Consistency for Functional Encryption

Christian Badertscher
IOHK
christian.badertscher@iohk.io

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

Aggelos Kiayias
University of Edinburgh and IOHK
aggelos.kiayias@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(·) 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(0) = f_i(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) 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.

1Note 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 wrongful 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 \( \gamma \). Subsequently, Bob, who possesses a key corresponding to an access structure \( \mathcal{A} \) will be able to decrypt the message as long as \( \gamma \in \mathcal{A} \). Consider now also another party, say Bob junior, possessing a key for the access structure \( \mathcal{A}' \subseteq \mathcal{A} \). 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-
corresponding 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 $sk_f$, 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 sufficient.
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].

Notation: The security parameter is denoted by $\lambda \in \mathbb{N}$ and its unary encoding by $1^\lambda$. We call a randomized algorithm $A$ probabilistic polynomial time (PPT), if there exists a polynomial $p()$ such that for every input $x$ the running time of $A(x)$ is bounded by $p(|x|)$. A function $\negl : \mathbb{N} \to \mathbb{R}^+$ is called negligible if for every positive polynomial $p(\lambda)$ a $\lambda_0 \in \mathbb{N}$ exists, such that for all $\lambda > \lambda_0 : \varepsilon(\lambda) < 1/p(\lambda)$. For a function $f$ with domain $\mathcal{X}$ and range $\mathcal{Y}$, we write $f^{-1}(y) := \{ x \in \mathcal{X} | f(x) = y \}$ 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 $A$ and $B$, we write $B^{A()}(x)$ to denote that $A$ gets $x$ as an input and has oracle access to $B$, that is, the response for an oracle query $q$ is $B(q)$. We use $A()[s]$ to denote that $A$ gets an additional input $s$ which it can update. In more detail, let $y \leftarrow A(x)[s]$ corresponds to the algorithm that invokes $(y,s) \leftarrow A(x,s)$ and returns $y$ and updates $s$.

We write $e_i^x$ for the unit vector of length $\ell$ 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 $G_{\text{Gen}}$ be a PPT algorithm that on input $1^\lambda$ returns a description $G = (G, p, g)$ of a cyclic group $G$ of order $p$ for a $\lambda$-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 $g \in G$. This notation is used in the case of matrices.

In more detail, for a matrix $A = (a_{i,j}) \in \mathbb{Z}_p^{n \times m}$, we define $[A]_g$ as the implicit representation of $A$ in $G$:

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

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 $A$ by $\text{Win}^\gamma_G(\lambda) := \Pr[G(\lambda, A) = 1]$, where the probability is taken over the random coins of $G$ and $A$.

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 $f : \mathcal{X} \rightarrow \mathcal{Y}$, 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 $\lambda$ 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 $\bot$ (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 $\bot$ maps to $\bot$ and that symbol $\diamond$ has no preimage:

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

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

and

$$\tilde{f}^{-1}(y) = \begin{cases} f^{-1}(y) & \text{if } y \in \mathcal{Y} \\ \{\bot\} & \text{if } y = \bot \\ 0 & \text{if } y = \diamond \end{cases}$$

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 $f : \mathcal{X} \rightarrow \mathcal{Y}$, 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 $\text{FE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$:

- **Setup($\lambda$):** Takes as input a unary representation of the security parameter $\lambda$ and outputs the master public key $\text{mpk}$ and the master secret key $\text{msk}$.
- **KeyGen(mpk, msk, f):** Takes as input the master public key $\text{mpk}$, the master secret key $\text{msk}$ and a function $f \in \mathcal{F}_\lambda$, and outputs a functional key $\text{sk}_f$. The key for the leakage function $f_0$ is the empty string denoted by $\varepsilon$.
- **Enc(mpk, x):** Takes as input the master public key $\text{mpk}$ and a string $x \in \mathcal{X}_\lambda$, and outputs a ciphertext $\text{ct}$ or err (to denote an encryption error).
- **Dec(mpk, f, sk_f, ct):** Takes as input a functional key $\text{sk}_f$ and a ciphertext $\text{ct}$ and outputs a function value $y \in \mathcal{Y}_\lambda$ or one of the special symbols of the function extension: $\bot$ indicates an invalid ciphertext and $\diamond$ invalid keys.

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

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

For notational simplicity, we omit certain input values when they are not required by a concrete scheme (such as the additional $\text{mpk}$ 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 encryptions) 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 $\mathcal{D}_k := \mathcal{D}_{k+1, k}$.

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

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 $G_{\text{Gen}}$ if for all PPT adversaries $A$, $\text{Adv}_{G_{\text{Gen}}, A}^{\mathcal{D}_k\text{-MDDH}}(\lambda) := |\Pr[A(G, [A], [Aw]) = 1] - \Pr[A(G, [A], [u]) = 1]| \leq \text{negl}(\lambda)$, where the probability is taken over $G = (G, p, g), A \leftarrow \mathcal{D}_k, w \leftarrow \mathbb{Z}_p^k, u \leftarrow \mathbb{Z}_p^{k+1}$ and the coin tosses of adversary $A$.

