryption procedure, it is necessary to compute the discrete logarithm of $[\langle \boldsymbol{x}, \boldsymbol{y} \rangle]_g$. As described in [7], we assume that the computed inner product lies in an polynomial bounded interval $\{0, \ldots, L\}$, i.e. $\langle \boldsymbol{x}, \boldsymbol{y} \rangle \in \{0, \ldots, L\}$ with a known L. This ensures that the discrete logarithm computation can be performed in $\tilde{\mathcal{O}}(L^{1/2})$, using Pollard's kangaroo method [58] (or even $\tilde{\mathcal{O}}(L^{1/3})$, by precomputing a table of size $\tilde{\mathcal{O}}(L^{1/3})$ [19]). Due to correctness, we assume that for every encrypted vector $\boldsymbol{x}$, with $\|\boldsymbol{x}\|_\infty < X$, and every functional key corresponding to $\boldsymbol{y}$, with $\|\boldsymbol{y}\|_\infty < Y$, the decryption gives us the right output $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$. This results in the fact that L must be bigger than $m \cdot X \cdot Y$. In this case, the decryption procedure outputs $\perp$.

Now, we describe the behavior of an attacker $\mathcal{A}$ against the input consistency of the scheme. After the challenger has generated the parameters, $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and has sent mpk to the adversary, the adversary generates a ciphertext ct, by encrypting the vector $\boldsymbol{x} := (L+1) \cdot \boldsymbol{e}_1$ after the rules defined in the encryption procedure. In the next step, $\mathcal{A}$ queries the key generation oracle for the vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_m$, receives $\mathsf{sk}_{\boldsymbol{e}_1}$ and $\mathsf{sk}_{\boldsymbol{e}_m}$ as a reply and sends ct to the challenger.

After receiving ct, the challenger computes $y_1 = \mathsf{Dec}(\mathsf{mpk}, \boldsymbol{e}_1, \mathsf{sk}_{\boldsymbol{e}_1}, \mathsf{ct})$ and $y_2 = \mathsf{Dec}(\mathsf{mpk}, \boldsymbol{e}_m, \mathsf{sk}_{\boldsymbol{e}_m}, \mathsf{ct})$. We consider the computation of y_1 and y_2 in more detail. In the decryption $\mathsf{Dec}(\mathsf{mpk}, \boldsymbol{e}_1, \mathsf{sk}_{\boldsymbol{e}_1}, \mathsf{ct})$, the decryptor computes $g^{\langle (L+1) \cdot \boldsymbol{e}_1, \boldsymbol{e}_1 \rangle} = g^{L+1}$ and tries to perform the discrete logarithm computation. This computation fails, due to the fact that $L+1$ is not part of the bounded interval $\{0, \ldots, L\}$, therefore the procedure outputs $\perp$ (this argument can be made for any fixed bound L). For the decryption procedure $\mathsf{Dec}(\mathsf{mpk}, \boldsymbol{e}_m, \mathsf{sk}_{\boldsymbol{e}_m}, \mathsf{ct})$, the decryptor computes $g^{\langle (L+1) \cdot \boldsymbol{e}_1, \boldsymbol{e}_m \rangle} = g^0 = 1$, for which the discrete logarithm can be easily computed. This results in $y_1 := \perp$ and $y_2 = 0$.

For the consistency check, we need to compute the preimages of the two different encryptions, i.e. $f_{\boldsymbol{e}_1}^{-1}(\perp)$ and $f_{\boldsymbol{e}_m}^{-1}(0)$. The first preimage is defined as $f_{\boldsymbol{e}_1}^{-1}(\perp) = \{\perp\}$ (due to Definition 2) and the second preimage as $f_{\boldsymbol{e}_m}^{-1}(0) = \{\boldsymbol{x} \in \mathbb{Z}_X^m : \langle \boldsymbol{x}, \boldsymbol{e}_m \rangle = 0\} = \{\binom{\boldsymbol{x}}{0}, \text{ with } \boldsymbol{x} \in \mathbb{Z}_X^{m-1}\}$. For the final step in the consistency check, we compute the intersection of the two preimages $f_{\boldsymbol{e}_1}^{-1}(\perp) \cap f_{\boldsymbol{e}_m}^{-1}(0) = \{\perp\} \cap \{\binom{\boldsymbol{x}}{0}, \text{ with } \boldsymbol{x} \in \mathbb{Z}_X^{m-1}\} = \emptyset$

This results in a consistency attack and therefore proves the lemma. □

Lemma 2. *Let* FE *be the IND-CPA secure functional encryption scheme for the functionality class* $\mathcal{F}_p^m$*, with* p *prime, described in Fig. 7, then the scheme* FE *is not input consistent. Namely, there exists a PPT adversary* $\mathcal{A}$ *such that:*

$$\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1] = 1 \ .$$

Proof. The proof works in the same manner as for Lemma 1. The polynomial bound L for the discrete logarithm computation in the last step must be smaller than p, such that we can still find a value $L+1$ for which the described attack works. If this is not the case, and the decryption procedure still remains efficient, it is possible to compute the discrete logarithm of g^x for all $x \in \mathbb{Z}_p$ by letting the decryption algorithm perform the task on random group elements. This yields a contradiction against the MDDH assumption and therefore, due to the fact that the security of the scheme is based on MDDH, a contradiction against the IND-CPA security of the scheme. □

B. Proof of Theorem 3

Proof. To prove the input consistency of the scheme described in Fig. 7, we need to show that no matter what ciphertext an adversary generates there exists at least one underlying plaintext that explains the decryption behavior of ct under different functional keys $\mathsf{sk}_{\boldsymbol{y}_i}$ queried by the adversary $\mathcal{A}$ during the game. We prove this by relying on the algebraic properties of the groups for which the functional encryption scheme is defined. In more detail, we show that there exists always a solution for a linear equation system that is defined by the different inner product computations between the functional keys and the submitted ciphertext. The existence of a solution shows that there exists at least one plaintext that explains the functional decryption behavior.

Now, we describe how the game proceeds. In the first step, the challenger generates the master public key and the master secret key by executing the setup procedure $(\mathsf{mpk}, \mathsf{msk}) = ((\mathcal{G}, [\mathbf{A}]_g, [\mathbf{WA}]_g), \mathbf{W}) \leftarrow \mathsf{Setup}(1^\lambda)$. In the next step, the adversary $\mathcal{A}$ receives mpk and has access to a key generation oracle $\mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, \cdot)$. Whenever $\mathcal{A}$ queries the key generation oracle with a vector $\boldsymbol{y}_i$, the challenger generates $\mathsf{sk}_{\boldsymbol{y}_i} = \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, \boldsymbol{y}_i)$, adds $(\mathsf{sk}_{\boldsymbol{y}_i}, \boldsymbol{y}_i)$ to the list F and sends $\mathsf{sk}_{\boldsymbol{y}_i}$ to $\mathcal{A}$. At some point in the game, $\mathcal{A}$ sends ct to the challenger and the challenger computes $[z_i]_g := \mathsf{Dec}(\mathsf{mpk}, \boldsymbol{y}_i, \mathsf{sk}_{\boldsymbol{y}_i}, \mathsf{ct})$ for all $(\mathsf{sk}_{\boldsymbol{y}_i}, \boldsymbol{y}_i) \in F$. We consider how the decryption works in more detail and determine $[z_i]_g$ specifically corresponding to $\boldsymbol{y}_i$.

For the ciphertext, output by the adversary $\mathcal{A}$, we write $\mathsf{ct} = \binom{\boldsymbol{c}_0'}{\boldsymbol{c}_1'}$, with $\boldsymbol{c}_0' = [\boldsymbol{c}_0]_g \in \mathbb{G}^{k+1}$ and $\boldsymbol{c}_1' = [\boldsymbol{c}_1]_g \in \mathbb{G}^m$. To be more specific, we also write $\boldsymbol{c}_0' \in \mathbb{G}^{k+1}$ and $\boldsymbol{c}_1' \in \mathbb{G}^m$ as explicit group elements, i.e. $\boldsymbol{c}_0' := \begin{pmatrix} g^{c_{0,1}} \\ \vdots \\ g^{c_{0,k+1}} \end{pmatrix}$ and $\boldsymbol{c}_0' := \begin{pmatrix} g^{c_{1,1}} \\ \vdots \\ g^{c_{1,m}} \end{pmatrix}$ with the generator g, $\boldsymbol{c}_0 := (c_{0,1}, \ldots, c_{0,k+1}) \in \mathbb{Z}_p^{k+1}$ and $\boldsymbol{c}_1 := (c_{1,1}, \ldots, c_{1,m}) \in \mathbb{Z}_p^m$. To show that the

decryption procedure always decrypts to one underlying element, we compute the decryption procedure for an arbitrary honestly generated key $\mathsf{sk}_{\boldsymbol{y}}$. We denote the master secret key as $\mathbf{W} := \begin{pmatrix} | & & | \\ \boldsymbol{w}_1 & \cdots & \boldsymbol{w}_{k+1} \\ | & & | \end{pmatrix}$, and correspondingly $\mathbf{W}^\top := \begin{pmatrix} - & \boldsymbol{w}_1^\top & - \\ & \vdots & \\ - & \boldsymbol{w}_{k+1}^\top & - \end{pmatrix}$. Using the matrix description, the functional key is defined as $\mathbf{W}^\top \cdot \boldsymbol{y} = \begin{pmatrix} \langle \boldsymbol{w}_1^\top, \boldsymbol{y} \rangle \\ \vdots \\ \langle \boldsymbol{w}_{k+1}^\top, \boldsymbol{y} \rangle \end{pmatrix} \in \mathbb{Z}_p^{k+1}$. For the decryption, we need to compute two different components: $[\boldsymbol{c}_0^\top \mathsf{sk}_{\boldsymbol{y}}]_g$ and $[\boldsymbol{c}_1^\top \boldsymbol{y}]_g$.

First, we describe how to compute $[\boldsymbol{c}_0^\top \mathsf{sk}_{\boldsymbol{y}}]_g$: We exponentiate all of the components of $\boldsymbol{c}_1'$ with the components of $\mathsf{sk}_{\boldsymbol{y}}$ and compute the product of the resulting vector components. In more detail, we compute $\prod_{i\in[k+1]} g^{c_{0,i}\cdot\langle \boldsymbol{w}_i^\top, \boldsymbol{y}\rangle} = g^{\sum_{i\in[k+1]}\langle c_{0,i}\cdot \boldsymbol{w}_i^\top, \boldsymbol{y}\rangle} = g^{\langle \sum_{i\in[k+1]} c_{0,i}\cdot \boldsymbol{w}_i^\top, \boldsymbol{y}\rangle} = g^{\langle \mathbf{W}\cdot \boldsymbol{c}_0, \boldsymbol{y}\rangle}$. We proceed in the same way for the second component $\prod_{i\in[m]} g^{c_{1,i}\cdot y_i} = g^{\sum_{i\in[m]} c_{1,i}\cdot y_i} = g^{\langle \boldsymbol{c}_1, \boldsymbol{y}\rangle}$.

For the final computation, before the discrete logarithm computation, we need to multiply the two components, which results in $g^{\langle \mathbf{W}\cdot\boldsymbol{c}_0, \boldsymbol{y}\rangle} \cdot g^{\langle \boldsymbol{c}_1, \boldsymbol{y}\rangle} = g^{\langle \mathbf{W}\cdot\boldsymbol{c}_0, \boldsymbol{y}\rangle + \langle \boldsymbol{c}_1, \boldsymbol{y}\rangle} = g^{\langle \mathbf{W}\cdot\boldsymbol{c}_0 + \boldsymbol{c}_1, \boldsymbol{y}\rangle}$.

In the final step of the decryption procedure the discrete logarithm computation happens. We denote the final decryptions with respect to the different $\boldsymbol{y_i}$'s by $z_i := \log(g^{\langle \mathbf{W}\cdot\boldsymbol{c}_0 + \boldsymbol{c}_1, \boldsymbol{y}\rangle})$.

To prove the input consistency, we show that the preimage of z_i contains the value $\mathbf{W}\cdot\boldsymbol{c}_0 + \boldsymbol{c}_1$ for the case that $z_i = \mathsf{oob}$ and the case that $z_i \neq \mathsf{oob}$. This leads to the fact that $\mathbf{W}\cdot\boldsymbol{c}_0 + \boldsymbol{c}_1 \in \bigcap_{i\in[n]} f_{\boldsymbol{y}_i}^{-1}(z_i)$ for all $z_i \in \{0,\ldots,L\}\cup\{\mathsf{oob}\}$, which covers all the possible values of z_i.

Both of these cases follow directly from the definition of the preimage. In more detail, as described in the beginning of Section V-C, it holds that $f_{\boldsymbol{y}}^{-1}(\mathsf{oob})$ contains all the vectors $\boldsymbol{x}$ such that $\langle \boldsymbol{x}, \boldsymbol{y}\rangle > L$. For the case that $z_i = \mathsf{oob}$ it holds that $\langle \mathbf{W}\cdot\boldsymbol{c}_0 + \boldsymbol{c}_1, \boldsymbol{y}_i\rangle > L$, after the analysis above, and therefore $\mathbf{W}\cdot\boldsymbol{c}_0 + \boldsymbol{c}_1 \in f_{\boldsymbol{y}_i}^{-1}(\mathsf{oob})$. For the case that $z_i \in \{0,\ldots,L\}$ it holds that the preimage contains all the vectors $\boldsymbol{x}$, such that $\langle \boldsymbol{x}, \boldsymbol{y}_i\rangle = z_i$. Therefore, again with the analysis above, it follows that $\mathbf{W}\cdot\boldsymbol{c}_0 + \boldsymbol{c}_1 \in f_{\boldsymbol{y}_i}^{-1}(z_i)$ for $z_i \in \{0,\ldots,L\}$. Overall, this leads to the fact that $\mathbf{W}\cdot\boldsymbol{c}_0 \in \bigcap_{i\in[n]} f_{\boldsymbol{y}_i}^{-1}(z_i)$ for all $i \in [n]$ with $z_i \in \{0,\ldots,L\}\cup\{\mathsf{oob}\}$. □

The scheme described in Fig. 7 is obviously CPA secure for the functionality class $\mathcal{F}_{p,L}^m$ if the base FE scheme is CPA secure.

Theorem 14. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be the IND-CPA secure functional encryption scheme for the functionality class* $\mathcal{F}_p^m$*, with p prime, described in Fig. 7. Then the functional encryption scheme* $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ *for the functionality class* $\mathcal{F}_{p,L}^m$*, with p prime, described in Fig. 7 is IND-CPA secure. Namely, for any PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}_{\mathsf{FE}',\mathcal{A}}^{\text{IND-CPA}}(\lambda) \leq \mathsf{Adv}_{\mathsf{FE},\mathcal{B}}^{\text{IND-CPA}}(\lambda) \ .$$

C. Consistent Scheme for the Exponential Inner Product Functionality Class

The scheme used in this section, is another modified version of the inner product encryption scheme (described in Fig. 7) for the functionality class of exponential inner products $\mathcal{P}_p^m$. We define the functionality class more formally:

To modify the inner product encryption scheme to fit our new functionality class, we proceed without the discrete logarithm computation in the end of the decryption procedure and just output the value $g^{\langle \boldsymbol{x}, \boldsymbol{y}\rangle}$. For this scheme, the input consistency property can be proven formally.

Proof. We proceed in a similar way as in the proof of Theorem 3. To prove the input consistency of the described scheme, we need to show that no matter what ciphertext an adversary generates there exists at least one underlying plaintext that would explain the decryption behavior of ct under different functional keys $\mathsf{sk}_{\boldsymbol{y_i}}$ queried by the adversary $\mathcal{A}$ during the game. We prove this by relying on the algebraic properties of the groups for which the functional encryption scheme is defined. In more detail, we show that there exists always a solution for a linear equation system that is defined by the different inner product computations between the functional keys and the submitted ciphertext. The existence of a solution shows that there exists at least one plaintext that explains the functional decryption behavior.

Now, we describe how the game proceeds. In the first step, the challenger generates the master public key and the master secret key by executing the setup procedure $(\mathsf{mpk}, \mathsf{msk}) = ((\mathcal{G}, [\mathbf{A}]_g, [\mathbf{WA}]_g), \mathbf{W}) \leftarrow \mathsf{Setup}(1^\lambda)$. In the next step, the adversary $\mathcal{A}$ receives mpk and has access to a key generation oracle $\mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, \cdot)$. Whenever $\mathcal{A}$ queries the key generation oracle with a vector $\boldsymbol{y}_i$, the challenger generates $\mathsf{sk}_{\boldsymbol{y}_i} = \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, \boldsymbol{y}_i)$, adds $(\mathsf{sk}_{\boldsymbol{y}_i}, \boldsymbol{y}_i)$ to the list F and sends $\mathsf{sk}_{\boldsymbol{y}_i}$ to $\mathcal{A}$. At some point in the game, $\mathcal{A}$ sends ct to the challenger and the challenger computes $[z_i]_g := \mathsf{Dec}(\mathsf{mpk}, \boldsymbol{y}_i, \mathsf{sk}_{\boldsymbol{y}_i}, \mathsf{ct})$ for all $(\mathsf{sk}_{\boldsymbol{y}_i}, \boldsymbol{y}_i) \in F$. We consider how the decryption works in more detail and determine $[z_i]_g$ specifically corresponding to $\boldsymbol{y}_i$.

For the ciphertext, output by the adversary $\mathcal{A}$, we write $\mathsf{ct} = \begin{pmatrix} \boldsymbol{c}_0' \\ \boldsymbol{c}_1' \end{pmatrix}$, with $\boldsymbol{c}_0' = [\boldsymbol{c}_0]_g \in \mathbb{G}^{k+1}$ and $\boldsymbol{c}_1' = [\boldsymbol{c}_1]_g \in \mathbb{G}^m$. To be more specific, we also write $\boldsymbol{c}_0' \in \mathbb{G}^{k+1}$ and $\boldsymbol{c}_1' \in \mathbb{G}^m$ as explicit group elements, i.e. $\boldsymbol{c}_0' := \begin{pmatrix} g^{c_{0,1}} \\ \vdots \\ g^{c_{0,k+1}} \end{pmatrix}$ and $\boldsymbol{c}_0' := \begin{pmatrix} g^{c_{1,1}} \\ \vdots \\ g^{c_{1,m}} \end{pmatrix}$ with the generator g, $\boldsymbol{c}_0 := (c_{0,1}, \ldots, c_{0,k+1}) \in \mathbb{Z}_p^{k+1}$ and $\boldsymbol{c}_1 := (c_{1,1}, \ldots, c_{1,m}) \in \mathbb{Z}_p^m$. To show that the decryption procedure always decrypts to one underlying

element, we compute the decryption procedure for an arbitrary honestly generated key $\mathsf{sk}_{\boldsymbol{y}}$. We denote the master secret key as $\mathbf{W} := \begin{pmatrix} | & & | \\ \boldsymbol{w}_1 & \cdots & \boldsymbol{w}_{k+1} \\ | & & | \end{pmatrix}$, and correspondingly $\mathbf{W}^\top := \begin{pmatrix} - & \boldsymbol{w}_1^\top & - \\ & \vdots & \\ - & \boldsymbol{w}_{k+1}^\top & - \end{pmatrix}$. Using the matrix description, the functional key is defined as $\mathbf{W}^\top \cdot \boldsymbol{y} = \begin{pmatrix} \langle \boldsymbol{w}_1^\top, \boldsymbol{y} \rangle \\ \vdots \\ \langle \boldsymbol{w}_{k+1}^\top, \boldsymbol{y} \rangle \end{pmatrix} \in \mathbb{Z}_p^{k+1}$. For the decryption, we need to compute two different components: $[\boldsymbol{c}_0^\top \mathsf{sk}_{\boldsymbol{y}}]_g$ and $[\boldsymbol{c}_1^\top \boldsymbol{y}]_g$.

First, we describe how to compute $[\boldsymbol{c}_0^\top \mathsf{sk}_{\boldsymbol{y}}]_g$: We exponentiate all of the components of $\boldsymbol{c}_1'$ with the components of $\mathsf{sk}_{\boldsymbol{y}}$ and compute the product of the resulting vector components. In more detail, we compute $\prod_{i\in[k+1]} g^{c_{0,i}\cdot\langle \boldsymbol{w}_i^\top, \boldsymbol{y}\rangle} = g^{\sum_{i\in[k+1]}\langle c_{0,i}\cdot \boldsymbol{w}_i^\top, \boldsymbol{y}\rangle} = g^{\langle \sum_{i\in[k+1]} c_{0,i}\cdot \boldsymbol{w}_i^\top, \boldsymbol{y}\rangle} = g^{\langle \mathbf{W}\cdot \boldsymbol{c}_0, \boldsymbol{y}\rangle}$. We proceed in the same way for the second component $\prod_{i\in[m]} g^{c_{1,i}\cdot y_i} = g^{\sum_{i\in[m]} c_{1,i}\cdot y_i} = g^{\langle \boldsymbol{c}_1, \boldsymbol{y}\rangle}$.

To generate the final output, we need to multiply the two components, which results in $g^{\langle \mathbf{W}\cdot \boldsymbol{c}_0, \boldsymbol{y}\rangle} \cdot g^{\langle \boldsymbol{c}_1, \boldsymbol{y}\rangle} = g^{\langle \mathbf{W}\cdot \boldsymbol{c}_0, \boldsymbol{y}\rangle + \langle \boldsymbol{c}_1, \boldsymbol{y}\rangle} = g^{\langle \mathbf{W}\cdot \boldsymbol{c}_0 + \boldsymbol{c}_1, \boldsymbol{y}\rangle}$.

Coming back to the initial description of the game, the decryption procedure outputs $[z_i]_g := [\langle \mathbf{W}\cdot \boldsymbol{c}_0 + \boldsymbol{c}_1, \boldsymbol{y}_i\rangle]_g$. Due to the fact that the vectors $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$ are set by the adversary and the matrix $\mathbf{W}$ is fixed after the setup procedure, the decryption relies only on the value $\boldsymbol{y}_i$. This results in the decryptions $g^{\langle \mathbf{W}\cdot \boldsymbol{c}_0 + \boldsymbol{c}_1, \boldsymbol{y}_1\rangle}, \ldots, g^{\langle \mathbf{W}\cdot \boldsymbol{c}_0 + \boldsymbol{c}_1, \boldsymbol{y}_n\rangle}$ for all the different secret key queries $\boldsymbol{y}_i$. Consequently,

$\mathbf{W}\cdot \boldsymbol{c}_0 + \boldsymbol{c}_1 \in f_{g,\boldsymbol{y}_1}^{-1}([z_1]_g), \ldots, \mathbf{W}\cdot \boldsymbol{c}_0 + \boldsymbol{c}_1 \in f_{g,\boldsymbol{y}_n}^{-1}([z_n]_g)$, which further implies that

$$\mathbf{W}\cdot \boldsymbol{c}_0 + \boldsymbol{c}_1 \in \bigcap_{i\in[n]} f_{g,\boldsymbol{y}_i}^{-1}([z_i]_g) \ .$$

This makes the intersection non-empty for every possible ciphertext ct generated by $\mathcal{A}$. Therefore, the scheme is input consistent. □

The scheme described in Fig. 7 for the functionality class $\mathcal{P}_p^m$ achieves IND-CPA security:

Theorem 15. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be the IND-CPA secure functional encryption scheme for the functionality class* $\mathcal{F}_p^m$*, with p prime, described in Fig. 7. Then the functional encryption scheme* $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ *for the functionality class* $\mathcal{P}_p^m$*, with p prime, described in Fig. 7 is IND-CPA secure. Namely, for any PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{B}$ *such that*

$$\mathsf{Adv}_{\mathsf{FE}',\mathcal{A}}^{\text{IND-CPA}}(\lambda) \leq \mathsf{Adv}_{\mathsf{FE},\mathcal{B}}^{\text{IND-CPA}}(\lambda) \ .$$

This statement follows by a straightforward reduction to CPA security by observing that the restriction on the functional keys, i.e., the requirement $f_{\boldsymbol{y}}(\boldsymbol{x}^0) = f_{\boldsymbol{y}}(\boldsymbol{x}^1)$ is preserved for all keys, because if $\langle \boldsymbol{x}^0, \boldsymbol{y}\rangle = \langle \boldsymbol{x}^1, \boldsymbol{y}\rangle$ then it follows that $g^{\langle \boldsymbol{x}^0, \boldsymbol{y}\rangle} = g^{\langle \boldsymbol{x}^1, \boldsymbol{y}\rangle}$.

Appendix E
Input Consistency Compiler

A. First Compiler

To achieve input-consistency under CPA and CFE security, we augment the output of an encryption algorithm with a non-interactive zero-knowledge (NIZK) proof that an underlying plaintext exists. The NIZK proof is generated over the master public key, the encryption algorithm's randomness and the underlying plaintext. The zero-knowledge property of the NIZK makes sure that no information about the underlying plaintext is leaked, whereas the soundness prevents a malicious party from generating a valid proof over an invalid ciphertext. A formal description of this compiler is presented in Fig. 17 and the relation R_{in}, that needs to be supported by the NIZK scheme, is defined in Fig. 18. We show that the described construction indeed turns a functional encryption scheme into an input consistent functional encryption scheme.

$\underline{\mathsf{Setup}'(1^\lambda):}$
$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
Return $(\mathsf{mpk}', \mathsf{msk}') = ((\mathsf{CRS}, \mathsf{mpk}), \mathsf{msk})$
$\underline{\mathsf{KeyGen}'(\mathsf{mpk}', \mathsf{msk}', f):}$
Parse $\mathsf{mpk}' := (\mathsf{CRS}, \mathsf{mpk}), \mathsf{msk}' := \mathsf{msk}$
$\mathsf{sk}_f = \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$
Return $\mathsf{sk}_f' = \mathsf{sk}_f$
$\underline{\mathsf{Enc}'(\mathsf{mpk}', x):}$
Parse $\mathsf{mpk}' := (\mathsf{CRS}, \mathsf{mpk})$
$\mathsf{ct} = \mathsf{Enc}(\mathsf{mpk}, x; r)$ with $r \leftarrow \{0,1\}^\lambda$
Generate $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, (\mathsf{mpk}, \mathsf{ct}), (x, r))$ for R_{in} (Fig. 18)
Return $\mathsf{ct}' = (\mathsf{ct}, \pi)$
$\underline{\mathsf{Dec}'(\mathsf{mpk}', f, \mathsf{sk}_f', \mathsf{ct}'):}$
Parse $\mathsf{mpk}' := (\mathsf{CRS}, \mathsf{mpk}), \mathsf{sk}_f' := \mathsf{sk}_f, \mathsf{ct}' := (\mathsf{ct}, \pi)$
If $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}, \mathsf{ct}), \pi) = 1$
  Return $\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}_f, \mathsf{ct})$
Else
  Return $\perp$

Fig. 17: Input consistency compiler

Theorem 16. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a functional encryption scheme and* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *a NIZK proof system for relation* R_{in}*, then the construction* $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ *defined in Figure 17 satisfies input consistency.*

Relation R_{in}:
Instance: $z = (\mathsf{mpk}, \mathsf{ct})$
Witness: $w = (x, r)$, $x \in \mathcal{X}$, random coins r
$R_{\mathsf{in}}(z, w) = 1$ if and only if:
$\mathsf{ct} = \mathsf{Enc}(\mathsf{mpk}, x; r)$

Fig. 18: Relation used in the input consistency compiler

Namely, for any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}$ such that:

$$|\Pr[\text{in-CONS}^{\mathsf{FE}'}(1^\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\text{Sound}}_{\mathsf{NIZK},\mathcal{B}}(\lambda) \ .$$

Proof. To prove the input consistency of the scheme FE', we rely on the soundness of the NIZK proof system. In more detail, we construct an adversary $\mathcal{B}$ that generates a malicious proof, by relying on an adversary $\mathcal{A}$ for the input consistency experiment in-CONS$^{\mathsf{FE}}$.

In the beginning of the reduction, $\mathcal{B}$ receives a common reference string CRS from its underlying experiment, it generates $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, sets $\mathsf{mpk}' := (\mathsf{CRS}, \mathsf{mpk})$ and sends mpk' to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks a key generation query f, $\mathcal{B}$ computes the key $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{msk}, f)$, adds (sk_f, f) to the list F and sends sk_f to $\mathcal{A}$.

At some point, $\mathcal{A}$ sends a ciphertext $\mathsf{ct}' = (\mathsf{ct}, \pi)$ to $\mathcal{B}$. If $\mathsf{NIZK.Verify}((\mathsf{mpk}, \mathsf{ct}), \pi) = 0$ then the adversary $\mathcal{B}$ halts. In this case, $\mathsf{Dec}(\mathsf{mpk}, f_i, \mathsf{sk}_{f_i}, \mathsf{ct})$ yields $\perp$ for all $i \in [n]$, by definition of the compiler (and hence the adversary $\mathcal{A}$ loses the game). If $\mathsf{NIZK.Verify}((\mathsf{mpk}, \mathsf{ct}), \pi) = 1$, $\mathcal{B}$ simply outputs $(\mathsf{mpk}_1, \mathsf{ct}, \pi)$ as its forgery and halts.

Let us analyze the output of $\mathcal{A}$ to see that the condition to break input consistency must imply a soundness violation of the NIZK scheme. In more detail, we define the event E as the event that the adversary $\mathcal{A}$ performs a consistency attack under the assumption that $(\mathsf{mpk}, \mathsf{ct}) \in L$ and show that the occurrence of the event E would contradict the assumption.

Now, we analyze the possible outcomes for the decryptions y_i in the case of a consistency attack. We show that $y_i \neq \perp$ for all $i \in [n]$ (this is covered by event E_1). Furthermore we show that if $y_i \neq \perp$ then there exists an x such that $x \in \bigcap_{i\in[n]} f_i^{-1}(y_i)$ (this is denoted by event E_2).

In the case of event E_1, we assume that at least one of the decryptions is equal to $\perp$, i.e. $y_i \neq \perp$. We distinguish between two cases:

1) It holds that $y_i = \perp$ for all $i \in [n]$. In this case, the adversary $\mathcal{A}$ did not perform a consistency attack. In more detail, the intersection $\bigcap_{i\in[n]} f_i^{-1}(y_i)$ will contain the $\perp$ value.
2) At least one, but not all, of the decryptions are equal to $\perp$, i.e. $y_i = \perp$. Since $\perp$ is not an element $\mathcal{X}$, and therefore not an encryption value, then, by perfect correctness of the underlying FE scheme, it follows that there exists no w such that $((\mathsf{mpk}, \mathsf{ct}), w) \in R_{\mathsf{in}}$ (i.e. it cannot be a valid instance). This is a contradiction to the assumption that $(\mathsf{mpk}, \mathsf{ct}) \in L$.

Considering both the above mentioned points, we can conclude that $y_i \neq \perp$ for all $i \in [n]$.

For the analysis of event E_2, we assume, for the sake of contradiction, that the intersection $\bigcap_{i\in[n]} f_i^{-1}(y_i)$ is empty and it holds (with respect to event E_1) that $y_i \neq \perp$ for all $i \in [n]$. In this case, the adversary $\mathcal{A}$ has generated a valid proof π for an $x \notin L_{\mathsf{in}}$. Again, by the perfect correctness of the FE scheme, the adversary $\mathcal{B}$ broke the soundness of the NIZK scheme, because it has found a ciphertext ct and provided a proof to be a valid encryption while the functional outputs say that there is no such underlying plaintext.

By combining the events E_1 and E_2, we have proven that event E cannot occur. To recap, whenever $(\mathsf{mpk}, \mathsf{ct}) \in L$, it is not possible for an adversary $\mathcal{A}$ to perform a consistency attack. Hence the only way the adversary can break setup consistency is by breaking the soundness property of the NIZK scheme, i.e., providing the statement $(\mathsf{mpk}, \mathsf{ct}) \notin L$.

This yields the bound

$$|\Pr[\text{in-CONS}^{\mathsf{FE}'}(1^\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\text{Sound}}_{\mathsf{NIZK},\mathcal{B}}(\lambda) \ .$$

and therefore we obtain the theorem. □

Besides proving that the compiler achieves input consistency, we also need to prove the security preservation under the two different notions of IND-CPA security and CFE security.

We first prove the security preservation of the compiler under CPA security and conclude with the preservation for CFE security.

Theorem 17. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an IND-CPA secure functional encryption scheme and* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *a NIZK proof system, then the construction* $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$*, defined in Figure 17, is IND-CPA secure. Namely, for any PPT adversary $\mathcal{A}$, there exist PPT algorithms $\mathcal{B}$ and $\mathcal{B}'$ such that*

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE}',\mathcal{A}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}'}(\lambda) \ .$$

Proof. To prove this statement, we use a hybrid argument with the games defined in Fig. 19. Note that G_0 corresponds to the game IND-CPA$^{\mathsf{FE}}_0(1^\lambda, \mathcal{A})$ and G_3 to the game IND-CPA$^{\mathsf{FE}}_1(1^\lambda, \mathcal{A})$. This results in:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE}',\mathcal{A}}(1^\lambda) = |\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(\lambda)| \ .$$

We describe the different games in more detail:

Game G_1: In this game, we change from an honestly generated CRS and honestly generated proofs to a simulated CRS and simulated proofs. The transition from G_0 to G_1 is justified by the zero-knowledge property of the NIZK. Namely, in Lemma 3, we exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Game G_2: In this game, we change from an encryption of x^0 to x^1 for the encryption queries. The transition from G_1 to G_2 is justified by the IND-CPA security of FE. Namely, in Lemma 4, we exhibit a PPT adversary $\mathcal{B}_1$ such that:

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_1}(\lambda) \ .$$

Game G_3: This game is the $\text{IND-CPA}^{\mathsf{FE}}_1(1^\lambda, \mathcal{A})$ game. The transition from G_2 to G_3 is almost symmetric to the transition from G_0 to G_1 except from the fact that the reduction encrypts x^1 instead of x^0. As in Lemma 3, the transition is justified by the zero-knowledge property of NIZK. Namely, we can exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Putting everything together, we obtain the theorem. □

Lemma 3 (Transition from G_0 to G_1)**.** *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_0$ such that*

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_0$ that simulates G_β towards $\mathcal{A}$ when interacting with the underlying $\mathrm{ZK}^{\mathsf{NIZK}}_\beta$ experiment.

In the beginning of the reduction, $\mathcal{B}_0$ receives CRS from the $\mathrm{ZK}^{\mathsf{NIZK}}_\beta$ experiment. It generates a functional encryption instance $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, sets $\mathsf{mpk}' = (\mathsf{CRS}, \mathsf{mpk})$ and gives mpk' to the adversary.

Whenever $\mathcal{A}$ asks an encryption query (x^0, x^1), $\mathcal{B}_0$ generates the ciphertext $\mathsf{ct} = \mathsf{Enc}(\mathsf{mpk}, x^0; r)$ with $r \leftarrow \{0,1\}^\lambda$ and sends $y = (\mathsf{mpk}, \mathsf{ct})$ and $w = (x, r)$ as a statement-witness pair to its challenger. As an answer, $\mathcal{B}_0$ receives a proof π for R_{in}. It sets $\mathsf{ct}' = (\mathsf{ct}, \pi)$ and sends it to $\mathcal{A}$.

For a key generation query f, $\mathcal{B}_0$ generates $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$ for and sends $\mathsf{sk}'_f = \mathsf{sk}_f$ as a reply to $\mathcal{A}$.

This covers the simulation of the game G_β. Finally $\mathcal{B}_0$ outputs the same bit β' returned by $\mathcal{A}$. It follows, from the perfect simulation, that the advantage of $\mathcal{B}_0$ is the same as the advantage of $\mathcal{A}$. □

Lemma 4 (Transition from G_1 to G_2)**.** *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_1$ such that*

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_1}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_1$ that simulates $\mathsf{G}_{1+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\text{IND-CPA}^{\mathsf{FE}}_\beta$ experiment.

In the beginning of the reduction, $\mathcal{B}_1$ receives mpk from the experiment. It simulates a CRS, i.e. $(\mathsf{CRS}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$, sets $\mathsf{mpk}' = (\mathsf{CRS}, \mathsf{mpk})$ and gives mpk' to the adversary.

Whenever $\mathcal{A}$ asks an encryption query (x^0, x^1), $\mathcal{B}_1$ forwards it to its own encryption oracle to receive $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^\beta)$, simulates a proof for the relation R_{in}, i.e. $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x^\beta)$ and sends $\mathsf{ct}' = (\mathsf{ct}, \pi)$ to $\mathcal{A}$.

For a key generation query f, $\mathcal{B}_1$ queries its own key generation oracle on f to receive $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$, sets $\mathsf{sk}'_f = \mathsf{sk}_f$ and sends sk'_f to $\mathcal{A}$.

This covers the simulation of the game $\mathsf{G}_{1+\beta}$. Finally $\mathcal{B}_1$ outputs the same bit β' returned by $\mathcal{A}$. It follows, from the perfect simulation, that the advantage of $\mathcal{B}_1$ is the same as the advantage of $\mathcal{A}$. □

Beside showing the IND-CPA security preservation, we also need to show the CFE security preservation.