Finally, we recall non-interactive witness indistinguishable (NIWI) proofs [13], [20], [46].
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 $\NIWI = (\NIWI.\text{Prove}, \NIWI.\text{Verify})$:

\begin{align*}
\NIWI.\text{Prove}(1^\lambda, x, w): & \text{ Takes as input the unary representation of the security parameter } \lambda, \text{ a statement } x \text{ and a witness } w, \text{ and outputs a proof } \pi. \\
\NIWI.\text{Verify}(1^\lambda, x, \pi): & \text{ Takes as input the unary representation of the security parameter } \lambda, \text{ a statement } x \text{ and a proof } \pi, \text{ and outputs 0 or 1.}
\end{align*}

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

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

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

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

$$\Adv_{\NIWI,A}^{\text{Sound}}(\lambda) := \Pr[(x, \pi) \leftarrow A(1^\lambda) : \NIWI.\text{Verify}(1^\lambda, x, \pi, 1) = 1 \land x \notin L].$$

A NIWI proof system NIWI is called perfectly sound if $\Adv_{\NIWI,A}^{\text{Sound}}(\lambda) = 0$ for all algorithms $A$, and computationally sound, if $\Adv_{\NIWI,A}^{\text{Sound}}(\lambda) \leq \text{negl}(\lambda)$ for all PPT algorithms $A$.

Definition 8 (Witness-Indistinguishability). Let $\NIWI = (\NIWI.\text{Prove}, \NIWI.\text{Verify})$ be a NIWI proof system for a relation $R$ and the corresponding language $L$. For $\beta \in \{0, 1\}$, we define the experiment $\NIWI^{\beta}(1^\lambda, A)$ in Fig. 1. The associated advantage of an adversary $A = (A_1, A_2)$ is defined as

$$\Adv_{\NIWI,A,S}^{\beta}(\lambda) := |\Pr[\NIWI^{\beta}(1^\lambda, A) = 1] - \Pr[\NIWI^{\beta}(1^\lambda, A) = 1]|.$$ 

A NIWI proof system is called witness-indistinguishable, if $\Adv_{\NIWI,A}^{\text{WNIWI}}(\lambda) = 0$ for all algorithms $A = (A_1, A_2)$, and computationally witness-indistinguishable, if $\Adv_{\NIWI,A}^{\text{WNIWI}}(\lambda) \leq \text{negl}(\lambda)$ for all PPT algorithms $A = (A_1, A_2)$.

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.

<table>
<thead>
<tr>
<th>Notions</th>
<th>Input Provider</th>
<th>Setup &amp; Key Generator</th>
<th>Decryptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
<td>Honest</td>
<td>Honest</td>
<td>Honest</td>
</tr>
<tr>
<td>in-CONS</td>
<td>Corrupted</td>
<td>Honest</td>
<td>Honest</td>
</tr>
<tr>
<td>set-CONS</td>
<td>Honest</td>
<td>Corrupted</td>
<td>Honest</td>
</tr>
<tr>
<td>st-in-CONS</td>
<td>Corrupted</td>
<td>Honest</td>
<td>Honest</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Honest</td>
<td>Honest</td>
<td>Corrupted</td>
</tr>
</tbody>
</table>

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 $F$, 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 $sk_{f_1}, \ldots, sk_{f_s}$ 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 $sk_{f_1}, \ldots, sk_{f_n}$, and inconsistent otherwise.

We formalize input-consistency as an experiment.

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

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

a) **Discussion:** The game reflects (in-)consistency: After the adversary $A$ asked the key generation oracle $\text{KeyGen}(\text{msk}, \cdot)$ for functional keys $sk_{f_1}, \ldots, sk_{f_n}$ 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 $\bigcap_{i \in [n]} f_i^{-1}(\text{Dec}(\text{mpk}, f_i, sk_{f_i}, ct))$. 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 $x \in f_i^{-1}(y)$ to be computable (where $X_i$, $Y_i$, and $f_i^{-1}(\cdot)$ are by definition finite sets in an experiment parameterized by $\lambda$). 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., $f_i^{-1}(\diamond) = \emptyset$ (see Definition 2), which results in an empty intersection; in particular there exists no $x$ in the message space $x \in X \cup \{\perp\}$ 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 $\text{mpk}$, two ciphertexts $ct_1$, $ct_2$ and a set of functional keys $\{sk_{f_i}\}_{i \in [n]}$ that decrypt the ciphertexts $ct_1$, $ct_2$ 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.
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 \( \text{ct} \) under the different functional keys \( \text{sk}_{f_1}, \ldots, \text{sk}_{f_n} \). Formally:

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

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

a) **Discussion**: The experiment above strengthens the attack capabilities of the input consistency experiment. Here, the adversary can output a master public key \( \text{mpk} \), ciphertexts \( \{\text{ct}_i\}_{i \in [2]} \) and a set of functional keys \( \{\text{sk}_{f_j}\}_{j \in [n]} \) (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 \( (\text{sk}_{f_j}, f_j) \) that yield a decryption \( y_{i,j} \neq \dagger \), for at least one ciphertext \( \text{ct}_i \). Only keys in \( F \) can provoke a consistency breach. As we detail below, this assigns the correct meaning of key validity to \( \dagger \). The challenger checks for a common explanation, i.e., whether there exists a message in the intersection of the preimages under the different functions \( \{f_j\}_{j \in [n]\setminus\{j\}} \). If the intersection is empty, the adversary has generated a ciphertext \( \text{ct} \) with a decryption behavior that cannot be explained. Again, the symbols \( \dagger \) and \( \bot \) of Definition 2 deserve a special observation. A key’s invalidity only provokes an inconsistency if not all decryptions w.r.t. this key yield \( \dagger \). If a key yields consistently the “decryption” \( \dagger \), this key is detected as invalid; otherwise, if for some ciphertext \( \text{ct} \), we have that exactly one decryption \( y_{i,j} \neq \dagger \), 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., \( y_{i,j} = \bot \) for some key \( \text{sk}_{f_j} \), then all keys in \( F \) must consistently declare this ciphertext invalid and agree on the single possible preimage \( \{\bot\} \) (since special symbol \( \bot \) maps only to \( \{\bot\} \)). 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 \( \text{mpk} \) and a functional key \( \text{sk} \) 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 \( \text{sk} \) 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 \( \text{sk} \) reveals a mismatch with respect to the input values and the declared function \( f \) (unless \( \text{sk} \) is identified as invalid). As for strong input consistency, consistency of key validity as modeled by \( \dagger \) 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 \( \text{FE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}) \) satisfies setup consistency (or set-CONS for short), if for any polynomial-time adversary \( A \) interacting with experiment \( \text{set-CONS}^{\text{FE}} \) in Fig. 4, there exists a negligible function \( \text{negl} \) such that:

\[
\text{Pr}[\text{set-CONS}^{\text{FE}}(1^\lambda, A)] \leq \text{negl}(\lambda).
\]
In addition, we say that $\text{FE}$ 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 $A$ outputs two master public keys $\text{mpk}_1$ and $\text{mpk}_2$, a functional key $\text{sk}$, a function $f$ and two chosen messages $x_1$ and $x_2$, the challenger encrypts the messages under $\text{mpk}_1$ to generate $\text{ct}_i = \text{Enc}(\text{mpk}_1, x_i)$. 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 $\text{mpk}_1$). Let us, for concreteness, discuss the case where both encryptions are valid: if both decryption invocations under $\text{sk}$ 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 $y_i \neq 1$ 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 $x_i \neq 1$ maps to $1$.

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 $\text{mpk}_1$ 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 $id$. The encryption algorithm of the modified scheme encrypts using $\text{mpk}$ exactly as the base scheme does whenever the given master public key has the form $id||\text{mpk}$. However, if given a public key with a different identifier $id||\text{mpk}$, 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 $\text{Setup}$ 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 noton 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

\[
\text{set-CONS}^{\text{FE}}(1^\lambda, A)\\
(\text{mpk}_1, \text{mpk}_2, \text{sk}, f, x_1, x_2) \leftarrow A(1^\lambda)\\
(\text{Assume } x_1 \in X' \text{ and } \text{sk} \neq x')\\
\text{ct}_i \leftarrow \text{Enc}(\text{mpk}_i, x_i), \text{for all } i \in \{1, 2\}\\
I := \{i \mid i \in \{1, 2\} \land \text{ct}_i \neq \text{err}\}\\
(|I| \neq 1 \text{ for universal encryption property})\\
y_i := \text{Dec}(\text{mpk}_2, f, \text{sk}, \text{ct}_i) \text{ for all } i \in I\\
\text{If } y_i \neq \diamond \text{ for some } i \in I\\
\text{If } y_i \neq f(x_i) \text{ for some } i \in I\\
\text{Output 1}\\
\text{Output 0}
\]
The goal of plain functional encryption is to realize this functionality 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 (cti ≠ 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 protocol (Func\textsubscript{\textit{Rep.}}\textsubscript{\textit{f} \rightarrow \textit{fo}} generates a public-delayed output to \( \mathcal{C} \)). 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 Func\textsubscript{\textit{Rep.}}\textsubscript{\textit{f} \rightarrow \textit{fo}} below, where the public-delayed output to \( \mathcal{C} \) 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 \( \pi_{\text{FE}}^{A,B,C,t} \) for parties \( A, B, \) and \( C \), where party \( A \) acts as an input provider, parties \( B_i \) act as the receivers, and party \( C \) 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 \( \mathcal{C} \) and \( A \) and between \( \mathcal{C} \) and the receivers \( \{B_i\}_{i \in [t]} \). Note that we do not require broadcast from \( \mathcal{C} \) 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 \textit{w.r.t.} to the other public key invalid under its own public key.
The functionality is parameterized by function class $F$, the number $t$ of decryptors/receivers, and by three distinct party identities $P := \{A, B, C\}$. These dummy parties interact with the functionality and identify particular roles.