Theorem 18. *Let $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a CFE secure functional encryption scheme, i.e., there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ such that $\mathrm{Real}^{\mathsf{FE}}(1^\lambda, \mathcal{A}) \approx \mathrm{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$ w.r.t. any adversary $\mathcal{A}$, and let further $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ be a NIZK proof system for the relation R_{in}, then for the construction $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$, defined in Figure 17, we can design a simulator $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2, \mathcal{S}'_3)$ such that for any adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ against the new scheme we can design adversarys $\mathcal{A}$ and $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathcal{D}',\mathrm{CFE}}_{\mathsf{FE}',\mathcal{A}',\mathcal{S}'}(\lambda) \leq \mathsf{Adv}^{\mathcal{D},\mathrm{CFE}}_{\mathsf{FE},\mathcal{A}}(\lambda) + \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}}(\lambda).$$

Proof. The simulator is defined as follows: $\mathcal{S}'_1$ runs $\mathcal{S}_1$ to obtain the pair (mpk, s) and runs the simulator of the NIZK scheme to obtain (CRS, τ) and outputs $\mathsf{mpk}' := (\mathsf{mpk}, \mathsf{CRS})$. Next, $\mathcal{S}'_2 := \mathcal{S}_2$, and finally, $\mathcal{S}'_3$ runs $\mathcal{S}_3$, receives a ciphertext ct_ℓ, and simulates a NIZK proof using the trapdoor τ for instance $(\mathsf{mpk}', \mathsf{ct})$. Note that the internal state managed by $\mathcal{S}$ is managed by $\mathcal{S}'$. We obtain the statement again by a sequence of hybrid steps. Let G_0 be the real CFE experiment and G_2 the ideal CFE experiment with the above simulator. Let G_1 be a hybrid experiment, where the only change is that we replace the CRS and simulate all NIZK proofs. Analogous to the proof of Theorem 17, an adversary $\mathcal{B}$ with advantage α in distinguishing the outputs of experiments G_0 and G_1 (with respect to a certain adversary $\mathcal{A}''$) directly yields a distinguisher telling apart simulated and genuine proofs with the same advantage. For the second step, we see that any pair $(\mathcal{D}', \mathcal{A}')$ with advantage α in distinguishing the outputs of the experiments G_1 and G_2 can be transformed into a pair $(\mathcal{D}, \mathcal{A})$ such that the outputs of $\mathrm{Real}^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ and $\mathrm{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$ are distinguishable by $\mathcal{D}$ with advantage α. To see this, note that by the modular design of the scheme, the adversary can be defined as follows: when the first adversary $\mathcal{A}_1$ receives mpk it simulates CRS and internally runs an instance of $\mathcal{A}'_1((\mathsf{mpk}, \mathsf{CRS}))$. For requests to the key-generation oracle, $\mathcal{A}_1$ simply relays them to the oracle of $\mathcal{A}'_1$. Besides this, any internal state of $\mathcal{A}'_1$ is maintained by $\mathcal{A}_1$, and passed on to the second adversary. $\mathcal{A}_1$ outputs whatever $\mathcal{A}'_1$ outputs. Second, whenever $\mathcal{A}_2$ receives some ciphertext,

say ct_ℓ, it internally runs $\mathcal{A}_2'$ on input (ct_ℓ, π), where π is a simulated proof for the relation R_{in}. Finally, $\mathcal{A}_2$ outputs whatever $\mathcal{A}_2'$ outputs. We see that the output distribution of $\mathsf{Real}^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ is identical to the output of $\mathcal{A}'$ in experiment G_1 and the output distribution of $\mathsf{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$ is identical to the output distribution of G_2, the ideal experiment with FE' and simulator $\mathcal{S}'$. This proves the theorem. □

B. Second Advanced Compiler

For the advanced input consistency compiler that takes a CPA secure scheme and achieves CCA security, we make use of the Naor-Yung approach [55] and combine it with the approach of the presented input consistency compiler. In more detail, we run two different instances of the functional encryption scheme and create a proof that shows that both of these encryptions are generated in a valid way, i.e. there exists a random r_i and a message x_i to create a ciphertext ct_i for $i \in [2]$. The compiler is displayed in Fig. 20. In comparison to the NIZK proof system used in the input consistency compiler above, we need to assume one-time simulation-soundness for this advanced case. This leads to the following theorem which is of independent interest beyond the study of consistency.

Theorem 19. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an IND-CPA secure functional encryption scheme and* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *a NIZK proof system satisfying one-time simulation soundness, then the construction* $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$*, defined in Figure 20, is IND-CCA secure. Namely, for any PPT adversary* $\mathcal{A}$*, there exist PPT adversaries* $\mathcal{B}, \mathcal{B}'$ *and* $\mathcal{B}''$*, such that:*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{FE}',\mathcal{A}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{B}}(\lambda) + \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + 2 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}''}(\lambda).$$

Proof. To prove this statement, we use a hybrid argument with the games defined in Fig. 22. Note that G_0 corresponds to the game $\text{IND-CCA}_0^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ and G_4 to the game $\text{IND-CCA}_1^{\mathsf{FE}}(1^\lambda, \mathcal{A})$. This results in:

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{FE}',\mathcal{A}}(1^\lambda) = |\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_4}_{\mathcal{A}}(1^\lambda)| \ .$$

We describe the different games in more detail:

Game G_1: In this game, we change from an honestly generated CRS and honestly generated proofs to a simulated CRS and simulated proofs. The transition from G_0 to G_1 is justified by the zero-knowledge property of NIZK. Namely, in Lemma 5, we exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Game G_2: In this game, we change from an encryption of x^0 to x^1 in the first component of the ciphertext, i.e. $\mathsf{ct} = (\mathsf{Enc}(\mathsf{mpk}_1, x^1), \mathsf{Enc}(\mathsf{mpk}_2, x^0),)$. The transition from G_1 to G_2 is justified by the IND-CPA security of FE and the one-time simulation-soundness of NIZK. Namely, in Lemma 6, we exhibit PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_2}(\lambda) \ .$$

Game G_3: In this game, we change from an encryption of x^0 to x^1 in the second component of the ciphertext, i.e. $\mathsf{ct} = (\mathsf{Enc}(\mathsf{mpk}_1, x^1), \mathsf{Enc}(\mathsf{mpk}_2, x^1),)$. The transition from G_2 to G_3 is almost symmetric to the transition from game G_1 to G_2 except that it is not necessary to rely on the one-time simulation soundness of the NIZK system and the ciphertext contains an encryption of x^1 in the first position. As in Lemma 6, the transition is justified by the IND-CPA security of FE. Namely, we can exhibit a PPT adversary $\mathcal{B}_2$ such that:

$$|\mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_1}(\lambda) \ .$$

Game G_4: This game is the $\text{IND-CCA}_1^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ game. The transition from G_3 to G_4 is almost symmetric to the transition from G_0 to G_1 except from the fact that the reduction encrypts x^1 instead of x^0. As in Lemma 5, the transition is justified by the zero-knowledge property of NIZK. Namely, we can exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_4}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Putting everything together, we obtain the theorem. □

Lemma 5 (Transition from G_0 to G_1)**.** *For any PPT adversary* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{B}_0$ *such that*

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_0$ that simulates G_β towards $\mathcal{A}$ when interacting with the underlying $\text{ZK}_\beta^{\mathsf{NIZK}}$ experiment.

In the beginning of the reduction, $\mathcal{B}_0$ receives CRS from the $\text{ZK}_\beta^{\mathsf{NIZK}}$ experiment. It generates two functional encryption instance $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$ for $i \in [2]$, sets $\mathsf{mpk}' = (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]})$ and gives mpk' to the adversary.

Whenever $\mathcal{A}$ asks an encryption query (x^0, x^1), $\mathcal{B}_0$ generates the ciphertext $\mathsf{ct}_i = (\mathsf{Enc}(\mathsf{mpk}_i, x^0; r_i))_{i\in[2]}$ with $r_i \leftarrow \{0,1\}^\lambda$ for $i \in [2]$ and sends $y = (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}$ and $w = (x, \{r_i\}_{i\in[2]})$ as a statement-witness pair to its challenger. As an answer, $\mathcal{B}_0$ receives a proof π for the relation $R_{\mathsf{in}}^{\mathsf{CCA}}$. It sets $\mathsf{ct}' = (\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$ and sends it to $\mathcal{A}$.

For a key generation query f, $\mathcal{B}_0$ generates $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f)$ for $i \in [2]$ and sends $\mathsf{sk}_f' = \{\mathsf{sk}_{f,i}\}_{i\in[2]}$ as a reply to $\mathcal{A}$.

Game	CRS & π	ct	justification/remark
G_0	$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$	$\mathsf{Enc}(\mathsf{mpk}, x^0)$	
G_1	$\boxed{\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)}$ $\boxed{\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)}$	$\mathsf{Enc}(\mathsf{mpk}, x^0)$	Zero-knowledge of NIZK
G_2	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}, \boxed{x^1})$	IND-CPA security of FE
G_3	$\boxed{\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)}$ $\boxed{\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)}$	$\mathsf{Enc}(\mathsf{mpk}, x^1)$	Zero-knowledge of NIZK

Fig. 19: Overview of the games to prove the IND-CPA security preservation of the input consistency compiler described in Fig. 17.

Whenever $\mathcal{A}$ submits a decryption query (f, ct'), with $\mathsf{ct}' = (\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$, $\mathcal{B}_0$ generates the functional key $\mathsf{sk}_{f,1} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_1, \mathsf{msk}_1, f)$ and executes $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]})$. If $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}) = 1$, $\mathcal{B}_0$ computes $y := \mathsf{Dec}(\mathsf{mpk}_1, f, \mathsf{sk}_{f,1}, \mathsf{ct}_1)$ and sends y to $\mathcal{A}$. Otherwise, $\mathcal{B}_0$ sends $\perp$ to $\mathcal{A}$.

This covers the simulation of the game G_β. Finally $\mathcal{B}_0$ outputs the same bit β' returned by $\mathcal{A}$. It follows, from the perfect simulation, that the advantage of $\mathcal{B}_0$ is the same as the advantage of $\mathcal{A}$.

□

As in [51], we prove a claim that shows that whenever a decryption oracle query is asked and this query contains a valid NIZK proof, then the corresponding ciphertext is explainable under the queried function. This is necessary for the proof of the transition from G_1 to G_2 for the simulation of the decryption oracle.

Claim 1. *For any PPT adversary $\mathcal{A}$ participating in $\mathsf{G}_{1+\beta}$ for $\beta \in \{0,1\}$, the probability that, during the experiment, $\mathcal{A}$ queries its decryption oracle QDec with a function-ciphertext-pair that is not explainable but has an accepting proof is negligible. Namely, we exhibit a PPT adversary $\mathcal{B}_1$, such that*

$$\Pr\left[\begin{array}{l}\exists (f, \{\mathsf{ct}'_i\}_{i\in[2]}, \pi') \in Q : \\ (\{\mathsf{ct}'_i\}_{i\in[2]}, \pi') \neq (\{\mathsf{ct}_i\}_{i\in[2]}, \pi), \\ \mathsf{NIZK.Verify}(\mathsf{CRS}, \{\mathsf{ct}'_i\}_{i\in[2]}, \pi') = 1 \text{ and} \\ \mathsf{Dec}(\mathsf{mpk}_1, \mathsf{sk}_{f,1}, \mathsf{ct}'_1) \neq \mathsf{Dec}(\mathsf{mpk}_2, \mathsf{sk}_{f,2}, \mathsf{ct}'_2)\end{array}\right] \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda),$$

where $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f)$ for $i \in [2]$, $(\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$ is the reply to the encryption query (x^0, x^1) made by $\mathcal{A}$ and Q the list containing all the decryption queries $(f, \{\mathsf{ct}'_i\}_{i\in[2]}, \pi')$ asked by $\mathcal{A}$, knowing the master public key $\mathsf{mpk}' := (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]})$, the reply to its challenge query $(\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$ and by having access to the key generation oracle $\mathsf{KeyGen}'(\mathsf{mpk}', \mathsf{msk}', \cdot)$, during the game.

Proof. We build an adversary $\mathcal{B}_1$ that simulates $\mathsf{G}_{1+\beta}$ towards $\mathcal{A}$ when interacting with the underlying one time simulation-soundness experiment.

After the adversary $\mathcal{B}_1$ has received CRS from the underlying experiment, it generates $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$ for $i \in [2]$, sets $\mathsf{mpk}' := (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]})$ and sends mpk' to $\mathcal{A}$. Whenever $\mathcal{A}$ submits a key generation query f, $\mathcal{B}_1$ generates the functional keys $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f)$ for $i \in [2]$, sets $\mathsf{sk}'_f = \{\mathsf{sk}_{f,i}\}_{i\in[2]}$ and sends it to $\mathcal{A}$.

For the challenge query (x^0, x^1) asked by $\mathcal{A}$, $\mathcal{B}_1$ computes $\mathsf{ct}_1 = \mathsf{Enc}(\mathsf{mpk}_1, x^\beta)$ and $\mathsf{ct}_2 = \mathsf{Enc}(\mathsf{mpk}_2, x^0)$ (where $\beta = 0$ in game G_1 and $\beta = 1$ in G_2) and asks its experiment for a simulated proof π of the statement $(\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}$. It sets $\mathsf{ct}' := (\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$ and sends ct' to $\mathcal{A}$.

Whenever $\mathcal{A}$ outputs a decryption query $(f, \mathsf{ct}' := (\{\mathsf{ct}\}_{i\in[2]}, \pi))$, $\mathcal{B}_1$ verifies the proof. If the output of the verification is 1, $\mathcal{B}_1$ computes $y_{f,1} = \mathsf{Dec}(\mathsf{mpk}_1, \mathsf{sk}_{f,1}, \mathsf{ct}_1)$ and $y_{f,2} = \mathsf{Dec}(\mathsf{mpk}_2, \mathsf{sk}_{f,2}, \mathsf{ct}_2)$. If $y_{f,1} \neq y_{f,2}$, $\mathcal{B}_1$ sends $(\{\mathsf{ct}\}_{i\in[2]}, \pi)$ as a proof forgery to its challenger. Otherwise it sends $y_{f,1}$ to $\mathcal{A}$. If the verification outputs 0, $\mathcal{B}_1$ sends $\perp$ to $\mathcal{A}$. □

After introducing and proving Claim 1, we prove the transition from G_1 to G_2.

Lemma 6 (Transition from G_1 to G_2)**.** *For any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, such that*

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_2}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_2$ that simulates $\mathsf{G}_{1+\beta}$ to $\mathcal{A}$ when interacting with the underlying $\text{IND-CPA}^{\mathsf{FE}}_\beta$ experiment.

In the beginning of the reduction, $\mathcal{B}_2$ receives mpk_1 from the underlying experiment. It simulates a CRS, i.e. $(\mathsf{CRS}, \tau) \leftarrow \mathcal{S}_1(1^\lambda)$, generates a functional encryp-

$\underline{\mathsf{Setup}'(1^\lambda):}$
$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$
For $i \in [2]$
  $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$
Return $(\mathsf{mpk}', \mathsf{msk}')$
  $= ((\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]}), \{\mathsf{msk}_i\}_{i\in[2]})$
$\underline{\mathsf{KeyGen}'(\mathsf{mpk}', \mathsf{msk}', f):}$
Parse $\mathsf{mpk}' := (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]}), \mathsf{msk}' := \{\mathsf{msk}_i\}_{i\in[2]}$
For $i \in [2]$
  $\mathsf{sk}_{f,i} = \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f)$
Return $\mathsf{sk}'_f = \{\mathsf{sk}_{f,i}\}_{i\in[2]}$
$\underline{\mathsf{Enc}'(\mathsf{mpk}', x):}$
Parse $\mathsf{mpk}' := (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]})$
For $i \in [2]$
  $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{mpk}_i, x; r_i)$ with $r_i \leftarrow \{0,1\}^\lambda$
If $\exists i \in [2] : \mathsf{ct}_i = \mathsf{err}$ then return err
$\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}, (x, \{r_i\}_{i\in[2]}))$
for $R^{\mathsf{CCA}}_{\mathsf{in}}$ (Fig. 21)
If $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}, \pi) = 0$ return err
Return $\mathsf{ct}' = (\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$
$\underline{\mathsf{Dec}'(\mathsf{mpk}', f, \mathsf{sk}'_f, \mathsf{ct}'):}$
Parse $\mathsf{mpk}' := (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]}), \mathsf{sk}'_f := \{\mathsf{sk}_{i,f}\}_{i\in[2]}$,
  $\mathsf{ct}' := (\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$
If $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}, \pi) = 1$
  Return $\mathsf{Dec}(\mathsf{mpk}_1, f, \mathsf{sk}_{f,1}, \mathsf{ct}_1)$
Else
  Return $\perp$

Fig. 20: Advanced input consistency compiler. Shaded instructions indicate difference to the simpler input consistency compiler.

Relation $R^{\mathsf{CCA}}_{\mathsf{in}}$:
Instance: $z = (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}$
Witness: $w = (x, \{r_i\}_{i\in[2]})$, $x \in \mathcal{X}$, random coins r_i
$R^{\mathsf{CCA}}_{\mathsf{in}}(z, w) = 1$ if and only if:
  $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{mpk}_i, x; r_i)$, for both $i \in [2]$

Fig. 21: Relation used in the advanced input consistency compiler.

tion instance $(\mathsf{mpk}_2, \mathsf{msk}_2) \leftarrow \mathsf{Setup}(1^\lambda)$ sets $\mathsf{mpk}' := (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]})$ and sends mpk' to $\mathcal{A}$. Whenever $\mathcal{A}$ submits a key generation query f, $\mathcal{B}_2$ forwards this query to its own key generation oracle $\mathsf{KeyGen}(\mathsf{mpk}_1, \mathsf{msk}_1, \cdot)$, to receive $\mathsf{sk}_{f,1}$ as an answer. Then, $\mathcal{B}_2$ generates $\mathsf{sk}_{f,2} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_2, \mathsf{msk}_2, f)$ by itself, sets $\mathsf{sk}_f = \{\mathsf{sk}_{f,i}\}_{i\in[2]}$ and sends it to $\mathcal{A}$.

For the challenge query (x^0, x^1) asked by $\mathcal{A}$, $\mathcal{B}_2$ forwards it to its own encryption oracle and receives $\mathsf{ct}_1 = \mathsf{Enc}(\mathsf{mpk}_1, x^\beta)$ as an answer. It generates $\mathsf{ct}_2 = \mathsf{Enc}(\mathsf{mpk}_2, x^0)$ by itself, simulates a valid proof π of the relation R_{in} using the statement $y = (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}$, i.e. $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, y)$ and sends $\mathsf{ct}' := (\{\mathsf{ct}_i\}_{i\in[2]}, \pi)$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks a decryption query $(f, \mathsf{ct}' := (\{\mathsf{ct}_i\}_{i\in[2]}, \pi))$, $\mathcal{B}_2$ first verifies the proof π, i.e. it executes $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}, \pi)$. If the verification outputs 1, $\mathcal{B}_2$ generates $\mathsf{sk}_{f,2} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_2, \mathsf{msk}_2, f)$, executes $\mathsf{Dec}(\mathsf{mpk}_2, f, \mathsf{sk}_{f,2}, \mathsf{ct}_2)$ and sends the result to $\mathcal{A}$. This is contrary to the actual decryption oracle, which would always generate the key $\mathsf{sk}_{f,1} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_1, \mathsf{msk}_1, f)$ and use it to decrypt the first ciphertext ct_1. Since Claim 1 shows that for all ciphertext queries made by $\mathcal{A}$ that have a valid proof (except with negligible probability) it holds that $\mathsf{Dec}(\mathsf{mpk}_1, \mathsf{sk}_{f,1}, \mathsf{ct}_1) = \mathsf{Dec}(\mathsf{mpk}_2, \mathsf{sk}_{f,2}, \mathsf{ct}_2)$. Therefore, it is possible to generate a functional key for either of the position and use the corresponding decryption output as a reply for $\mathcal{A}$[6]. If the verification outputs 0, $\mathcal{B}_2$ sends $\perp$ to $\mathcal{A}$.

This covers the simulation of the game $\mathsf{G}_{1+\beta}$. Finally $\mathcal{B}_2$ outputs the same bit β' returned by $\mathcal{A}$. Together with the analysis of adversary $\mathcal{B}_1$, this yields the advantage mentioned in the lemma. □

After proving that the compiler does the security lifting from CPA to CCA, we also need to show that the compiler achieves input consistency.

Theorem 20. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a functional encryption scheme and* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *a NIZK proof system for* $R^{\mathsf{CCA}}_{\mathsf{set}}$ *(Fig. 26), then the construction* $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ *defined in Figure 20 is input consistent. Namely, for any PPT adversary* $\mathcal{A}$*, exists a PPT adversary* $\mathcal{B}$ *such that*

$$|\Pr[\text{in-CONS}^{\mathsf{FE}'}(1^\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\mathrm{Sound}}_{\mathsf{NIZK},\mathcal{B}}(\lambda) \ .$$

Proof. The proof proceeds exactly in the same way as the proof of Theorem 16. It is ensured by the soundness of the NIZK proof that both of the ciphertexts encrypt the same underlying message. □

1) Instantiations: Our compilers can be instantiated with any NIZK scheme such as [45], [46]. An important special case are lattice based constructions. Since recent

[6]It is not necessary to rely on Claim 1 in the transition from G_2 to G_3, since $\mathcal{B}_2$ is able to generate a functional key for the first position and therefore is able to generate the decryption oracle perfectly.

Game	CRS & π	ct	justification/remark
G_0	$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$	
G_1	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$	Zero-knowledge of NIZK
G_2	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$	IND-CPA of FE and simulation-soundness of NIZK
G_3	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^1)$	IND-CPA of FE
G_4	$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^1)$	Zero-knowledge of NIZK

Fig. 22: Overview of the games to prove the IND-CCA security of the advanced input consistency compiler described in Fig. 20.

results [57], [60] show how to construct NIZKs from LWE, it can be combined with a functional encryption scheme for all classes of circuits from LWE known from [27], [42], to obtain a specific instantiation of the presented compiler from LWE. Furthermore, specific versions of the compiler for restricted classes, namely inner-product classes achievable from standard assumptions [7], could be obtained by employing designated verifier NIZKs from [26], [30], [48], [59]. Since the latter are based on the Diffie-Hellman assumption, the overall scheme is input-consistent based on standard assumptions. (The reason designated verification is sufficient follows from the fact that the key distribution process can be trusted in the input consistency scenario.) We note that it is an interesting research direction to investigate efficient constructions of consistent FE schemes.

Appendix F
Strong Input Consistency Compiler

In this section, we show that the verifiability property introduced in [11], can be understood as providing strong input consistency. Due to this modular reduction, we also directly inherit their compiler, and in general any compiler that achieves verifiable functional encryption.

We recall the syntax of VFE in Definition 18. In a nutshell, VFE extends standard FE by two additional algorithms:

$\mathsf{VerifyCT}(\mathsf{mpk}, \mathsf{ct})$**:** A predicate on ciphertexts (w.r.t. the public key) that decides whether ct is valid.

$\mathsf{VerifySK}(\mathsf{mpk}, f, \mathsf{sk})$**:** A predicate on pairs (sk, f) (w.r.t. the public key) that decides whether the pair is a valid key-function pair.

The *verifiability property* of [11] restated in Definition 19 of the supplemental material requires that whenever $\mathsf{VerifyCT}(\mathsf{mpk}, \mathsf{ct}) = \mathsf{VerifySK}(\mathsf{mpk}, f, \mathsf{sk}) = 1$, we have $\Pr[\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}, \mathsf{ct}) = f(x)] = 1$ (where the implication must hold over all possible values of the involved arguments).

We define a simple compiler, defined Fig. 23, in that makes use of these two verification procedures to achieve strong input consistency. Informally, the first algorithm is used as a ciphertext verification check (and we return $\perp$ if the check fails) and the second function is used to verify key-function pairs (and return $\diamond$ if the check fails). Note that the transformation clearly preserves the confidentiality notion of the underlying VFE scheme.

Theorem 21. *Let* $\mathsf{VFE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{VerifyCT}, \mathsf{VerifySK})$ *be a verifiable functional encryption scheme then the construction* $\mathsf{FE}' = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}')$ *defined in Figure 23 is strongly input consistent. Namely, for any PPT adversary* $\mathcal{A}$*, it holds that:*

$$|\Pr[\text{st-in-CONS}^{\mathsf{FE}'}(1^\lambda, \mathcal{A}) = 1]| = 0 \ .$$

Proof. To prove the strong input consistency of the scheme FE', we rely on the verifiability of the VFE scheme. In more detail, we construct an adversary $\mathcal{B}$ that violates verifiability with the probability with which $\mathcal{A}$ wins the strong input consistency experiment st-in-CONS.

In the beginning of the reduction, $\mathcal{B}$ receives a master public key mpk, two ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, and a tuple of secret keys with the corresponding functions $\{(\mathsf{sk}_j, f_j)\}_{j\in[n]}$ from $\mathcal{A}$. In the next step, $\mathcal{B}$ computes $y_{j,i} := \mathsf{Dec}'(\mathsf{mpk}', f_j, \mathsf{sk}_j, \mathsf{ct}_i)$ for all $j \in [n], i \in \{1, 2\}$ and defines the set F as all the functional keys that do not output $\diamond$, i.e. $F := \{(\mathsf{sk}_j, f_j)\}_{j\in[n]\wedge(y_{j,1}\neq\diamond\vee y_{j,2}\neq\diamond)}$. Let E denote the event that the intersection $\bigcap_{j\in[n],(\cdot,f_j)\in F} f_j^{-1}(y_{j,i})$ is equal to $\emptyset$. We show that the occurrence of E contradicts the verifiability notion.

For the sake of contradiction, we assume that verifiability holds as stated. Now, we analyze the different scenarios in

$\underline{\mathsf{Setup}'(1^\lambda):}$
$(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
Return $(\mathsf{mpk}', \mathsf{msk}') = (\mathsf{mpk}, \mathsf{msk})$
$\underline{\mathsf{KeyGen}'(\mathsf{mpk}', \mathsf{msk}', f):}$
Parse $\mathsf{mpk}' \coloneqq \mathsf{mpk}, \mathsf{msk}' \coloneqq \mathsf{msk}$
$\mathsf{sk}_f = \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$
Return $\mathsf{sk}'_f = \mathsf{sk}_f$
$\underline{\mathsf{Enc}'(\mathsf{mpk}', x):}$
Parse $\mathsf{mpk}' \coloneqq \mathsf{mpk}$
$\mathsf{ct} = \mathsf{Enc}(\mathsf{mpk}, x)$
Return $\mathsf{ct}' = \mathsf{ct}$
$\underline{\mathsf{Dec}'(\mathsf{mpk}', f, \mathsf{sk}'_f, \mathsf{ct}'):}$
Parse $\mathsf{mpk}' \coloneqq \mathsf{mpk}, \mathsf{sk}'_f \coloneqq \mathsf{sk}_f, \mathsf{ct}' \coloneqq \mathsf{ct}$
If $\mathsf{VerifyCT}(\mathsf{mpk}, \mathsf{ct}) \neq 1$
    Return $\perp$
If $\mathsf{VerifySK}(\mathsf{mpk}, f, \mathsf{sk}_f) \neq 1$
    Return $\diamond$
$y \coloneqq \mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}_f, \mathsf{ct})$
Return y

Fig. 23: Strong input consistency compiler

which the intersection is empty and show that these cannot occur. For this purpose, we define the events E_1, E_2, E_3:

Event E_1 denotes the case that if $y_{j,i} \neq \perp$ for a single $j \in [n]$ and a fixed $i \in \{0,1\}$, then $y_{j,i} \neq \perp$ for all $j \in [n]$ and the same fixed $i \in \{0,1\}$. Event E_2 denotes the case that if $y_{j,i} \neq \diamond$ for a single $i \in \{0,1\}$ and a fixed $j \in [n]$, then $y_{j,i} \neq \diamond$ for both $i \in \{0,1\}$ and the same fixed $j \in [n]$. The final event, event E_3, denotes the case that if $y_{j,i} \notin \{\perp, \diamond\}$, then $y_{j,i} = f_j(x_i)$.

We start by analyzing event E_1. Let one of the $y_{j,i} = \perp$ for a single $j \in [n]$ and a fixed $i \in \{1,2\}$. Now, if $\mathsf{VerifyCT}$ was satisfied by ct_i, then by the verifiability condition, $\perp$ is a valid output and thus by definition of the special symbol $\perp$, the only "preimage" explaining the output is $\perp$ and all the decryptions of ct_i under different functional keys would lead the same output. Therefore, under the above assumption, all other $y_{j,i}$ must yield $\perp$.

In the other case, the $\mathsf{VerifyCT}$ algorithm outputs 0 for the ciphertext ct_i (for a fixed i). However, then the $\mathsf{VerifyCT}$ algorithm also outputs 0 on every other decrypt request since it is deterministic and does not depend on any functional key $\mathsf{sk}_{j'}$, $j' \in [n] \setminus \{j\}$. This leads to the fact that $y_{j,i} = \perp$ for all $j \in [n]$ and a fixed $i \in \{1,2\}$. In this case, the intersection contains the $\perp$ symbol.

For event E_2, let one of the $y_{j,i} = \diamond$ for a single $i \in \{1,2\}$ and a fixed $j \in [n]$. If this case occurs, then the $\mathsf{VerifySK}$ algorithm must have output 0 for the functional key sk_j for a fixed j as otherwise, since by definition of the special symbol $\diamond$, there is no "preimage" explaining the output (and thus the verifiability property violated). As before, the $\mathsf{VerifySK}$ algorithm also outputs 0 if it gets queried using another ciphertext $\mathsf{ct}_{i'}$ but the same functional key sk_j. This yields that $y_{j,i'} = \diamond$ for $i' \neq i$ and directly deletes the key sk_j from the list F (due to the definition of F).

Finally, we analyze event E_3. Let $y_{j,i} \notin \{\perp, \diamond\}$, then both of the verify algorithms, $\mathsf{VerifyCT}$ and $\mathsf{VerifySK}$, output 1 in the decryption procedure. This ensures, together with the verifiability property, i.e. $\Pr[\mathsf{Dec}(\mathsf{mpk}, f_j, \mathsf{sk}, \mathsf{ct}_i) = f_j(x_i)] = 1$, where f_j is the function associated with sk_j and x_i the plaintext associated with ct_i, that $y_{j,i} = f_j(x_i)$.

Taking into account the analysis of event E_1 and E_2, the decryption $y_{j',i}$, corresponding to a different functional key $\mathsf{sk}_{j'}$, is unequal to $\diamond$ (due to event E_4 and the definition of F) and unequal to $\perp$ (due to event E_1 and the fact that $y_{j,i}$ is a valid decryption). By doing the same analysis for $y_{j',i}$ as for $y_{j,i}$ as in event E_2, we obtain that $y_{j',i} = f_{j'}(x_i)$. We can do the same analysis for all the remaining $y_{j'',i}$ with $j'' \in [n] \setminus \{j, j'\}$ and therefore it follows that $\bigcap_{j \in [n], (\cdot, f_j) \in F} f_j^{-1}(y_{j,i}) \neq \emptyset$. The same analysis also needs to be done for the second ciphertext $\mathsf{ct}_{i'}$ with $i' \neq i$.

This shows that event E cannot occur and therefore that the proposed construction achieves strong input consistency. □

By showing the relation above, our treatment nicely includes the verifiability property and hence the analysis in the next section also gives a UC interpretation of the construction (achieved by both strong input consistency and verifiability).

In comparison, our strong input consistency notion is more positioned as a property that an attacker tries to break, rather than a general requirement of a scheme that holds for all arguments and follows directly as a strengthening of input consistency.

While the verifiability property is also necessary for our concrete compiler (as long as Dec is deterministic), strong input consistency does not achieve the verifiability property on its own. The reason for this is that the definition of strong input consistency does not change the FE syntax, verification checks are inherently done by Dec with access to at least one secret key. Thus, no guaranteed verifiability algorithm of the form $\mathsf{VerifyCT}(\mathsf{mpk}, \mathsf{ct})$ (or $\mathsf{VerifySK}(\mathsf{mpk}, f, \mathsf{sk}_f)$) can be directly deduced from a generic FE scheme that is strongly input-consistent according to our notion. Hence, our notion puts forth a seemingly weaker form of (implicit) secret-key verifiability.

Also, when leaving the standard model (e.g. switching to the random-oracle model), Definition 19 could technically be violated even though *finding* these values by an efficient adversary $\mathcal{A}$—which the strong input consistency definition asks for—might be infeasible. We further observe as an interesting open problem, whether strong input consistent (and also verifiable) FE schemes exist that satisfy CCA or CFE security (e.g., in the random-oracle model).

Appendix G
Details on the Setup Consistency Compilers

A. Proof of Theorem 5

Proof. To prove the setup consistency of the scheme FE', we rely on the soundness of the NIWI proof system. In more detail, we construct an adversary $\mathcal{B}$ that successfully generates a valid proof for a statement not in the language by assuming an adversary $\mathcal{A}$ for the setup consistency experiment set-CONS.

In the beginning of the reduction, $\mathcal{B}$ receives $(\mathsf{mpk}' := (\mathsf{mpk}_1, \mathsf{mpk}_2, \mathsf{mpk}_3), \mathsf{mpk}'' := (\mathsf{mpk}'_1, \mathsf{mpk}'_2, \mathsf{mpk}'_3), \mathsf{sk} := (\{\mathsf{sk}_i\}_{i\in[3]}, \pi), f, x_1, x_2)$ from $\mathcal{A}$. If π is such that the condition $\mathsf{NIWI.Verify}(1^\lambda, (\{\mathsf{mpk}''_i\}_{i\in[3]}, \{\mathsf{sk}_i\}_{i\in[3]}, f), \pi) = 0$ or $\mathsf{mpk}'' \neq \mathsf{mpk}'$, then $\mathcal{B}$ halts. Note that in this case, $\mathcal{A}$ would never win, as the outcome of decryption procedure is equal to "$\diamond$" for all ciphertexts. Another case, in which $\mathcal{B}$ halts, is the case in which $\mathsf{Dec}'(\mathsf{mpk}'', f, \mathsf{sk}, \mathsf{Enc}(\mathsf{mpk}', x_i)) = f(x_i)$ for both $i \in [2]$. Here, the generated functional key and the public parameters have an honest behavior and therefore $\mathcal{A}$ has not generated a forgery for the NIWI proof. Therefore the adversary $\mathcal{B}$ halts. Otherwise, the adversary outputs the statement $(\{\mathsf{mpk}''_i\}_{i\in[3]}, \{\mathsf{sk}_i\}_{i\in[3]}, f)$ and proof π as a NIWI forgery.

Let us analyze the output of $\mathcal{A}$ to see that the condition to break setup consistency must imply a soundness violation of the NIWI scheme. In more detail, we define the event E as the event that the adversary $\mathcal{A}$ performs a consistency attack under the assumption that $(\{\mathsf{mpk}''_i\}_{i\in[3]}, \{\mathsf{sk}_i\}_{i\in[3]}, f) \in L$ and show that the occurrence of E would contradict the assumption. Let us compute $\mathsf{ct}'_i := (\mathsf{mpk}', \{\mathsf{ct}_j\}_{j\in[3]}) \leftarrow \mathsf{Enc}'(\mathsf{mpk}', x_i)$ for $i \in [2]$ and $y_i = \mathsf{Dec}'(\mathsf{mpk}'', f, \mathsf{sk}, \mathsf{ct}'_i)$ for $i \in [2]$. For concreteness, assume that both ciphertexts are not equal to err (however, the argument holds for any pattern, since erroneous ciphertexts are ignored in the setup consistency game).

Now, we analyze the possible outcomes for the decryptions y_1 and y_2 in the case of a consistency attack. We show that $y_i \neq \diamond$, we denote this by event E_1, enforces that $y_i = f(x_i)$ for all $i \in [2]$ and furthermore that $y_i \neq \diamond$ for all $i \in [2]$.

In the case of event E_1, we assume $y_1 \neq \diamond$ (we do the analysis for y_1, the case for y_2 follows respectively) and, for the sake of contradiction, we also need to assume that $(\{\mathsf{mpk}''_i\}_{i\in[3]}, \{\mathsf{sk}_i\}_{i\in[3]}, f) \in L$ holds. Under these circumstances consistency must be satisfied.

By the perfect correctness of the underlying FE scheme and the validity of the proof, at least two functional keys sk_i and sk_j, for $i \neq j$ are correctly generated and matching to the master public keys mpk_i and mpk_j in the encryption (note that by the definition of Dec' that decryption is only performed if both Enc' and Dec' use the same triple $(\mathsf{mpk}_1, \mathsf{mpk}_2, \mathsf{mpk}_3)$) and thus $y^{(i)}_{f,1} \leftarrow \mathsf{Dec}(\mathsf{mpk}_i, f, \mathsf{sk}_i, \mathsf{ct}_1)$ and $y^{(j)}_{f,1} \leftarrow \mathsf{Dec}(\mathsf{mpk}_j, f, \mathsf{sk}_j, \mathsf{ct}_1)$ are equal to $f(x_1)$. Therefore also the majority of the decryption values for ct_1 is equal to $f(x_1)$ and the final decryption outputs $f(x_1)$. Hence assuming $y_1 \neq \diamond$ implies $y_1 = f(x_1)$.

For event E_2, we need to show that $y_1 \neq \diamond$ and $y_2 \neq \diamond$ in the case of a consistency attack and under the assumption that $(\{\mathsf{mpk}''_i\}_{i\in[3]}, \{\mathsf{sk}_i\}_{i\in[3]}, f) \in L$. We start by considering the case that $y_1 = y_2 = \diamond$. If $y_1 = y_2 = \diamond$, then the adversary $\mathcal{A}$ did not perform a consistency attack. This is a contradiction to our assumption and therefore this case cannot occur. In the next step we assume that $y_1 \neq \diamond$ and $y_2 = \diamond$ (or $y_1 = \diamond$ and $y_2 \neq \diamond$ respectively). If $y_1 \neq \diamond$, then follows, with the analysis for E_1, that $y_1 = f(x_1)$ and that at least two of the functional keys sk_i and sk_j are correctly generated and matching the master public keys mpk_i and mpk_j. But this would also lead, due to perfect correctness of the functional encryption scheme, to a correct decryption of the ciphertext ct'_2, which yields $y_2 = f(x_2)$. This shows that the case $y_1 \neq \diamond$ and $y_2 = \diamond$ (or $y_1 = \diamond$ and $y_2 \neq \diamond$) cannot occur.

By combining the events E_1 and E_2, we proved that event E cannot occur. To recap, whenever $(\{\mathsf{mpk}''_i\}_{i\in[3]}, \{\mathsf{sk}_i\}_{i\in[3]}, f) \in L$, it is not possible for an adversary $\mathcal{A}$ to perform a consistency attack. Hence the only way the adversary can break setup consistency is by breaking the soundness property of the NIWI scheme, i.e., providing the statement $(\{\mathsf{mpk}''_i\}_{i\in[3]}, \{\mathsf{sk}_i\}_{i\in[3]}, f) \notin L$.

This yields the bound

$$|\Pr[\text{set-CONS}^{\mathsf{FE}'}(1^\lambda, \mathcal{A}) = 1]| \leq \mathsf{Adv}^{\text{Sound}}_{\mathsf{NIWI},\mathcal{B}}(\lambda) \ .$$

and therefore we obtain the theorem. □

B. Security Preservation of the First Compiler

We start with the CPA case, which is straightforward:

Theorem 22. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an IND-CPA secure functional encryption scheme and* $\mathsf{NIWI} = (\mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$ *a NIWI proof system for* R_{set} *(Fig. 9), then the construction* FE' *defined in Fig. 8 is IND-CPA secure. Namely, for any PPT adversary* $\mathcal{A}$*, there exits a PPT adversaries* $\mathcal{B}$ *and* $\mathcal{B}'$ *such that:*

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE}',\mathcal{A}}(\lambda) \leq 3 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}}(\lambda) + 2 \cdot \mathsf{Adv}^{\text{WI}}_{\mathsf{NIWI},\mathcal{B}'}(\lambda) \ .$$

Proof. To prove this statement, we use a hybrid argument with the games defined in Fig. 24. Note that G_0 corresponds to the game $\text{IND-CPA}^{\mathsf{FE}}_0(1^\lambda, \mathcal{A})$ and G_5 to the game $\text{IND-CPA}^{\mathsf{FE}}_1(1^\lambda, \mathcal{A})$. This results in:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE}',\mathcal{A}}(1^\lambda) = |\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_5}_{\mathcal{A}}(1^\lambda)| \ .$$

We describe the different games in more detail:

Game G_0: This game is the $\text{IND-CPA}^{\mathsf{FE}}_0(1^\lambda, \mathcal{A})$ game. We assume, without loss of generality, that the challenger uses the indices $j_1 = 2$ and $j_2 = 3$ for the generation of the NIWI proof in the key generation procedure.