**Setup.** Upon receiving input $(\text{setup}, \text{sid})$ via dummy party $C$ (or from the adversary on behalf of corrupted $C$), set $\text{setup} \leftarrow \text{true}$, $R_i \leftarrow \emptyset$, for each $i \in [t]$. Ignore the request if the party-id does not correspond to $C$. 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 $A$ (or from the adversary on behalf of corrupted $A$), and if $\text{setup} = \text{true}$, do the following:

- If pid $A$ is honest then verify that $x \in X$ (ignore request otherwise). If party $C$ is honest, then compute handle $h \leftarrow \text{getHandle}$ and store $M[h] \leftarrow (x, \{x\})$ and return $(\text{written}, \text{sid}, h)$ to the calling party.
- If party $C$ is corrupted, then do one of the following depending on the version of the repository:
  - $\text{Rep}$: Provide public delayed-output to the adversary and do the previous actions only upon receiving ACK for this operation.
  - $\text{Rep}^*$: Provide private delayed-output to the adversary and do the previous actions only upon receiving ACK for this operation.

- If pid $A$ is marked as corrupted, verify that $x \in X \cup \{\bot, \text{unknown}\}$ (and ignore the request otherwise). Then choose $h \leftarrow \text{getHandle}$ 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 $C$ (or from the adversary on behalf of $C$), do the following: if $f \in F^+$, 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 $B_i$ (or from the adversary on behalf of corrupted $B$), first parse $M[h]$ as $(x, \mathcal{M})$. In case $M[h] = \bot$, return noData.

- If $B_i$ is honest do the:
  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 $x = \bot$ then return $(\text{Read}, \text{sid}, \bot)$ to the calling party. (***)
  2) Otherwise, if $x = \text{unknown}$, 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 $\mathcal{M}_{\text{new}} \leftarrow \text{preMap}(\mathcal{M}, f, y)$.
      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. (**)
      ii) Otherwise, update the entry either by $M[h] \leftarrow (\text{unknown}, \mathcal{M}_{\text{new}})$ 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 $B_i$ is marked as corrupted but none of $A$ or $C$, 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 $B_i$ is corrupted alongside $A$ or $C$, 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 $B_i$ and at least one more party among $\{A, C\}$ 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 \( C \) and each of the receivers \( B_i \), 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 \( B_i \) 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, \( \sigma^{A,B,C,t}_{\text{FE}} \) 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 \( C \) generates the public keys (and sends them to the other parties) and assigns functions to parties \( B_i \) by sending the functional keys. Party \( A \) 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 \( B_i \) and decrypted using a (valid) \( \text{sk} \) 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 \( \text{FE} = \{\text{Setup, KeyGen, Enc, Dec}\} \) be a functional encryption scheme for functionality class \( \mathcal{F} \), and let \( A, B, C \) be three identifiers. Protocol \( \sigma^{A,B,C,t}_{\text{FE}} \) realizes \( \text{Func}^{A,B,C,t}_{\text{Rep}}(\mathcal{F}^+, \mathcal{F}_0) \) (under static corruption) under the following conditions:

- If party \( A \) is corrupted, and \( C \) is honest (and potentially a subset of receivers is corrupted), then in-CONS is a sufficient requirement on \( \text{FE} \) such that \( \sigma^{A,B,C,t}_{\text{FE}} \) realizes \( \text{Func}^{A,B,C,t}_{\text{Rep}}(\mathcal{F}^+, \mathcal{F}_0) \).
- If party \( C \) is corrupted and party \( A \) is honest (and possibly a subset of receivers is corrupted), then set-CONS is a sufficient requirement on \( \text{FE} \) such that \( \sigma^{A,B,C,t}_{\text{FE}} \) realizes \( \text{Func}^{A,B,C,t}_{\text{Rep}}(\mathcal{F}^+, \mathcal{F}_0) \).
- If parties \( A \) and \( C \) are corrupted (and possibly a subset of receivers is corrupted), then \( \text{FE} \) is a sufficient requirement on \( \text{FE} \) such that \( \sigma^{A,B,C,t}_{\text{FE}} \) realizes \( \text{Func}^{A,B,C,t}_{\text{Rep}}(\mathcal{F}^+, \mathcal{F}_0) \).
- If both \( A \) and \( C \) are honest, and only a subset of receivers is corrupted, then CFE security is a sufficient requirement on \( \text{FE} \) such that \( \sigma^{A,B,C,t}_{\text{FE}} \) realizes \( \text{Func}^{A,B,C,t}_{\text{Rep}}(\mathcal{F}^+, \mathcal{F}_0) \).