Game G_1: In this game, we change the encryption under the first master public key mpk_1 from x^0 to x^1. The

transition from G_0 to G_1 is justified by the IND-CPA security of FE. Namely, in Lemma 7, we exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_0}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_1}(1^\lambda)| \leq \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_0}(\lambda) \ .$$

Game G_2: In this game, we change the indices that are used in the generation of the NIWI proof in the key generation procedure from $j_1 = 2$ and $j_2 = 3$ to $j_1 = 1$ and $j_2 = 3$. The transition from G_0 to G_1 is justified by the witness-hiding property of NIWI. Namely, in Lemma 8, we exhibit a PPT adversary $\mathcal{B}_1$ such that:

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_1}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_2}(1^\lambda)| \leq \mathsf{Adv}^{\text{WI}}_{\mathsf{NIWI},\mathcal{B}_1}(\lambda) \ .$$

Game G_3: In this game, we change the encryption under the second master public key mpk_2 from x^0 to x^1. The transition from G_2 to G_3 is almost symmetric to the transition from game G_0 to G_1 except that the ciphertext under the first master public key mpk_1 contains an encryption of x^1. As in Lemma 7, the transition is justified by the IND-CPA security of FE. Namely, we can exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_2}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_3}(1^\lambda)| \leq \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_0}(\lambda) \ .$$

Game G_4: In this game, we change the indices that are used in the generation of the NIWI proof in the key generation procedure from $j_1 = 1$ and $j_2 = 3$ to $j_1 = 1$ and $j_2 = 2$. The transition from G_3 to G_4 is almost symmetric to the transition from game G_1 to G_2 except that the ciphertexts under the first two master public keys mpk_1 and mpk_2 contain encryptions of x^1. As in Lemma 8, the transition is justified by the witness-hiding property of NIWI. Namely, we can exhibit a PPT adversary $\mathcal{B}_1$ such that:

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_3}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_4}(1^\lambda)| \leq \mathsf{Adv}^{\text{WI}}_{\mathsf{NIWI},\mathcal{B}_1}(\lambda) \ .$$

Game G_5: This game is the $\text{IND-CPA}_1^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ game, where the challenger uses the indices $j_1 = 1$ and $j_2 = 2$ for the generation of the NIWI proof in the key generation procedure. The transition from G_4 to G_5 is almost symmetric to the transition from game G_0 to G_1 except that the ciphertexts under the first two master public keys mpk_1 and mpk_2 contain encryptions of x^1. As in Lemma 7, the transition is justified by the IND-CPA security of FE. Namely, we can exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_4}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_5}(1^\lambda)| \leq \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_0}(\lambda) \ .$$

Putting everything together, we obtain the theorem. □

Lemma 7 (Transition from G_0 to G_1)**.** *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_0$ such that*

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_0}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_1}(1^\lambda)| \leq \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_0}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_0$ that simulates G_β towards $\mathcal{A}$ when interacting with the underlying $\text{IND-CPA}_\beta^{\mathsf{FE}}$ experiment.

In the beginning of the reduction, $\mathcal{B}_0$ receives mpk_1 from the experiment. It generates two other functional encryption instances $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda)$, for $i \in [3] \setminus \{1\}$, sets $\mathsf{mpk}' = \{\mathsf{mpk}_i\}_{i\in[3]}$ and gives mpk' to the adversary.

Whenever $\mathcal{A}$ asks an encryption query (x^0, x^1), $\mathcal{B}_0$ forwards it to its own encryption oracle to receive $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{mpk}_1, x^\beta)$, generates $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, x^0)$, for $i \in [3] \setminus \{1\}$ on its own and sends $\mathsf{ct}' = \{\mathsf{ct}_i\}_{i\in[3]}$ to $\mathcal{A}$.

For a key generation query f, $\mathcal{B}_0$ queries its own key generation oracle on f to receive $\mathsf{sk}_{f,1} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_1, \mathsf{msk}_1, f)$, generates $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f)$ for $i \in [3] \setminus \{1\}$ on its own and generates a proof $\pi \leftarrow \mathsf{NIWI.Prove}(1^\lambda, z, w)$ with $z = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3]}, f), w = (\{\mathsf{msk}_i\}_{i\in[3]\setminus\{1\}}, \{r_i\}_{i\in[3]\setminus\{1\}}, \{s_i\}_{i\in[3]\setminus\{1\}})$ for the relation R_{set}, by using its information of two-out-of-the-three different instances. As a reply for the key generation query, $\mathcal{B}_0$ sends $\mathsf{sk}'_f := (\{\mathsf{sk}_{f,i}\}_{i\in[3]}, \pi)$ to $\mathcal{A}$.

This covers the simulation of the game G_β. Finally $\mathcal{B}_0$ outputs the same bit β' returned by $\mathcal{A}$. It follows, from the perfect simulation, that the advantage of $\mathcal{B}_0$ is the same as the advantage $\mathcal{A}$. □

Lemma 8 (Transition from G_1 to G_2)**.** *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_1$ such that*

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_1}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_2}(1^\lambda)| \leq \mathsf{Adv}^{\text{WI}}_{\mathsf{NIWI},\mathcal{B}_1}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_1$ that simulates $\mathsf{G}_{1+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\text{WI}_\beta^{\mathsf{NIWI}}$ experiment.

In the beginning of the reduction, $\mathcal{B}_1$ samples $s_i \leftarrow \{0,1\}^\lambda$, generates the keys for the three functional encryption instances $(\mathsf{mpk}_i, \mathsf{msk}_i) := \mathsf{Setup}(1^\lambda; s_i)$, for $i \in [3]$, sets $\mathsf{mpk}' = \{\mathsf{mpk}_i\}_{i\in[3]}$, saves $\{s_i\}_{i\in[3]}$ and gives mpk' to the adversary.

Whenever $\mathcal{A}$ asks an encryption query (x^0, x^1), $\mathcal{B}_1$ encrypts x^1 using the first public key and x^0 using the second and third public key, i.e $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{mpk}_1, x^1)$ and $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, x^0)$, for $i \in [3] \setminus \{1\}$. Afterwards, it sends $\mathsf{ct}' = \{\mathsf{ct}_i\}_{i\in[3]}$ to $\mathcal{A}$.

For a key generation query f, $\mathcal{B}_1$ samples $r_i \leftarrow \{0,1\}^\lambda$ and generates $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f; r_i)$ for $i \in [3]$. Afterwards, $\mathcal{B}_1$ submits (z, w_0, w_1) with $z = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3}, f), w_0 = (\{\mathsf{msk}_i\}_{i\in[3]\setminus\{1\}}, \{r_i\}_{i\in[3]\setminus\{1\}}, \{s_i\}_{i\in[3]\setminus\{1\}})$ and $w_1 = (\{\mathsf{msk}_i\}_{i\in[3]\setminus\{2\}}, \{r_i\}_{i\in[3]\setminus\{2\}}, \{s_i\}_{i\in[3]\setminus\{2\}})$ as a challenge query to its challenger and receives π as a reply. Finally, $\mathcal{B}_1$ sends $(\{\mathsf{sk}_{f,i}\}_{i\in[3]}, \pi)$ to $\mathcal{A}$ as a reply to the key generation query.

This covers the simulation of the game $\mathsf{G}_{1+\beta}$. Finally $\mathcal{B}_1$ outputs the same bit β' returned by $\mathcal{A}$. It follows, from the perfect simulation, that the advantage of $\mathcal{B}_1$ is the same as the advantage $\mathcal{A}$. □

Game	ct	NIWI Witness	justification/remark
G_0	$\mathsf{Enc}(\mathsf{mpk}_1, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 2$ $j_2 = 3$	
G_1	$\mathsf{Enc}(\mathsf{mpk}_1, \boxed{x^1})$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 2$ $j_2 = 3$	IND-CPA security of FE
G_2	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$\boxed{j_1 = 1}$ $j_2 = 3$	Witness-hiding of NIWI
G_3	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, \boxed{x^1})$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 1$ $j_2 = 3$	IND-CPA security of FE
G_4	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 1$ $\boxed{j_2 = 2}$	Witness-hiding of NIWI
G_5	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_3, \boxed{x^1})$	$j_1 = 1$ $j_2 = 2$	IND-CPA security of FE

Fig. 24: Overview of the games to prove the IND-CPA security preservation of the setup consistency compiler described in Fig. 8.

Due to the much stronger simulation-based security requirement of CFE, the existence of simulators $\mathcal{S}_1$ (for setup generation), that outputs a simulated mpk and an initial (joint) state $s = s_{\mathsf{init}}$, $\mathcal{S}_2$ for the simulation of functional keys (based on the joint state s which might be update in this process), and finally $\mathcal{S}_3$ for simulating ciphertexts (again with access to the joint state s) does formally not imply knowledge of a master secret key that would be needed to create valid proofs for the relation R_{set}. Hence, the theorem captures preservation only for a specific class of simulators and not all CFE secure schemes. We note that for the brute-force scheme in Fig. 16 in Section C there exists a simulator that belongs to the class we are proving the security preservation for.

More formally, we require $\mathcal{S}_1$ to output mpk and maintain state s such that $(\mathsf{mpk}, \mathsf{msk}) = \mathsf{Setup}(1^\lambda; s)$ holds with probability 1 in $\mathsf{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$. Additionally, we require that for any adversary $\mathcal{A}$ any functional key sk_f output by $\mathcal{S}_2$ on input f satisfies $\mathsf{sk}_f = \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$ with probability 1 in $\mathsf{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$ where mpk and msk are the values obtained by $\mathsf{Setup}(1^\lambda; s)$ where s is the initial private state output by $\mathcal{S}_1$.

Note that the simulator for the brute-force scheme described in [52] runs the normal setup-algorithm in the simulation (and can hence provide the randomness used during the generation) and the master secret key fixes all secret keys.

Theorem 23. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a CFE secure functional encryption scheme with respect to simulators* $(\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ *that satisfy the above condition in* $\mathsf{Ideal}^{\mathsf{FE}}(1^\lambda, \mathcal{A}, \mathcal{S})$ *(for any adversary* $\mathcal{A}$*). Let further* $\mathsf{NIWI} = (\mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$ *be a NIWI proof system for* R_{set} *(Fig. 9). Under the assumption that* KeyGen *is deterministic, the construction* FE' *defined in Figure 8 is CFE secure.*

Proof. Under the theorem's assumptions, the simulator $\mathcal{S}_1$'s output is essentially equivalent to the master secret key. Together with the fact that key derivation is deterministic, we see that all NIWI proofs can be generated without problem. More detailed, we can run three independent simulations of the FE scheme for the overall simulation. That is, let $\mathcal{S}_1$,$\mathcal{S}_2$,$\mathcal{S}_3$ be the simulators for FE. Then the composite simulator $\mathcal{S}' = (\mathcal{S}'_1, \mathcal{S}'_2, \mathcal{S}'_3)$ works as follows: $\mathcal{S}'_1$ runs $\mathcal{S}_1$ three times to obtain (mpk_i, s_i). To answer key-generation queries for functions f, $\mathcal{S}'_2$ runs $\mathcal{S}_2$ three times on the respective joint state (and the function values of all previous queries) to obtain $\mathsf{sk}_{f,i}$. Note that by the assumption on the simulation for the underyling scheme, for $(\mathsf{mpk}_i, \mathsf{msk}_i) = \mathsf{Setup}(1^\lambda; s_i)$ we have that $\mathsf{sk}_{f,i} = \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f)$. By the theorem assumption, KeyGen is deterministic and hence we have all witnesses to simulate a genuine proof $\pi \leftarrow \mathsf{NIWI.Prove}(1^\lambda, z, w)$ with $z = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3]}, f)$ and $w = (\{s_i\}_{i\in[3]}, \{r_i\}_{i\in[3]})$. Finally, simulating a ciphertext is done by invoking $\mathcal{S}_3$ three times on all three simulated instances (and on the joint state and the function values of the actual plaintext). □

C. Advanced Setup Consistency Compiler

Theorem 24. *Let* $\mathsf{FE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an IND-CPA secure functional encryption scheme,* $\mathsf{NIWI} = (\mathsf{NIWI.Prove}, \mathsf{NIWI.Verify})$ *a NIWI proof system and* $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify})$ *a NIZK proof system system satisfying one-time simulation soundness, then the construction* FE' *defined in Figure 25 is IND-CCA secure. Namely, for any PPT adversary* $\mathcal{A}$*, there exist PPT adversaries* $\mathcal{B}$*,* $\mathcal{B}'$*,* $\mathcal{B}''$ *and* $\mathcal{B}'''$ *such that:*

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{FE}',\mathcal{A}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\text{ZK}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + 5 \cdot \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}'}(\lambda) + 3 \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}''}(\lambda).$$

Proof. To prove this statement, we use a hybrid argument with the games defined in Fig. 27. Note that G_0 corresponds to the game $\text{IND-CCA}^{\mathsf{FE}}_0(1^\lambda, \mathcal{A})$ and G_7 to the game $\text{IND-CCA}^{\mathsf{FE}}_1(1^\lambda, \mathcal{A})$. This results in:

$$\mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{FE}',\mathcal{A}}(1^\lambda) = |\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_7}_{\mathcal{A}}(1^\lambda)| \ .$$

We describe the different games in more detail:

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

$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$

For $i \in [3]$:

$(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda; s_i)$ with $s_i \leftarrow \{0,1\}^\lambda$

Return $(\mathsf{mpk}', \mathsf{msk}')$

$= ((\{\mathsf{mpk}_i\}_{i\in[3]}, \mathsf{CRS}), \{(\mathsf{msk}_i, s_i)\}_{i\in[3]})$

$\mathsf{KeyGen}'(\mathsf{mpk}', \mathsf{msk}', f)$:

Parse $\mathsf{mpk}' := (\{\mathsf{mpk}_i\}_{i\in[3]}, \mathsf{CRS})$,

$\mathsf{msk}' := \{(\mathsf{msk}_i, s_i)\}_{i\in[3]}$

For $i \in [3]$:

$\mathsf{sk}_{f,i} = \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f; r_i)$ with $r_i \in \{0,1\}^\lambda$

Generate $\pi_{\mathsf{sk}} \leftarrow \mathsf{NIWI.Prove}(1^\lambda, z, w)$ with

$z = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3]}, f)$

$w = (\{\mathsf{msk}_i\}_{i\in[3]}, \{r_i\}_{i\in[3]}, \{s_i\}_{i\in[3]})$

where L is defined corresponding to R_{set} (Fig. 9)

Return $\mathsf{sk}'_f = (\{\mathsf{sk}_{f,i}\}_{i\in[3]}, \pi_{\mathsf{sk}})$

$\mathsf{Enc}'(\mathsf{mpk}', x)$:

Parse $\mathsf{mpk}' := (\{\mathsf{mpk}_i\}_{i\in[3]}, \mathsf{CRS})$

For $i \in [3]$:

$\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i, x; u_i)$ with $u_i \in \{0,1\}^\lambda$

If $\exists i \in [3] : \mathsf{ct}_i = \mathsf{err}$ return err

$\pi_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[3]}, (x, \{u_i\}_{i\in[3]}))$,

for $R^{\mathsf{CCA}}_{\mathsf{set}}$ (Fig. 26)

If $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[3]}, \pi_{\mathsf{ct}}) = 0$ return err

Return $\mathsf{ct}' = (\mathsf{mpk}', \{\mathsf{ct}_i\}_{i\in[3]}, \pi_{\mathsf{ct}})$

$\mathsf{Dec}'(\mathsf{mpk}', \mathsf{sk}'_f, f, \mathsf{ct}')$:

Parse $\mathsf{mpk}' := \{\mathsf{mpk}_i\}_{i\in[3]}, \mathsf{sk}'_f := (\{\mathsf{sk}_{f,i}\}_{i\in[3]}, \pi_{\mathsf{sk}})$,

$\mathsf{ct}' := (\mathsf{mpk}'', \{\mathsf{ct}_i\}_{i\in[3]}, \pi_{\mathsf{ct}})$

If $\mathsf{mpk}' = \mathsf{mpk}''$

If $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[3]}, \pi_{\mathsf{ct}}) = 1$

If $\mathsf{NIWI.Verify}(1^\lambda, (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3]}, f), \pi_{\mathsf{sk}}) = 1$

$y_{f,i} := \mathsf{Dec}(\mathsf{mpk}_i, \mathsf{sk}_{f,i}, f, \mathsf{ct}_i)$, for $i \in [3]$:

If there are indices $a, b \in [3], a \neq b$ s.t.

$y_{f,a} = y_{f,a}$

Return $y \leftarrow \mathrm{MajVal}(y_{f,1}, y_{f,2}, y_{f,3})$

Return $\diamond$

Fig. 25: Advanced setup consistency compiler. $\mathrm{MajVal}(\cdot)$ calculates and returns the majority value of the input values, if there is a clear majority and $\diamond$ otherwise. Shaded instructions again indicate the difference to the simpler setup compiler.

Relation $R^{\mathsf{CCA}}_{\mathsf{set}}$:

Instance: $z = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{ct}_i\}_{i\in[3]})$

Witness: $w = (x, \{u_i\}_{i\in[3]})$

$R^{\mathsf{CCA}}_{\mathsf{set}}(z, w) = 1$ if and only if:

$\mathsf{ct}_i = \mathsf{Enc}(\mathsf{mpk}_i, x; u_i)$ for all $i \in [3]$

Fig. 26: Relation used in the advanced setup consistency compiler

Game G_0: This game is the $\text{IND-CCA}^{\mathsf{FE}}_0(1^\lambda, \mathcal{A})$ game. We assume without loss of generality that the challenger uses the indices $j_1 = 2$ and $j_2 = 3$ for the generation of the NIWI proof in the key generation procedure.

Game G_1: In this game, we change from an honestly generated CRS and honestly generated proofs to a simulated CRS and simulated proofs. The transition from G_0 to G_1 is justified by the zero-knowledge property of NIZK. Namely, in Lemma 9, we exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Game G_2: In this game, we change the encryption under the first master public key mpk_1 from x^0 to x^1. The transition from G_1 to G_2 is justified by the IND-CPA security of FE and the one-time simulation-soundness of NIZK. Namely, in Lemma 10, we exhibit PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_2}(\lambda) \ .$$

Game G_3: In this game, we change the indices that are used in the generation of the NIWI proof in the key generation procedure from $j_1 = 2$ and $j_2 = 3$ to $j_1 = 1$ and $j_2 = 3$. The transition from G_2 to G_3 is justified by the witness-hiding property of NIWI and the one-time simulation-soundness of NIZK. Namely, in Lemma 11, we exhibit PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_3$ such that:

$$|\mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{WI}}_{\mathsf{NIWI},\mathcal{B}_3}(\lambda) \ .$$

Game G_4: In this game, we change the encryption under the second master public key mpk_2 from x^0 to x^1. The transition from G_3 to G_4 is almost symmetric to the transition from game G_1 to G_2 except that the ciphertext under the first master public key mpk_1 contains an encryption of x^1. As in Lemma 10, the transition is justified by the IND-CPA security of FE and the one-time simulation-soundness of NIZK. Namely, we can exhibit PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:

Game	CRS & π	ct	NIWI Witness	justification/remark
G_0	$\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 2$ $j_2 = 3$	
G_1	$\boxed{\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)}$ $\boxed{\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)}$	$\mathsf{Enc}(\mathsf{mpk}_1, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 2$ $j_2 = 3$	Zero-knowledge of NIZK
G_2	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}_1, \boxed{x^1})$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 2$ $j_2 = 3$	IND-CPA of FE and one-time simulation-soundness of NIZK
G_3	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^0)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$\boxed{j_1 = 1}$ $j_2 = 3$	Witness-hiding of NIWI and one-time simulation-soundness of NIZK
G_4	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, \boxed{x^1})$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 1$ $j_2 = 3$	IND-CPA of FE and one-time simulation-soundness of NIZK
G_5	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^0)$	$j_1 = 1$ $\boxed{j_2 = 2}$	Witness-hiding of NIWI and one-time simulation-soundness of NIZK
G_6	$\mathsf{CRS} \leftarrow \mathcal{S}_1(1^\lambda)$ $\pi \leftarrow \mathcal{S}_2(\mathsf{CRS}, \tau, x)$	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_3, \boxed{x^1})$	$j_1 = 1$ $j_2 = 2$	IND-CPA of FE and one-time simulation-soundness of NIZK
G_7	$\boxed{\mathsf{CRS} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)}$ $\boxed{\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{CRS}, x, w)}$	$\mathsf{Enc}(\mathsf{mpk}_1, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_2, x^1)$ $\mathsf{Enc}(\mathsf{mpk}_3, x^1)$	$j_1 = 1$ $j_2 = 2$	Zero-knowledge of NIZK

Fig. 27: Overview of the games to prove the IND-CCA security of the advanced setup consistency compiler described in Fig. 25.