The above statements hold for the repository \( \text{Func}^{A,B,C,t}_{\text{Rep}}(\mathcal{F}^+, \mathcal{F}_0) \) if \( \text{FE} \) 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 \( \text{FE} \) in order for \( \sigma^{A,B,C,t}_{\text{FE}} \) to realize the specified security guarantees by \( \text{Func}^{A,B,C,t}_{\text{Rep}}(\mathcal{F}^+, \mathcal{F}_0) \) 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 \( \mathcal{F}_m^\mathcal{X,Y} \) and the functionality class \( \mathcal{F}_m^\mathcal{P} \) over \( \mathbb{Z}_p^m \) which we define here for completeness:

**Inner Product (IP) over \( \mathbb{Z}_p \).** Let \( \mathcal{F} = \{ \mathcal{F}_m^\mathcal{X,Y} \}_{\lambda \in \mathbb{N}} \) be a family (indexed by \( \lambda \)) of sets \( \mathcal{F}_m^\mathcal{X,Y} \), where \( \lambda \) is a modulus of length \( \lambda \). Omitting the index \( \lambda \), the set \( \mathcal{F}_m^\mathcal{X,Y} = \{ f_y : \mathbb{Z}_p^m \rightarrow \mathbb{Z}_p, \text{ for } y \in \mathbb{Z}_p^m \} \) where

\[
\begin{align*}
  f_y(x) &= \langle x, y \rangle \mod P \\
\end{align*}
\]

defines the inner-product operation over \( \mathbb{Z}_p \).

**Bounded-Norm IP over \( \mathbb{Z} \).** Let \( \mathcal{F} = \{ \mathcal{F}_m^\mathcal{X,Y} \}_{\lambda \in \mathbb{N}} \) be a family (indexed by \( \lambda \)) of sets \( \mathcal{F}_m^\mathcal{X,Y} \). Omitting the index \( \lambda \), the set \( \mathcal{F}_m^\mathcal{X,Y} = \{ f_y : \mathbb{Z}_m \rightarrow \mathbb{Z}, \text{ with } y \in \mathbb{Z}_m \} \), where \( \mathbb{Z}_m := \{ x : x \in \mathbb{Z}_m, \text{ with } ||x||_\infty < X \}, \mathbb{Z}_p^m := \{ y \in \mathbb{Z}_m, \text{ with } ||y||_\infty < Y \} \) and where \( f_y(x) = \langle x, y \rangle \), 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.

Besides this, we introduce a natural modification of the above functionality class and denote it by \( \mathcal{F}_m^\mathcal{P} \) 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, \( \mathcal{F}_m^\mathcal{P} \) introduced below in Section V-D which is input consistent.
Theorem 2. The functional encryption scheme $\text{FE}$ described in Fig. 7 for the functionality class $\bigl\{F_{m,X,Y}^\lambda\bigr\}$ and $\bigl\{F_{m,X,Y}^\lambda\bigr\}$, with $p$ prime, is not input consistent. Namely, there exists a PPT adversary $A$ such that

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

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 $\text{oob}$ in the case that it is not able to do the discrete logarithm computation in the last step. The preimage of the $\text{oob}$ symbol is then defined as all the $x$ such that $\langle x, y \rangle$ 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 $\mathcal{F} = \{F_{m,L}^\lambda\}_{\lambda \in \mathbb{N}}$ be a family (indexed by $\lambda$) of sets $F_{m,L}^\lambda$, where $P_L$ is a modulus of length $\lambda$ and $L$. Omitting the index $\lambda$, the set $F_{m,L} = \{f_y : \mathbb{Z}_p^m \to \mathbb{Z}_p, \text{ for } y \in \mathbb{Z}_p^m\}$ where

$$f_y(x) = \begin{cases} \langle x, y \rangle \mod p & \text{if } \langle x, y \rangle \in \{0, \ldots, L\} \\ \text{oob} & \text{if } \langle x, y \rangle > L \end{cases}$$

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

The out-of-bound symbol $\text{oob}$ 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 $f_y^{-1}(\text{oob}) = \{x \in \mathbb{Z}_p^m : \langle x, y \rangle > L\}$. The preimage for all other outcomes is $f_y^{-1}(z) = \{x \in \mathbb{Z}_p^m : \langle x, y \rangle = z\}$. When we consider the functional encryption scheme for the modified inner-product functionality $F_{m,L}^\lambda$, it achieves input consistency.

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

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

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 achieve this by omitting the discrete logarithm computation in the end of the decryption procedure and just output the value $g(x,y)$. More formally:

**Exponential IP over $\mathbb{Z}_p$.** Let $\mathcal{P} = \{P_{m,L}^\lambda\}_{\lambda \in \mathbb{N}}$ be a family of sets $P_{m,L}^\lambda$, where $p$ is a prime of length $\lambda$ and $G$ a group
of size $p$ and generator $g$. The sets are defined by $P^m_\mathcal{F} = \{f_{g,y} : (\mathbb{Z}^m_p) \rightarrow \mathbb{G}, \text{ with } y \in \mathbb{Z}^m_p\}$ where

$$f_{g,y}(x) = g(x,y).$$

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 $P^m_\mathcal{F}$, with $p$ prime, described in Fig. 7 is input consistent. Namely, for any PPT adversary $\mathcal{A}$ it holds that $\Pr[\text{in-CONS}_\mathcal{FE}(1^\lambda, \mathcal{A}) = 1] = 0$.

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.

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 algorithms for ciphertext and functional key verification, we can derive a (standard) FE scheme that, as part of the decryption procedure, verifies ciphertexts (and returns $\bot$ 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 $R_{\text{set}}$ 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

As mentioned in Section II, we omit $\lambda$ for simplicity.
Let \( \mathcal{A} \) be a functional encryption scheme and \( \text{NIWI} = (\text{NIWI.Prove}, \text{NIWI.Verify}) \) a NIWI proof system for \( R_{\text{set}} \) (Fig. 9), then the construction \( \mathcal{F} = (\text{Setup}', \text{KeyGen}', \text{Enc}', \text{Dec}') \) defined in Figure 8 is setup consistent. Namely, for any PPT adversary \( \mathcal{A} \), there exists a PPT adversary \( \mathcal{B} \) such that:
\[
|\Pr[\text{set-CONS}^\mathcal{F}(1^\lambda, \mathcal{A}) = 1]| \leq \text{Adv}_{\text{NIWI}, \mathcal{B}}^{\text{Sound}}(\lambda).
\]

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


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

A. Security Definitions

**Definition 12** (CPA & CCA Security of FE). Let \( \text{FE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}) \) 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 IND-CPA\(_\beta\)\(^{\text{FE}}(1^\lambda, \mathcal{A})\) and IND-CCA\(_\beta\)\(^{\text{FE}}(1^\lambda, \mathcal{A})\) in Fig. 10. The associated advantage of an adversary \( \mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \) for XX \( \in \{\text{CPA}, \text{CCA}\} \) is defined by

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

An adversary \( \mathcal{A} \) is valid if for the two submitted challenges \( x^0 \) and \( x^1 \) and all keys \( \text{sk}_f \) the attacker obtained for \( f \) via calls to \( \text{KeyGen} \) (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 \( \text{QDec}(f, ct) \) on the challenge ciphertext \( ct = \text{Enc}(\text{mpk}, x^2) \).

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

---

**Definition 13** (Composable Functional Encryption Security). Let \( \text{FE} \) be a functional encryption scheme, \( \mathcal{F} = \{ \mathcal{F}_\lambda \}_{\lambda \in \mathbb{N}} \) a function family, define the experiments Real\(_{\text{FE}}\)\(^{\lambda, \mathcal{A}}(\mathcal{S})\) and Ideal\(_{\text{FE}}\)\(^{\lambda, \mathcal{A}, \mathcal{S}}\) 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:

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

where \( \mathcal{D} \) is a PPT distinguisher.

A functional encryption scheme \( \text{FE} \) is CFE secure, if there exists a PPT simulator \( \mathcal{S} \), such that for any PPT distinguisher \( \mathcal{D} \) it holds that \( \text{Adv}_{\text{FE,}A,S}^{\mathcal{D},\text{CFE}}(\lambda) \leq \text{negl}(\lambda) \) 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 \(L = \{x \mid \exists w \text{ such that } (x, w) \in R\} \) (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 \(\text{NIZK} = (\text{NIZK.Setup}, \text{NIZK.Prove}, \text{NIZK.Verify})\):

- **NIZK.Setup**\((1^\lambda)\): Takes as input a security parameter \(\lambda\) and outputs the common reference string \(\text{CRS}\).
- **NIZK.Prove**\((\text{CRS}, x, w)\): Takes as input the common reference string \(\text{CRS}\), a statement \(x\) and a witness \(w\), and outputs a proof \(\pi\).
- **NIZK.Verify**\((\text{CRS}, x, \pi)\): Takes as input the common reference string \(\text{CRS}\), a statement \(x\) and a proof \(\pi\), and outputs 0 or 1.

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

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

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 \(\text{NIZK} = (\text{NIZK.Setup}, \text{NIZK.Prove}, \text{NIZK.Verify})\) for a relation \(R\) and the corresponding language \(L\), we define the soundness advantage of an adversary \(\mathcal{A}\) as the probability:

\[
\text{Adv}_{\text{NIZK}, \mathcal{A}}^{\text{Sound}}(\lambda) := \text{Pr}[\text{CRS} \leftarrow \text{NIZK.Setup}(1^\lambda); (x, \pi) \leftarrow \mathcal{A}(\text{CRS}) : \text{NIZK.Verify}(\text{CRS}, x, \pi) = 1 \land x \notin L].
\]