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_3}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_4}(1^\lambda)| \leq \mathsf{Adv}_{\mathsf{NIZK},\mathcal{B}_1}^{\text{Sim-Sound}}(\lambda) + \mathsf{Adv}_{\mathsf{FE},\mathcal{B}_2}^{\text{IND-CPA}}(\lambda) \ .$$

$$|\mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_4}(1^\lambda) - \mathsf{Win}_{\mathcal{A}}^{\mathsf{G}_5}(1^\lambda)| \leq \mathsf{Adv}_{\mathsf{NIZK},\mathcal{B}_1}^{\text{Sim-Sound}}(\lambda) + \mathsf{Adv}_{\mathsf{NIWI},\mathcal{B}_3}^{\text{WI}}(\lambda) \ .$$

Game G_5: In this game, we change the indices that are used in the generation of the NIWI proof in the key generation procedure from $j_1 = 1$ and $j_2 = 3$ to $j_1 = 1$ and $j_2 = 2$. The transition from G_4 to G_5 is almost symmetric to the transition from game G_2 to G_3 except that the ciphertexts under the first two master public keys mpk_1 and mpk_2 contain encryptions of x^1. As in Lemma 11, the transition is justified by the witness-hiding property of NIWI and the one-time simulation-soundness of NIZK. Namely, we can exhibit PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_3$ such that:

Game G_6: In this game, we change the encryption under the third master public key mpk_3 from x^0 to x^1. The transition from G_5 to G_6 is almost symmetric to the transition from game G_1 to G_2 except that the ciphertext under the first two master public keys mpk_1 and mpk_2 contains encryptions of x^1. As in Lemma 10, the transition is justified by the IND-CPA security of FE and the one-time simulation-soundness of NIZK. Namely, we can exhibit PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that:

$$|\mathsf{Win}^{\mathsf{G}_5}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_6}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_2}(\lambda) \ .$$

Game G_7: This game is the IND-CCA$_1^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ game, where the challenger uses the indices $j_1 = 1$ and $j_2 = 2$ for the generation of the NIWI proof in the key generation procedure. The transition from G_6 to G_7 is almost symmetric to the transition from G_0 to G_1 except from the fact that the reduction encrypts x^1 instead of x^0. As in Lemma 9, the transition is justified by the zero-knowledge property of NIZK. Namely, we can exhibit a PPT adversary $\mathcal{B}_0$ such that:

$$|\mathsf{Win}^{\mathsf{G}_6}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_7}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Putting everything together, we obtain the theorem. □

Lemma 9 (Transition from G_0 to G_1)**.** *For any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{B}_0$ such that*

$$|\mathsf{Win}^{\mathsf{G}_0}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NIZK},\mathcal{B}_0}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_0$ that simulates G_β towards $\mathcal{A}$ when interacting with the underlying $\mathrm{ZK}^{\mathsf{NIZK}}_\beta$ experiment.

In the beginning of the reduction, $\mathcal{B}_0$ receives CRS from the $\mathrm{ZK}^{\mathsf{NIZK}}_\beta$ experiment. It generates three functional encryption instances $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda; s_i)$ with $s_i \leftarrow \{0,1\}^\lambda$ for $i \in [3]$, sets $\mathsf{mpk}' = (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[3]})$ and gives mpk' to the adversary.

Whenever $\mathcal{A}$ asks an encryption query (x^0, x^1), $\mathcal{B}_0$ generates the ciphertext $\mathsf{ct}_i = (\mathsf{Enc}(\mathsf{mpk}_i, x^0; u_i))_{i\in[3]}$ with $u_i \leftarrow \{0,1\}^\lambda$ for $i \in [3]$ and sends $y = (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[3]}$ and $w = (x, \{u_i\}_{i\in[3]})$ as a statement-witness pair to its challenger. As an answer, $\mathcal{B}_0$ receives a proof π_{ct} for the relation $R^{\mathsf{CCA}}_{\mathsf{set}}$. It sets $\mathsf{ct}' = (\{\mathsf{ct}_i\}_{i\in[3]}, \pi_{\mathsf{ct}})$ and sends it to $\mathcal{A}$.

For a key generation query f, $\mathcal{B}_0$ generates $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f; r_i)$ with $r_i \leftarrow \{0,1\}^\lambda$ for $i \in [3]$ and creates a NIWI proof π_{sk} over the relation R_{set} for the statement-witness pair $y = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3]}, f)$ and $w = (\{\mathsf{msk}_i\}_{i\in[3]}, \{r_i\}_{i\in[3]}, \{s_i\}_{i\in[3]})$. Then, $\mathcal{B}_0$ sends $\mathsf{sk}'_f = (\{\mathsf{sk}_{f,i}\}_{i\in[3]}, \pi_{\mathsf{sk}})$ as a reply to $\mathcal{A}$.

Whenever $\mathcal{A}$ submits a decryption query $(f, \mathsf{ct}' = (\{\mathsf{ct}_i\}_{i\in[2]}, \pi_{\mathsf{ct}}))$, $\mathcal{B}_0$ generates the functional keys $\mathsf{sk}_{i,f} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f)$ for $i \in [3]$ and executes $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[3]})$. If $\mathsf{NIZK.Verify}(\mathsf{CRS}, (\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[2]}) = 1$, $\mathcal{B}_0$ computes $y_{f,i} := \mathsf{Dec}(\mathsf{mpk}_i, f, \mathsf{sk}_{f,i}, \mathsf{ct}_i)$ for $i \in [3]$ and sends the majority vote, $y \leftarrow \mathrm{MajVal}(y_{f,1}, y_{f,2}, y_{f,3})$, to $\mathcal{A}$. If the verification outputs 0, $\mathcal{B}_0$ sends $\perp$ to $\mathcal{A}$.

This covers the simulation of the game G_β. Finally $\mathcal{B}_0$ outputs the same bit β' returned by $\mathcal{A}$. It follows, from the perfect simulation, that the advantage of $\mathcal{B}_0$ is the same as the advantage of $\mathcal{A}$. □

As in [51] and in the proof of Theorem 19, we prove a claim that shows that whenever a decryption oracle query is asked and this query contains a valid NIZK proof, then the corresponding ciphertext is explainable under the queried function. This is necessary for the proof of the transition from G_1 to G_2 for the simulation of the decryption oracle.

Claim 2. *For any PPT adversary $\mathcal{A}$ participating in G_k for $k \in \{1, \ldots, 6\}$, the probability that, during the experiment, $\mathcal{A}$ queries its decryption oracle* QDec *with a function-ciphertext-pair that is not explainable but has an accepting proof is negligible. Namely, we exhibit a PPT adversary $\mathcal{B}_1$, such that*

$$\Pr\left[\begin{array}{l} \exists (f, \{\mathsf{ct}'_i\}_{i\in[3]}, \pi') \in Q : \\ (\{\mathsf{ct}'_i\}_{i\in[3]}, \pi') \neq (\{\mathsf{ct}'_i\}_{i\in[3]}, \pi'), \\ \mathsf{NIZK.Verify}(\mathsf{CRS}, \{\mathsf{ct}'_i\}_{i\in[3]}, \pi') = 1 \\ \textit{and for all } i, j \in [3], i \neq j : \\ \mathsf{Dec}(\mathsf{mpk}_i, \mathsf{sk}_{f,i}, \mathsf{ct}'_i) \neq \mathsf{Dec}(\mathsf{mpk}_j, \mathsf{sk}_{f,j}, \mathsf{ct}'_j) \end{array}\right] \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda),$$

where $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f)$ for $i \in [3]$, $(\{\mathsf{ct}_i\}_{i\in[3]}, \pi)$ is the reply to the encryption query (x^0, x^1) made by $\mathcal{A}$ where $(\{\mathsf{ct}_i\}_{i\in[3]}, \pi)$ is the reply to the encryption query (x^0, x^1) made by $\mathcal{A}$ and Q the list containing all the decryption queries $(f, \{\mathsf{ct}'_i\}_{i\in[3]}, \pi')$ asked by $\mathcal{A}$, knowing the master public key $\mathsf{mpk}' := (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[2]})$, the reply to its challenge query $(\{\mathsf{ct}_i\}_{i\in[3]}, \pi)$ and by having access to the key generation oracle $\mathsf{KeyGen}'(\mathsf{mpk}', \mathsf{msk}', \cdot)$, during the game.

Proof. We build an adversary $\mathcal{B}_1$ that simulates G_k towards $\mathcal{A}$ when interacting with the underlying one-time simulation-soundness experiment.

After the adversary $\mathcal{B}_1$ has received CRS from the underlying experiment, it generates $(\mathsf{mpk}_i, \mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda; s_i)$ with $s_i \leftarrow \{0,1\}^\lambda$ for $i \in [3]$, sets $\mathsf{mpk}' := (\mathsf{CRS}, \{\mathsf{mpk}_i\}_{i\in[3]})$ and sends mpk' to $\mathcal{A}$. Whenever $\mathcal{A}$ submits a key generation query f, $\mathcal{B}_1$ generates the functional keys $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i, \mathsf{msk}_i, f; r_i)$ for $i \in [3]$ and creates a NIWI proof π_{sk} over the relation R_{set} for the statement-witness pair $y = (\{\mathsf{mpk}_i\}_{i\in[3]}, \{\mathsf{sk}_{f,i}\}_{i\in[3]}, f)$ and $w = (\{\mathsf{msk}_i\}_{i\in[3]}, \{r_i\}_{i\in[3]}, \{s_i\}_{i\in[3]})$. Then, $\mathcal{B}_1$ sends $\mathsf{sk}'_f = (\{\mathsf{sk}_{f,i}\}_{i\in[3]}, \pi_{\mathsf{sk}})$ as a reply to $\mathcal{A}$.

For the challenge query (x^0, x^1) asked by $\mathcal{A}$, $\mathcal{B}_1$ computes $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{mpk}_1, x^1)$ for $i \leq k$ and $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{mpk}_i, x^0)$ for $i > k$ and asks its experiment for a simulated proof π of the statement $(\mathsf{mpk}_i, \mathsf{ct}_i)_{i\in[3]}$. It sets $\mathsf{ct}' := (\{\mathsf{ct}_i\}_{i\in[3]}, \pi)$ and sends ct' to $\mathcal{A}$.

Whenever $\mathcal{A}$ outputs a decryption query $(f, \mathsf{ct}' := (\{\mathsf{ct}\}_{i\in[3]}, \pi))$, $\mathcal{B}_1$ verifies the proof. If the output of the verification is 1, $\mathcal{B}_1$ computes $y_{f,i} = \mathsf{Dec}(\mathsf{mpk}_i, \mathsf{sk}_{f,i}, \mathsf{ct}_i)$ for all $i \in [3]$. Then, $\mathcal{B}_1$ computes $y \leftarrow \mathrm{MajVal}(y_{f,1}, y_{f,2}, y_{f,3})$ and if $y = \diamond$ then $\mathcal{B}_1$ sends $(\{\mathsf{ct}\}_{i\in[3]}, \pi)$ as a proof forgery to its challenger. Otherwise it sends y to $\mathcal{A}$. If the verification outputs 0, $\mathcal{B}_1$ sends $\perp$ to $\mathcal{A}$. □

After introducing and proving Claim 2, we prove the

transition from G_1 to G_2

Lemma 10 (Transition from G_1 to G_2)**.** *For any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, such that*

$$|\mathsf{Win}^{\mathsf{G}_1}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{FE},\mathcal{B}_2}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_2$ that simulates $\mathsf{G}_{1+\beta}$ to $\mathcal{A}$ when interacting with the underlying IND-CPA$^{\mathsf{FE}}_\beta$ experiment.

In the beginning of the reduction, $\mathcal{B}_2$ receives mpk_1 from the underlying experiment. It simulates a CRS, i.e. $(\mathsf{CRS},\tau) \leftarrow \mathcal{S}_1(1^\lambda)$, generates several functional encryption instances $(\mathsf{mpk}_i,\mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda;s_i)$ with $s_i \in \{0,1\}^\lambda$ for $i \in [3] \setminus \{1\}$, sets $\mathsf{mpk}' := (\mathsf{CRS},\{\mathsf{mpk}_i\}_{i\in[3]})$ and sends mpk' to $\mathcal{A}$. Whenever $\mathcal{A}$ submits a key generation query f, $\mathcal{B}_2$ forwards this query to its own key generation oracle $\mathsf{KeyGen}(\mathsf{mpk}_1,\mathsf{msk}_1,\cdot)$, to receive $\mathsf{sk}_{f,1}$ as an answer. Then, $\mathcal{B}_2$ generates $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i,\mathsf{msk}_i,f;r_i)$ for $i \in [3] \setminus \{1\}$ by itself and creates a NIWI proof π_{sk} over the relation R_{set} for the statement-witness pair $y = (\{\mathsf{mpk}_i\}_{i\in[3]},\{\mathsf{sk}_{f,i}\}_{i\in[3]},f)$ and $w = (\{\mathsf{msk}_i\}_{i\in[3]},\{r_i\}_{i\in[3]},\{s_i\}_{i\in[3]})$. Then, $\mathcal{B}_2$ sends $\mathsf{sk}'_f = (\{\mathsf{sk}_{f,i}\}_{i\in[3]},\pi_{\mathsf{sk}})$ as a reply to $\mathcal{A}$.

For the challenge query (x^0,x^1) asked by $\mathcal{A}$, $\mathcal{B}_2$ forwards it to its own encryption oracle and receives $\mathsf{ct}_1 = \mathsf{Enc}(\mathsf{mpk}_1,x^\beta)$ as an answer. It generates $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{mpk}_i,x^1;u_i)$ with $u_i \leftarrow \{0,1\}^\lambda$, for $i \in [3] \setminus \{1\}$, simulates a valid proof π of the relation $R^{\mathsf{CCA}}_{\mathsf{set}}$ using the statement $y = (\mathsf{mpk}_i,\mathsf{ct}_i)_{i\in[3]}$, i.e. $\pi_{\mathsf{ct}} \leftarrow \mathcal{S}_2(\mathsf{CRS},\tau,y)$ and sends $\mathsf{ct}' := (\{\mathsf{ct}_i\}_{i\in[3]},\pi_{\mathsf{ct}})$ to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks a decryption query $(f,\mathsf{ct}' := (\{\mathsf{ct}\}_{i\in[3]},\pi_{\mathsf{ct}}))$, $\mathcal{B}_2$ first verifies the proof π_{ct}, i.e. it executes $\mathsf{NIZK.Verify}(\mathsf{CRS},(\mathsf{mpk}_i,\mathsf{ct}_i)_{i\in[3]},\pi_{\mathsf{ct}})$. If the verification outputs 1, $\mathcal{B}_2$ generates $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i,\mathsf{msk}_i,f)$ for $i \in [3] \setminus \{1\}$, computes $y_{f,i} \leftarrow \mathsf{Dec}(\mathsf{mpk}_i,f,\mathsf{sk}_{f,i},\mathsf{ct}_i)$ for $i \in [3] \setminus \{1\}$ and $y \leftarrow \mathrm{MajVal}(\{y_{f,i}\}_{i\in[3]\setminus\{k\}})$, to $\mathcal{A}$. Since Claim 2 shows that for all ciphertext queries made by $\mathcal{A}$ that have a valid proof (except with negligible probability), it holds that $\mathsf{Dec}(\mathsf{mpk}_1,f,\mathsf{sk}_{f,1},\mathsf{ct}_1) = \mathsf{Dec}(\mathsf{mpk}_2,f,\mathsf{sk}_{f,2},\mathsf{ct}_2) = \mathsf{Dec}(\mathsf{mpk}_3,f,\mathsf{sk}_{f,3},\mathsf{ct}_3)$. Therefore, it is sufficient to generate the decryptions $y_{f,2}$ and $y_{f,3}$ and use them as the decryption output and a reply for $\mathcal{A}$. If the verification outputs 0, $\mathcal{B}_2$ sends $\perp$ to $\mathcal{A}$.

This covers the simulation of the game $\mathsf{G}_{1+\beta}$. Finally $\mathcal{B}_2$ outputs the same bit β' returned by $\mathcal{A}$. Together with the analysis of adversary $\mathcal{B}_1$, this yields the advantage mentioned in the lemma. □

Lemma 11 (Transition from G_2 to G_3)**.** *For any PPT adversary $\mathcal{A}$, there exist PPT adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that*

$$|\mathsf{Win}^{\mathsf{G}_2}_{\mathcal{A}}(1^\lambda) - \mathsf{Win}^{\mathsf{G}_3}_{\mathcal{A}}(1^\lambda)| \leq \mathsf{Adv}^{\text{Sim-Sound}}_{\mathsf{NIZK},\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathrm{WI}}_{\mathsf{NIWI},\mathcal{B}_2}(\lambda) \ .$$

Proof. We build an adversary $\mathcal{B}_2$ that simulates $\mathsf{G}_{2+\beta}$ towards $\mathcal{A}$ when interacting with the underlying $\mathrm{WI}^{\mathsf{NIWI}}_\beta$ experiment.

In the beginning of the reduction, $\mathcal{B}_2$ simulates a CRS, i.e. $(\mathsf{CRS},\tau) \leftarrow \mathcal{S}_1(1^\lambda)$, generates several functional encryption instances $(\mathsf{mpk}_i,\mathsf{msk}_i) \leftarrow \mathsf{Setup}(1^\lambda;s_i)$ with $s_i \in \{0,1\}^\lambda$ for $i \in [3]$, sets $\mathsf{mpk}' := (\mathsf{CRS},\{\mathsf{mpk}_i\}_{i\in[3]})$ and sends mpk' to $\mathcal{A}$.

Whenever $\mathcal{A}$ asks an encryption query (x^0,x^1), $\mathcal{B}_2$ encrypts x^1 using the first public key and x^0 using the second and third public key, i.e $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(\mathsf{mpk}_1,x^1;u_1)$ and $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{mpk}_i,x^0;u_i)$, for $i \in [3] \setminus \{1\}$, where $u_i \leftarrow \{0,1\}^\lambda$. Afterwards, $\mathcal{B}_2$ simulates a valid proof π of the relation $R^{\mathsf{CCA}}_{\mathsf{set}}$ using the statement $y = (\mathsf{mpk}_i,\mathsf{ct}_i)_{i\in[3]}$, i.e. $\pi_{\mathsf{ct}} \leftarrow \mathcal{S}_2(\mathsf{CRS},\tau,y)$ and sends $\mathsf{ct}' := (\{\mathsf{ct}_i\}_{i\in[3]},\pi_{\mathsf{ct}})$ to $\mathcal{A}$.