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

**Definition 16** (Zero-Knowledge). Let \(\text{NIZK} = (\text{NIZK.Setup}, \text{NIZK.Prove}, \text{NIZK.Verify})\) be a NIZK proof system for a relation \(R\) and the corresponding language \(L\), \(\mathcal{S} = (S_1, S_2)\) a pair of algorithms (the simulator), with \(S_1^\prime(\text{CRS}, \tau, x, w) = S_2(\text{CRS}, \tau, x)\) for \((x, w) \in R\), and \(S_1^\prime(\text{CRS}, \tau, x, w) = \text{failure}\) for \((x, w) \notin R\). For \(\beta \in \{0, 1\}\), we define the experiment \(\text{ZK}_{\beta}^{\text{NIZK}}(1^\lambda, \mathcal{A})\) in Fig. 12. The associated advantage of an adversary \(\mathcal{A}\) is defined as

\[
\text{Adv}_{\text{NIZK}, \mathcal{A}, S}^{\text{ZK}}(\lambda) := \text{Pr}[(\text{ZK}_{0}^{\text{NIZK}}(1^\lambda, \mathcal{A}, S) = 1) \land \text{NIZK.Verify}(\text{CRS}, x, \pi, w) = 1] - \text{Pr}[(\text{ZK}_{1}^{\text{NIZK}}(1^\lambda, \mathcal{A}, S) = 1) \land \text{NIZK.Verify}(\text{CRS}, x, \pi, w) = 1].
\]

A NIZK proof system \(\text{NIZK}\) is called perfect zero-knowledge, with respect to a simulator \(\mathcal{S} = (S_1, S_2)\), if \(\text{Adv}_{\text{NIZK}, \mathcal{A}, S}^{\text{ZK}}(\lambda) = 0\) for all algorithms \(\mathcal{A}\), and computationally zero-knowledge, if \(\text{Adv}_{\text{NIZK}, \mathcal{A}, S}^{\text{ZK}}(\lambda) \leq \text{negl}(\lambda)\) 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 \(\text{NIZK} = (\text{NIZK.Setup}, \text{NIZK.Prove}, \text{NIZK.Verify})\) for an NP relation \(R\) with corresponding language \(L\) and a simulator \(\mathcal{S} = (S_1, S_2)\), we define the simulation-soundness advantage of an algorithm \(\mathcal{A}\) by

\[
\text{Adv}_{\text{NIZK}, \mathcal{A}, S}^{\text{Sim-Sound}}(\lambda) := \text{Pr}[(\text{NIZK.Prove})(\text{CRS}, x, \mathcal{A}(\text{CRS})) \land (x, \pi) \leftarrow \mathcal{A}(\text{CRS}) : (x, \pi) \notin Q \land x \notin L \land \text{NIZK.Verify}(\text{CRS}, x, \pi) = 1],
\]

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

A NIZK proof system is called one-time simulation sound with respect to the simulator \(\mathcal{S} = (S_1, S_2)\), if \(\text{Adv}_{\text{NIZK}, \mathcal{A}, S}^{\text{Sim-Sound}}(\lambda) \leq \text{negl}(\lambda)\) for all PPT algorithms \(\mathcal{A}\) that make at most one query to oracle \(S_2\).

### 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 \(\text{VFE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{VerifyCT}, \text{VerifySK})\) extends a functional encryption scheme \(\text{FE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})\) by two algorithms \(\text{VerifyCT}\) and \(\text{VerifySK}\) which have the following behavior:

- **VerifyCT**\((\text{mpk}, \text{ct})\): Takes as input the master public key \(\text{mpk}\) and a ciphertext \(\text{ct}\) and outputs 1 if the ciphertext \(\text{ct}\) was correctly generated using the master public key \(\text{mpk}\) for some message \(x\).
- **VerifySK**\((\text{mpk}, f, \text{sk})\): Takes as input the master public key \(\text{mpk}\), a function \(f\) and a functional key \(\text{sk}\) 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 \( \mathcal{FE} \) for \( \mathcal{F} \) is verifiable if, for all \( \text{mpk} \in \{0,1\}^* \), for all \( \text{ct} \in \{0,1\}^* \), there exists \( x \in \mathcal{X} \) such that for all \( f \in \mathcal{F} \) and \( \text{sk} \in \{0,1\}^* \), the following implication holds:

\[
\text{If } \text{VerifyCT}(\text{mpk}, \text{ct}) = 1 \text{ and } \text{VerifySK}(\text{mpk}, f, \text{sk}) = 1 \text{ then } \Pr[\text{Dec}(\text{mpk}, f, \text{sk}, \text{ct}) = f(x)] = 1
\]

### 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 \( \pi \) 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 \( \text{id} = (M, \text{id}) \), where \( M \) is the description of an ITM and \( \text{id} = (\text{sid}, \text{pid}) \) is a string consisting of a session identifier \( \text{sid} \) and a party identifier \( \text{pid} \in \mathcal{P} \). An instance, also called a session, of a protocol \( \pi \) (represented as an ITM \( M_\pi \)) with respect to a session number \( \text{sid} \) is defined as a set of ITIs \( \{\langle M_\pi, \text{id}_{\text{pid}} \rangle \}_{\text{pid} \in \mathcal{P}} \) where \( \text{id}_{\text{pid}} = (\text{sid}, \text{pid}) \).