For a key generation query f, $\mathcal{B}_2$ samples $r_i \leftarrow \{0,1\}^\lambda$ and generates $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i,\mathsf{msk}_i,f;r_i)$ for $i \in [3]$. Afterwards, $\mathcal{B}_2$ submits (z,w_0,w_1) with $z = (\{\mathsf{mpk}_i\}_{i\in[3]},\{\mathsf{sk}_{f,i}\}_{i\in[3},f), w_0 = (\{\mathsf{msk}_i\}_{i\in[3]\setminus\{1\}},\{r_i\}_{i\in[3]\setminus\{1\}},\{s_i\}_{i\in[3]\setminus\{1\}})$ and $w_1 = (\{\mathsf{msk}_i\}_{i\in[3]\setminus\{2\}},\{r_i\}_{i\in[3]\setminus\{2\}},\{s_i\}_{i\in[3]\setminus\{2\}})$ as a challenge query to its challenger and receives π as a reply. Finally, $\mathcal{B}_2$ sends $(\{\mathsf{sk}_{f,i}\}_{i\in[3]},\pi)$ to $\mathcal{A}$ as a reply to the key generation query.

Whenever $\mathcal{A}$ asks a decryption query $(f,\mathsf{ct}' := (\{\mathsf{ct}\}_{i\in[3]},\pi_{\mathsf{ct}}))$, $\mathcal{B}_2$ first verifies the proof π_{ct}, i.e. it executes $\mathsf{NIZK.Verify}(\mathsf{CRS},(\mathsf{mpk}_i,\mathsf{ct}_i)_{i\in[3]},\pi_{\mathsf{ct}})$. If the verification outputs 1, $\mathcal{B}_2$ generates $\mathsf{sk}_{f,i} \leftarrow \mathsf{KeyGen}(\mathsf{mpk}_i,\mathsf{msk}_i,f)$ for $i \in [3]$, computes $y_{f,i} \leftarrow \mathsf{Dec}(\mathsf{mpk}_i,f,\mathsf{sk}_{f,i},\mathsf{ct}_i)$ for $i \in [3]$ and sends $y \leftarrow \mathrm{MajVal}(\{y_{f,i}\}_{i\in[3]\setminus\{k\}})$ to $\mathcal{A}$. If the verification outputs 0, $\mathcal{B}_2$ sends $\perp$ to $\mathcal{A}$.

This covers the simulation of the game $\mathsf{G}_{2+\beta}$. Finally $\mathcal{B}_2$ outputs the same bit β' returned by $\mathcal{A}$. Together with the analysis of adversary $\mathcal{B}_1$, this yields the advantage mentioned in the lemma. □

After proving that the compiler achieves the security lifting from CPA to CCA, we also need to show that the compiler guarantees setup consistency.

Theorem 25. *Let* $\mathsf{FE} = (\mathsf{Setup},\mathsf{KeyGen},\mathsf{Enc},\mathsf{Dec})$ *be a functional encryption scheme,* $\mathsf{NIWI} = (\mathsf{NIWI.Prove},\mathsf{NIWI.Verify})$ *a NIWI proof system for R_{set} (Fig. 9) and* $\mathsf{NIZK} = (\mathsf{NIZK.Setup},\mathsf{NIZK.Prove},\mathsf{NIZK.Verify})$ *a NIZK proof system for $R^{\mathsf{CCA}}_{\mathsf{set}}$ (Fig. 26), then the construction* $\mathsf{FE}' = (\mathsf{Setup}',\mathsf{KeyGen}',\mathsf{Enc}',\mathsf{Dec}')$ *defined in Fig. 25 is setup consistent. Namely, for any PPT adversary $\mathcal{A}$, exists a PPT adversary $\mathcal{B}$ such that*

$$|\Pr[\text{set-CONS}^{\mathsf{FE}'}(1^\lambda,\mathcal{A}) = 1]| \leq \mathsf{Adv}^{\text{Sound}}_{\mathsf{NIWI},\mathcal{B}}(\lambda).$$

Sketch. The proof proceeds following the same reasoning as the proof of Theorem 5. The reason is that the introduction of the additional proof of the ciphertext must always yield 1 (as otherwise, the ciphertext will not be considered by set-CONS) and only those ciphertexts are considered by Dec', as defined in the compiler. Hence, we can invoke the same analysis, based on the invariant that for each ciphertext $\mathsf{ct}'_i = (\mathsf{ct}_i, \pi_i)$ $(i = 1, 2)$, π_i is valid therefore we can perform the identical case distinctions as in the proof of Theorem 5 based on the ciphertexts ct_i. □

Appendix H
Details Of the UC Analysis

A. Assumed Functionalities

We now describe the channels that we assume as setup. Together with FE they realize the ideal repository. The authenticated broadcast channel between sender S and receiver R leaks the message to the adversary. The functionality follows the standard UC corruption model, i.e., in case sender S is corrupted, the adversary can choose the message that is sent but cannot send different messages to different recipients. Since we consider static corruption, our channels are slightly simplified and do not capture the situation where an honest sender is corrupted before one of its messages is delivered (since either the sender is corrupt from the start or remains honest).

Functionality $\mathsf{Func}_{\text{auth}}^{S,R_1,\ldots,R_n}$

The functionality is parameterized by sender (extended) identity S and receiver (extended) identities R_i. The functionality initializes an empty array M.

- On input $(\textsc{Send}, \text{sid}, m)$ from S, store $M \leftarrow M||m$ and output $(\textsc{Sent}, m)$ to every R_i.
- Upon input $(\textsc{GetMsgs}, \text{sid})$ from the adversary (on the backdoor tape) output M to the adversary.

Furthermore, we assume point-to-point secure channels and existence of the real-world repository as defined next.

Functionality $\mathsf{Func}_{\text{sec}}^{S,R}$

The functionality is parameterized by sender S and receiver R. It initializes an empty array M.

- On input $(\textsc{Send}, \text{sid}, m)$ from S, store $M \leftarrow M||\mathsf{length}(\mathsf{m})$ and output $(\textsc{Sent}, m)$ to R.
- Upon input $(\textsc{GetMsgs}, \text{sid})$ from the adversary (on the backdoor tape) output M to the adversary.

Functionality $\mathsf{Func}_{\text{basic-rep},\mathcal{C}}^{\mathtt{A},\mathtt{B},t}$

The functionality is parameterized by a set $\mathcal{C} \subseteq \{0,1\}^*$ and the party identifiers $\mathtt{A}, \mathtt{B}_i$, $i \in [t]$, it interacts with. It manages a lookup table M, which is initially empty.

a) Input.: Upon receiving $(\textsc{write}, \text{sid}, x)$ from a party with party-id $\mathtt{A}$ do: If $c \in \mathcal{C}$, then compute handle $h \leftarrow \mathsf{getHandle}$ and store $M[h] \leftarrow c$. Return h to the calling party.
b) Output.: Upon receiving $(\textsc{read}, \text{sid}, h)$ from a party with party-id $\mathtt{B}_i$, $i \in [t]$, do: If $M[h] = \perp$ then return noData. Otherwise, i.e., if $M[h] \in \mathcal{C}$, return $M[h]$ to the calling party.

B. The FE protocol

The UC protocol $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ (based on a functional encryption scheme FE) to realize $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ from the basic repository an authenticated broadcast channel, and secure channels:

C. Proof of the UC Realization (Theorem 1)

Proof. We start by proving the first part of the theorem for the repository $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ and only then prove the necessary direction. Thereafter, we turn our attention to $\mathsf{Func}_{\text{Rep}^*,(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$.

Consistency implies the UC realization: We first describe the simulator $\mathcal{S}$ for the dummy UC adversary $\mathcal{D}$, which basically means that $\mathcal{S}$ receives the instructions by the environment $\mathcal{Z}$. We are in the static corruption case and thus can structure the proof by a case distinction according to the actual corruption set in the system to obtain the detailed claims of the theorem for each case, which we cast as separate lemmata below.

1) Simulation with only a corrupted input provider: Upon receiving $(\textsc{setup}, \text{sid})$ from $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$, the simulator $\mathcal{S}$ executes $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}()$ to obtain the master public and private key and provides mpk to the environment as the message received by the dishonest input provider and decryptors (or leaked by the authenticated broadcast channel). Upon receiving $(\textsc{assigned}, \text{sid}, f, i)$ from $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ then $\mathcal{S}$ computes $F_0 \leftarrow F_0 \cup \{f\}$ (where F_0 is initially empty) and evaluates $\mathsf{sk}_f \leftarrow \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$ and provides sk_f to the environment when asked $(\textsc{read}, \text{sid})$ (meant for the secure channel between the setup generator and some corrupted decryptor that receives the functional key for f). Finally, update $F \leftarrow F \cup \{(\mathsf{sk}_f, f)\}$ (again F is initially empty).

When given the adversarial input $(\textsc{write}, \text{sid}, \mathsf{ct})$ (an input meant for the real-world repository), the simulator does the following: it sets $x \leftarrow \mathsf{unknown}$ and outputs $(\textsc{write}, \text{sid}, x)$ to $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ in the name of $\mathtt{A}$ and returns the obtained handle to the environment.

Upon receiving $(\textsc{read}, \text{sid}, h, f)$ from $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$, recall the ciphertext ct associated to h (previously input by the corrupted input provider). Then compute $y \leftarrow \mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}_f, \mathsf{ct})$ and provide the input $(\textsc{read}, \text{sid}, h, f, (\mathsf{unknown}, y))$ to $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$.

If any receiver is corrupted, the adversary can instruct them to input $(\textsc{read}, \text{sid}, h)$ to directly obtain the cipher-

Protocol $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$

- Upon each invocation, protocol $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ first verifies that this ITIs party identifier matches pid $\in \{\mathtt{A}, \mathtt{B}_i, \mathtt{C}\}$ ($i \in [t]$) and rejects the message otherwise. This means that the ITI running the protocol must have the extended identity $\text{eid}_{\text{pid}} = (\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}, \text{sid}||\text{pid})$ for some sid and pid $\in \{\mathtt{A}, \mathtt{B}_i, \mathtt{C}\}$ (for some $i \in [t]$).
- Depending on the encoded pid, match the input to the following commands:

  pid = C: The behaviour of the manager is as follows:

  - On input (SETUP, sid) execute $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}()$, store the pair internally and send mpk via $\mathsf{Func}_{\text{auth}}^{\text{eid}_\mathtt{C},\text{eid}_\mathtt{A}}$ to the input provider and via $\mathsf{Func}_{\text{auth}}^{\text{eid}_\mathtt{C},\text{eid}_{\mathtt{B}_1},\ldots,\text{eid}_{\mathtt{B}_t}}$ to the decryptor.
  - On input (ASSIGN, sid, f, i), ensure that $f \in F$ and otherwise ignore the input. Execute $\mathsf{sk}_f = \mathsf{KeyGen}(\mathsf{mpk}, \mathsf{msk}, f)$, send sk_f via $\mathsf{Func}_{\text{sec}}^{\text{eid}_\mathtt{C},\text{eid}_{\mathtt{B}_i}}$ to the decryptor.

  pid = A: The behaviour of the input provider is as follows:

  - On input (WRITE, sid, x), ensure that $x \in \mathcal{X}$ and that an mpk has been received (otherwise ignore the input). Then, execute $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x)$ and if $\mathsf{ct} \neq \mathsf{err}$ output (WRITE, sid, ct) to $\mathsf{Func}_{\text{basic-rep},\mathcal{C}}^{\mathtt{A},\mathtt{B},t}$ and return the obtained handle h from the basic repository back to the caller by returning (WRITTEN, sid, h). (Give up activation if an error occurs).
  - On receiving the master public key mpk from $\mathsf{Func}_{\text{auth}}^{\text{eid}_\mathtt{C},\text{eid}_\mathtt{A}}$ and if this is the first time the key is delivered, store it internally for future reference. Ignore any future message from the channel.

  pid = $\mathtt{B}_i$: The behaviour of the decryptor is as follows:

  - On input (READ, sid, h, f) output (READ, sid, h) to $\mathsf{Func}_{\text{basic-rep},\mathcal{C}}^{\mathtt{A},\mathtt{B},t}$ to obtain a ciphertext ct. If no ciphertext is received or a pair (sk_f, f) is not recorded, then give up activation. Next, execute $y \leftarrow \mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}_f, \mathsf{ct})$ for each candidate pair (sk_f, f) recorded and delete the pair if $y = \diamond$ is obtained. Finally, give up the activation if all values returned $\diamond$. Otherwise, return the first $y \in \mathcal{Y} \cup \{\perp\}$ obtained by decrypting and output (READ, sid, y) to the caller.
  - On receiving the master public key mpk from $\mathsf{Func}_{\text{auth}}^{\text{eid}_\mathtt{C},\text{eid}_{\mathtt{B}_1},\ldots,eid_{\mathtt{B}_t}}$ and if this is the first time the master public key is delivered, store it internally for future reference.
  - On receiving a pair (sk, f) from $\mathsf{Func}_{\text{sec}}^{\text{eid}_\mathtt{C},\text{eid}_{\mathtt{B}_i}}$ and if $f \in F$, then store the pair (sk, f) in the list of received functional keys.
- Ignore the input if no case applies.

text associated to h. The simulated answer simply returns the previously input ciphertext ct for handle h.

Lemma 12. *For any environment $\mathcal{Z}$ (and dummy adversary) with non-negligible advantage in distinguishing the real and ideal worlds (w.r.t. simulator $\mathcal{S}$ above) when only corrupting party $\mathtt{A}$ (and possibly a subset of the receivers), we give a reduction ρ_1 to construct an adversary $\mathcal{A} := \rho_1(\mathcal{Z})$ that violates input-consistency with non-negligible probability.*

Proof. For the reduction to the input-consistency game, consider the following events defined in the real-world execution: Let E_1 denote the event that at any point in the execution of $\mathcal{Z}$ (with the protocol and dummy adversary $\mathcal{D}$), there is a handle h such that the set of associated output values $y_i^{(h)}$ obtained by party $\mathtt{B}$ on queries (READ, h, f_i) are such that $\{x' \in \mathcal{M} \mid \forall i : f_i(x') = y_i^{(h)}\} = \{\}$.

As long as $\neg E_1$ holds in the execution, the simulator $\mathcal{S}$ executes the real-world view perfectly, since the instruction marked with ($\star\star$) must never be executed in the ideal world and any value y returned to an honest decryptor is explainable by an element $x \in \mathcal{X} \cup \{\perp\}$ that fulfills $f(x) = y$ for the queried function $f \in \mathcal{F}$ (and thus also $\diamond$ is never observed).

Hence, let $\mathcal{A} := \rho_1(\mathcal{Z})$ be the consistency adversary that internally runs $\mathcal{Z}$ and emulates the real-world view towards $\mathcal{Z}$, i.e., upon any request by $\mathcal{Z}$, ρ_1 emulates the actions of the protocol when generating its replies to $\mathcal{Z}$. Note that such an emulation is possible with access to the honestly generated master public-key and with access to the key-generation oracle provided by the input-consistency game. Once event E_1 is observed, ρ_1 identifies the handle h that caused the event and outputs the associated ciphertext ct_h stored for handle h. Note that $\mathcal{A}$ is efficiently implementable by the assumption of (the efficiently implementable function) $\mathsf{preMap}()$ which can be used to detect E_1 (and further events in the other cases).[7]

[7] We note that by picking one ciphertext at random, ρ_1 could avoid the dependence on $\mathsf{preMap}()$ at the cost of obtaining a security loss. However, since in order to define the ideal UC functionality (which must be an efficient program) such an efficient map must exist, and since assuming it here yields more straightforward arguments, we rely on it throughout this proof.

The proof for this case is concluded by observing that $\mathcal{A}$ contradicts the theorem assumptions: we have that the evaluation of a set of functions $f_1, \ldots, f_n$ for a handle h returned values $y_1, \ldots, y_n$ that do not have a common explanation (event E_1) which implies that in-CONS$^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ returns 1 with the same probability as the event that $\mathcal{Z}$ provokes event E_1. □

2) Simulation with a corrupted setup generator: In this case we have that all inputs provided to party $\mathtt{A}$ must define a valid base value $x \in \mathcal{X}$ and thus, upon read queries by an honest receiver/decryptor $\mathtt{B}_i$ only 2.(a) in the description of $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ is triggered. The main task of simulator $\mathcal{S}$ in this simulation is to translate dishonest actions of party $\mathtt{C}$ when sending values towards the other two parties and controlling the influence on Alice actions. Thus, upon receiving $(\textsc{Send}, \mathrm{sid}, m)$ for the channel from $\mathtt{C}$ to $\mathtt{A}$, $\mathcal{S}$ just remembers that $\mathsf{mpk}_1 \leftarrow m$ (and similarly for the message sent to the receivers which is defined to be mpk_2). As soon as mpk_1 is defined, $\mathtt{C}$ sends $(\textsc{setup}, \mathrm{sid})$ to the ideal functionality (allowing the sender to input values).

As soon as both values mpk_1 and mpk_2 are defined the simulator starts producing functional secret keys as follows (note that before the master public key is received, no honest decryptor would extend its function set). For messages $(\textsc{Send}, \mathrm{sid}, m)$ for the channel from $\mathtt{C}$ to some honest receiver $\mathtt{B}_i$, the simulator first parses m as (sk, f) and if $f \notin \mathcal{F}$, it gives up activation. Otherwise, it encrypts a fixed message $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}_1, \bar{m})$ and performs a trial decryption $y \leftarrow \mathsf{Dec}(\mathsf{mpk}_2, f, \mathsf{sk}, \mathsf{ct})$. If $y = \diamond$ then give up activation. Otherwise, output $(\textsc{assign}, \mathrm{sid}, f, i)$ to $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ to assign the function f to be available for the decryptor. Note that in case the decryptor is dishonest, simply simulate the receipt of message m.

When activated by $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ with a public delayed output $(\textsc{write}, \mathrm{sid}, x)$ (in response to Alice's input), the simulator performs a trial encryption $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}_1, x)$ and sends ACK for this operation if and only if $\mathsf{ct} \neq \mathsf{err}$. Otherwise, the simulator activates the environment as next entity.[8]

Finally, for corrupted decryptors we have to simulate real-world ciphertexts corresponding. This is simple, as we do not have any ideal privacy guarantees anymore when aside of $\mathtt{C}$ at least one decryptor is dishonest: hence to simulate the ciphertext for handle h the simulator first obtains the message x via the command $(\textsc{reveal}, \mathrm{sid}, h)$ to the ideal-world repository and encrypts x as done in the real world and associates the obtained ciphertext ct with handle h. This concludes the simulation for this case.

Lemma 13. *For any environment $\mathcal{Z}$ (and dummy adversary) with non-negligible advantage in distinguishing the real and ideal worlds (w.r.t. simulator $\mathcal{S}$ above) when only corrupting party $\mathtt{C}$ (possibly alongside a subset of receivers), we give reductions ρ_2 and ρ_3 to construct adversaries $\mathcal{A}_i := \rho_i(\mathcal{Z})$ such that at least one of the $\mathcal{A}_i$ violates setup-consistency with non-negligible probability.*

Proof. For this case, we first make a hybrid argument: consider the protocol π', which is defined as $\pi^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathsf{FE}}$ but where party $\mathtt{A}$ provides its received master-public key mpk_1 to parties $\mathtt{B}_i$ via an additional covert broadcast channel and where parties $\mathtt{B}_i$, already upon receiving a functional key (sk, f), perform a trial decryption and rejects the key if $\mathsf{Dec}(\mathsf{mpk}_2, f, \mathsf{sk}, \mathsf{Enc}(\mathsf{mpk}_1, \bar{m})) = \diamond$, where mpk_2 is the master public key sent from part $\mathtt{C}$ to parties $\mathtt{B}_i$. We observe that protocol π' and $\pi^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathsf{FE}}$ have equivalent behaviors as long as the environment is not able to provide an input $(\textsc{write}, \mathrm{sid}, x)$ to party $\mathtt{A}$ that provokes event E_3 defined by the condition that $\mathsf{Dec}(\mathsf{mpk}_2, f, \mathsf{sk}, \mathsf{Enc}(\mathsf{mpk}_1, \bar{m})) = \diamond$ but $\mathsf{Dec}(\mathsf{mpk}_2, f, \mathsf{sk}, \mathsf{Enc}(\mathsf{mpk}_1, x)) \neq \diamond$ (of course within an honest receiver/decryptor and where f and sk have been received together from party $\mathtt{C}$). In case $\neg E_3$, the function f is never evaluated upon input $(\textsc{read}, h, f)$ (for any h) by protocol π', whereas in $\pi^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathsf{FE}}$ it might. Hence, let $\mathcal{A} := \rho_2(\mathcal{Z})$ be the adversary for the setup consistency game defined as follows: ρ_2 internally runs $\mathcal{Z}$ and emulates the execution of protocol $\pi^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathsf{FE}}$ towards $\mathcal{Z}$ (by monolithically executing all required protocol steps) until event E_3 is observed. In this case, ρ_2 outputs $(\mathsf{mpk}_1, \mathsf{mpk}_2, \mathsf{sk}, x, \bar{m})$, where x and sk are the values fulfilling the condition of event E_3. ρ_2 wins set-CONS$^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ with the same probability as event E_3 in the execution with $\mathcal{Z}$.

For the final argument, we proceed with the same pattern. This time, let E_4 be the event that in an execution with π', $\mathcal{Z}$ provokes for some handle h that a query $(\textsc{read}, \mathrm{sid}, h, f)$ to party $\mathtt{B}$, following a write instruction $(\textsc{write}, \mathrm{sid}, x)$ to party $\mathtt{A}$ that returned this handle h, yields an output value $y \neq f(x)$. Again, the simulation $\mathcal{S}$ interacting with the repository $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ is a perfect simulation of π': in both worlds, functions are assigned that pass the trial-decryption test, and all function evaluations, for some assigned function f, yield $f(x)$ as output as in this case only instruction $(\star\,\star\,\star)$ of the ideal-world repository is executed. Again, we can upper bound the distinguishing advantage of the real and ideal world by the probability that $\mathcal{Z}$ provokes E_4. The corresponding reduction $\mathcal{A} := \rho_3(\mathcal{Z})$ emulates a real-world execution towards $\mathcal{Z}$, where it mimics the protocol actions of the honest parties $\mathtt{A}$ and $\mathtt{B}_i$. This includes the receipt of two message mpk_1 and mpk_2 for parties $\mathtt{A}$ and $\mathtt{B}_i$, respectively. The reduction ρ_3, once it detects event E_4 is provoked, can output $(\mathsf{mpk}_1, \mathsf{mpk}_2, \mathsf{sk}, f, x, x)$, where (sk, f) is defined as the key function pair provoked event E_4. Hence, $\mathcal{A}$ achieves set-CONS$^{\mathsf{FE}}(1^\lambda, \mathcal{A}) = 1$ with at least the probability of $\mathcal{Z}$ provoking E_4. □

3) Simulation with a corrupted input provider and setup generator: The simulator in this case needs to combine parts of the above two simulation strategies for maliciously

[8] Recall that this leakage is implied by the fact that the property whether Alice encrypts successfully could depend on the plaintext she tries to encrypt.

generated setup parameters. That is, it defines mpk as the claimed master public key that party $\mathtt{C}$ sends to an honest party $\mathtt{B}_i$. For the other messages $(\textsc{Send}, \mathsf{sid}, m)$ for the channel from $\mathtt{C}$ to $\mathtt{B}_i$, the simulator again parses it as a key-function pair (sk, f) and does the validity tests as above and in case $f \in \mathcal{F}$ and the trial decryption $\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}, \mathsf{Enc}(\mathsf{mpk}, \bar{m}))$ (with respect to one master public key) does not yield $\diamond$, then output $(\textsc{assign}, \mathsf{sid}, f, i)$ to $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$. Simulating a dishonest receiver is straightforward.

For adversarial inputs by party $\mathtt{A}$, $\mathcal{S}$ again sets $x \leftarrow \mathsf{unknown}$ for the current set of key-function pairs, provides $(\textsc{write}, \mathsf{sid}, x)$ to $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ in the name of $\mathtt{A}$ and returns the obtained handle to the environment.

Finally, upon receiving $(\textsc{read}, \mathsf{sid}, h, f)$ from $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ (upon a reading instruction by honest party $\mathtt{B}_i$), obtain the ciphertext ct associated to h, compute $y \leftarrow \mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}_f, \mathsf{ct})$ and $x \leftarrow \mathsf{unknown}$ and provide the input $(\textsc{read}, \mathsf{sid}, h, f, (x, y))$ to $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$.

Lemma 14. *For any environment $\mathcal{Z}$ (and dummy adversary) with non-negligible advantage in distinguishing the real and ideal worlds (w.r.t. simulator $\mathcal{S}$ above) when only corrupting parties $\mathtt{A}$ and $\mathtt{C}$ (and possibly alongside a subset of receivers), we give reductions ρ_4 and ρ_5 to construct adversaries $\mathcal{A}_i := \rho_i(\mathcal{Z})$ such that at least one of the $\mathcal{A}_i$ violates strong input-consistency with non-negligible probability.*

Proof. We again make a first hybrid step and consider the protocol π'', where each party $\mathtt{B}_i$, upon receiving a functional key sk together with its claimed function f, performs a trial decryption $\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}, \bar{\mathsf{ct}})$, where $\bar{\mathsf{ct}} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \bar{m})$ (i.e., just with respect to the claimed master public key). Analogously to above, let E_5 be the event defined for an execution characterized by the condition that the environment provides a ciphertext ct and a secret key (sk, f) to some honest party $\mathtt{B}_i$ such that $\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}, \bar{\mathsf{ct}}) \neq \diamond$ and $\mathsf{Dec}(\mathsf{mpk}, f, \mathsf{sk}, \mathsf{ct}) = \diamond$. Again, $\pi^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathsf{FE}}$ and π'' have an identical behavior for any honest party $\mathtt{B}_j$ until event E_5 is triggered. As above, this yields a reduction $\mathcal{A} := \rho_4(\mathcal{Z})$, which emulates party $\mathtt{B}$'s actions towards $\mathcal{Z}$ and if $\mathcal{Z}$ provokes E_5, it outputs $(\mathsf{mpk}, \bar{\mathsf{ct}}, \mathsf{ct}, \{\mathsf{sk}\})$, where the triple $\bar{\mathsf{ct}}, \mathsf{ct}$ and sk are the values provided by $\mathcal{Z}$ that trigger event E_5.

The final reduction is obtained by defining, for an execution of $\mathcal{Z}$ with π'' the event E_6 (analogous to E_1 above): Let E_6 denote the event that at any point in the execution of $\mathcal{Z}$ (with the protocol and dummy adversary $\mathcal{D}$), there is a handle h such that the set of associated output values $y_i^{(h)}$ obtained by some honest receiver/decryptor $\mathtt{B}_i$ on queries $(\textsc{read}, h, f_i)$ are such that $\{x' \in \mathcal{M} \mid \forall i : f_i(x') = y_i^{(h)}\} = \{\}$. As long as E_6 does not occur, the outputs generated by any honest party $\mathtt{B}_i$ are the decrypted values computed by the simulator and thus computed as in the real-world execution where $(\star\star)$ is not executed in this case (this includes that $\diamond$ is not computed by the simulator as a return value y in this case). The final adversary $\mathcal{A} := \rho_5(\mathcal{Z})$ for st-in-CONS$^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ is now designed analogous to ρ_1: Here, ρ_5 only emulates the honest receivers' actions towards an environment. When it detects event E_6, it outputs $(\mathsf{mpk}, \mathsf{ct}, \mathsf{ct}, F)$ where ct is the ciphertext that provoked the output and F the key-function pairs provided (and simulated) w.r.t. the union of honest receivers. Again, we see that the adversary $\mathcal{A}$ contradicts the theorem assumptions, since it wins st-in-CONS$^{\mathsf{FE}}(1^\lambda, \mathcal{A})$ (event E_6). □

4) Simulation only with corrupted receivers: This case handles the scenario when the input provider and the setup generator are honest, and we have to argue anything that the union of dishonest receivers/decryptors can do in the real world—where they have access to all ciphertexts and received a set of secret keys—is simulatable in the ideal world, where we by definition only leak the information $f(x)$ if x is an input and f is an assigned function to one of the dishonest receivers in the corruption set.

Since our repository construction is functionally equivalent to the construction presented by Matt and Maurer [52], we inherit the security statement in this case by their statement: in particular, assume the algorithms $\mathcal{S}_1, \mathcal{S}_2$, and $\mathcal{S}_3$ guaranteed to exist by CFE security. The simulator acts as follows (where the union of corrupted receivers is simply seen as one "large corrupted decryptor" and treat it as the one corrupted party): it first simulates the public parameter by executing $(\mathsf{mpk}, s) \leftarrow \mathcal{S}_1()$. Upon receiving $(\textsc{assigned}, \mathsf{sid}, f, j)$ (for some index i) from $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ then $\mathcal{S}$ computes computes $F_0 \leftarrow F_0 \cup \{f\}$ and outputs $(\textsc{Read}, h_i, f)$ for each available label $h_1, \ldots, h_n$ in $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ to obtain the associated value y_i, and execute $\mathsf{sk}_f \leftarrow \mathcal{S}_2(f, y_1, \ldots, y_n)[[s]]$. This key is then output whenever the simulator must simulate the transmission of this key towards a dishonest receiver.

On input $(\textsc{read}, \mathsf{sid}, h)$ from a dishonest decryptor (expecting a real-world ciphertext), do the following: if a handle h has been generated (i.e., assigned to a value by $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$) and a ciphertext ct_h has already been simulated, then return ct_h to the adversary. Otherwise, the ciphertext for this handle is simulated as follows: for all already assigned functions $f_i \in F_0$, ask $(\textsc{Read}, h, f_i)$ to $\mathsf{Func}^{\mathtt{A},\mathtt{B},\mathtt{C},t}_{\mathrm{Rep},(\mathcal{F}^+,f_0)}$ to obtain all function values $y_1, \ldots, y_k$ for this handle (of the underlying input) and also y_0, which is the output of the leakage function f_0. Simulate (and internally store) the ciphertext $\mathsf{ct}_h \leftarrow \mathcal{S}_3(y_0, y_1, \ldots, y_k)[[s]]$ and return c_h as the answer to the adversary.

The reduction to CFE security directly follows from [52, Lemma 4.2].

5) All parties honest: The remaining case is a straightforward simulation of the real-world view: if all parties are honest, then the simulator simply has to generate and output honest public parameters as above.

The necessary direction.: To prove that the consistency requirements described by in-CONS, set-CONS, and st-in-CONS are also necessary, we show that if, for a given scheme FE there are adversaries $\mathcal{A}_1$, $\mathcal{A}_2$, $\mathcal{A}_3$ such that $\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}_1)]$, $\Pr[\text{set-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}_2)]$, or $\Pr[\text{st-in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}_3)]$ is non-negligible, then we can distinguish a real execution of $\pi_{\mathsf{FE}}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ from any ideal-world execution for $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ and *an arbitrary* simulator $\mathcal{S}'$. From the assumed adversaries we construct environments $\mathcal{Z}_1$ and $\mathcal{Z}_2$ that distinguish the real and ideal worlds, the former for the case when only $\mathtt{A}$ is corrupted, and for the latter when only party $\mathtt{C}$ is corrupted. For the sake of the argument, we only have to consider the simple setting with one honest receiver/decryptor $\mathtt{B}_1$ and one corrupted receiver $\mathtt{B}_2$.

Case in-CONS: $\mathcal{Z}_1$ internally runs $\mathcal{A}_1$ and answers its KeyGen-queries for functional keys corresponding to a given function f by providing the input $(\text{ASSIGN}, \text{sid}, f, 2)$ and obtaining the corresponding functional key from obtaining the value of the secure channel from party $\mathtt{C}$ to $\mathtt{B}_2$ since $\mathtt{B}_2$ is corrupted. When $\mathcal{A}_1$ outputs a ciphertext ct, $\mathcal{Z}_1$ does the following: it corrupts party $\mathtt{A}$ and instructs it (all via the dummy adversary) to issue the write instruction $(\text{WRITE}, \text{sid}, \mathsf{ct})$ (destined for the real-world repository) and expect handle h in return. Then issue n read instructions $(\text{READ}, \text{sid}, h, f_i)$ to (honest) receiver $\mathtt{B}_1$ to obtain n values $y_i, 1 \leq i \leq n$. If $\{x' \in \mathcal{M} \mid \forall i : f_i(x') = y_i\} = \{\}$ then $\mathcal{Z}_1$ outputs 1 and otherwise it outputs 0. It is clear that $\mathcal{Z}_1$ never outputs 1 when interacting with any simulator and functionality $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$, since $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ will never output inconsistent values. On the other hand, $\mathcal{Z}_1$ outputs 1 whenever $\mathcal{A}_1$ detects an inconsistency. Hence, the distinguishing advantage equals $\Pr[\text{in-CONS}^{\mathsf{FE}}(1^\lambda, \mathcal{A}_1,)]$.

Case set-CONS: $\mathcal{Z}_2$ internally runs $\mathcal{A}_2$ until it outputs $(\mathsf{mpk}_1, \mathsf{mpk}_2, \mathsf{sk}, f, x_0, x_1)$. $\mathcal{Z}_2$ then interacts with honest party $\mathtt{A}$ and $\mathtt{B}_1$ as follows via the party $\mathtt{C}$ that it corrupts: it instructs the corrupted party to send (via the channels) mpk_1 and mpk_2 to the respective parties $\mathtt{A}$ and $\mathtt{B}_1$ and sk to $\mathtt{B}_1$. Then it chooses a bit j at random and provides the inputs $(\text{WRITE}, \text{sid}, x_j)$ and $(\text{WRITE}, \text{sid}, x_{1-j})$ to party $\mathtt{A}$ to obtain the handles h_1 and h_2. Finally, it provides the input (READ, h_1, f) to party $\mathtt{B}_1$. If it obtains an answer y_1 (and the input is hence not ignored) and $y_1 \neq f(x_j)$, then $\mathcal{Z}_2$ outputs 1 as its decision bit. Otherwise, it provides the input (READ, h_2, f) to party $\mathtt{B}_1$. If it obtains an answer y_2 (and the input is hence not ignored) and $y_2 \neq f(x_{1-j})$, then $\mathcal{Z}_2$ outputs 1 as its decision bit. Finally, if exactly one query returned an answer, then $\mathcal{Z}_1$ outputs 1 as its decision bit. In any other case, $\mathcal{Z}_2$ outputs 0. By assumption, the probability that at least one of the equation $\mathsf{Dec}(\mathsf{mpk}_2, f, \mathsf{sk}, \mathsf{Enc}(\mathsf{mpk}_1, x_j)) \neq \diamond$ holds is at least $\Pr[\text{set-CONS}(1^\lambda, \mathcal{A}_2)]$ and therefore, with probability at least $\frac{\Pr[\text{set-CONS}(1^\lambda, \mathcal{A}_2)]}{2}$ reading h_1 will return a result when interacting with the protocol (as otherwise, the key sk would be ignored). When interacting with the ideal functionality and any simulator $\mathcal{S}'$, then by definition of $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$, either both requests are ignored, or both requests return the expected result $f(x_j)$ and $f(x_{1-j})$ upon the first and second decryption, respectively. Therefore, we conclude that the probability that $\mathcal{Z}_2$ outputs 1 when interacting with the ideal world is zero and we obtain a distinguishing advantage of at least $\frac{\Pr[\text{set-CONS}(1^\lambda, \mathcal{A}_2)]}{2}$ for $\mathcal{Z}_2$.

Case st-in-CONS: $\mathcal{Z}_3$ internally runs $\mathcal{A}_3$ which outputs $(\mathsf{mpk}, \mathsf{ct}_0, \mathsf{ct}_1, \{(\mathsf{sk}_1, f_1), \ldots, (\mathsf{sk}_n, f_n)\})$. $\mathcal{Z}_3$ then corrupts parties $\mathtt{A}$ and $\mathtt{C}$ and instructs party $\mathtt{C}$ to send mpk to (honest) party $\mathtt{B}_1$. $\mathcal{Z}_3$ then instructs party $\mathtt{A}$ to write each ciphertext ct_j to the real-world repository to obtain handle h_j. Then, it does the following for each secret key sk_i, $1 \leq i \leq n$:

1) It instructs party $\mathtt{C}$ to send sk_i to party $\mathtt{B}_1$, followed by a query $(\text{READ}, \text{sid}, h_1, f_i)$.
2) If the above requests got ignored, then it instructs party $\mathtt{C}$ to resend sk_i. In any case, it then issues $(\text{READ}, \text{sid}, h_2, f_i)$.
3) When exactly one of the two queries gets ignored, then $\mathcal{Z}_3$ outputs decision bit 1 and halts. If both returned a value, it records them as $y_i^{\mathsf{ct}_j}$ and proceeds with the next functional key.

If no decision has been reached, then $\mathcal{Z}_3$ defines the sets $S_j := \{x' \in \mathcal{M} \mid \forall i : f_i(x') = y_i^{(ct_j)}\}$ and outputs decision bit 1 if and only if at least one of S_0 or S_1 is equal to the empty set. Otherwise, $\mathcal{Z}_3$ outputs 0. When $\mathcal{Z}_3$ interacts with the protocol, then the values $y_i^{\mathsf{ct}_j}$ are computed exactly as in the game $\text{st-in-CONS}(1^\lambda, \mathcal{A}_3)$ and the decision bit is 1 if and only if the winning condition of the game is met (note that a query (w.r.t. f_i) is ignored if and only if the ciphertext decrypted to $\diamond$ with respect to sk_i in the above execution with $\mathcal{Z}_3$). On the other hand, if $\mathcal{Z}_3$ is interacting with the ideal system, then it would never output 1, as $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ does either answer both queries per evaluation for f_i or none. Furthermore, the above sets S_j are non-empty as ensured by $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ (ensured by instruction $(\star\star)$). On the other hand, it outputs 1 when interacting with the real protocol if and only if the conditions of the game are fulfilled by the output of $\mathcal{A}_3$, yielding a distinguishing advantage of $\Pr[\text{st-in-CONS}(1^\lambda, \mathcal{A}_3)]$.

Universal encryption property.: We finally turn our attention to $\mathsf{Func}_{\text{Rep}^*,(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$. The only difference to $\mathsf{Func}_{\text{Rep},(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ is the case treated in Lemma 13 (dishonest setup generator, honest input provider): $\mathsf{Func}_{\text{Rep}^*,(\mathcal{F}^+,f_0)}^{\mathtt{A},\mathtt{B},\mathtt{C},t}$ does, upon an input by Alice, not provide a public delayed output revealing x (instead, x is kept private); recall that the simulator $\mathcal{S}$ of Lemma 13 crucially needs to perform

a trial encryption of x to see whether to acknowledge Alice's write request or to reject it. However, if we assume the universal encryption property, knowledge of x is no longer needed: instead of doing a test dependent on x, the simulator performs a trial encryption as soon as mpk_1 is defined on a random message x^*. If successful, any future write-request by Alice can be acknowledged irrespective of the content (as otherwise, this contradicts the universal encryption property) and if the trial-encryption yields an error, the simulator will always deny Alice's write attempts irrespective of the content of the message. Hence a private delayed output that does not reveal x is sufficient in this case.

This concludes the proof of the theorem. □