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

b) Ideal-world process: The ideal process is formulated with respect to an ITM \( \text{Func} \) which is called an ideal functionality. In the ideal process, the environment \( \mathcal{Z} \) interacts with \( \text{Func} \), 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 \( \text{EXEC}_{\text{Func},A,Z}(k,z,r) \) where the inputs are as in the real-world process. Let \( \text{EXEC}_{\text{Func},S,Z}(k,z) \) denote the random variable obtained by choosing the randomness \( r \) uniformly at random and evaluating \( \text{EXEC}_{\text{Func},S,Z}(k,z,r) \). Let \( \text{EXEC}_{\text{Func},S,Z}(k,z) \) denote the ensemble \( \{\text{EXEC}_{\text{Func},S,Z}(k,z)\}_{k \in \mathbb{N}, z \in \{0,1\}^*} \).

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 \( \text{Func} \) if the real-world process (where the protocol is executed) is indistinguishable from the ideal-world process (relative to \( \text{Func} \)).

**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 \( z \in \bigcup_{c \leq k} \{0,1\}^c \). We use the shorthand notation \( \mathcal{X} \approx \mathcal{Y} \) to denote two indistinguishable ensembles.

**Definition 21.** Let \( \text{Func} \) be an ideal functionality and let \( \pi \)
be a protocol. We say that \( \pi \) securely realizes \( \text{Func} \) if for any (efficient) adversary \( \mathcal{A} \) there exists an (efficient) ideal-world adversary (the simulator) \( \mathcal{S} \) such that for every (efficient) environment \( \mathcal{E} \) it holds that \( \text{EXEC}_{\pi, \mathcal{A}, \mathcal{E}} \approx \text{EXEC}_{\text{Func}, \mathcal{S}, \mathcal{E}} \), 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 \( \text{ct} \) (now consisting of \( n \) components) such that \( \text{ct} \) would not be explainable by a vector of input values \( (x_1, \ldots, x_n) \) given output values \( y_i \) (derived from \( \text{ct} \)) 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 \( \text{mpk}' \) (that has probability zero of being generated by setup) which takes all messages to special ciphertext \( \text{ct} \) that decrypts to \( \bot \). Such a scheme is obviously not setup consistent but remains consistent since \( \text{ct} \) decrypts consistently. Along the same lines, one can introduce a new special ciphertext \( \text{ct} \) 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-inCONS 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 \( \text{FE} = \{ \text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec} \} \) described in Fig. 13 is strongly input consistent for any functionality class \( \mathcal{F} = \{ F_\lambda \}_{\lambda \in \mathbb{N}} \). Namely, for any PPT adversary \( \mathcal{A} \), it holds that:
\[
\text{Pr}[	ext{st-in-CONS}^\text{FE}(1^\lambda, \mathcal{A})] \leq \text{negl}(\lambda).
\]

**Proof.** After the challenger has received two ciphertexts \( \text{ct}_1, \text{ct}_2 \) and some functional keys \( \{ \text{sk}_{f_i}, f_i \}_{i \in [n]} \) from \( \mathcal{A} \), it parses \( \text{ct}_1 = x_1, \text{ct}_2 = x_2 \) and sorts out the keys where \( \text{sk}_{f_i} \neq f_i \lor f_i \notin \mathcal{F}_\lambda \) as demanded by the decryption function. This results in the set \( \{ \text{sk}_{f_i}, f_i \}_{i \in [m]} \) 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
we distinguish between two cases. First, for each \( j \in [2] \) s.t. \( \tilde{x}_j \in \mathcal{X}_\lambda \) we have by definition
\[
\bigcap_{i \in [s]} f_i^{-1}(\text{Dec}(\text{mpk}, f_i, \text{sk}_f, \text{ct}_j)) \\
= \bigcap_{i \in [s]} f_i^{-1}(f_i(\tilde{x}_j)) \ni \tilde{x}_j.
\]
And for each \( j \in [2] \) s.t. \( \tilde{x}_j \notin \mathcal{X}_\lambda \) it holds that
\[
\bigcap_{i \in [s]} f_i^{-1}(\text{Dec}(\text{mpk}, f_i, \text{sk}_f, \text{ct}_j)) \\
= \bigcap_{i \in [s]} f_i^{-1}(\bot) = \bot.
\]

In both of these cases, the intersection remains non-empty and strong input-consistency follows. \( \square \)

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 \( \text{PKE} = (\text{PKE.Setup}, \text{PKE.Enc}, \text{PKE.Dec}) \) be an IND-CCA secure public-key encryption scheme, then the functional encryption scheme \( \text{FE} = (\text{Setup}, \text{Enc}, \text{Dec}) \) 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
\[\text{Adv}_{\text{IND-CCA}}^{\text{PKE.FE}, \mathcal{A}}(\lambda) \leq s \cdot \text{Adv}_{\text{IND-CCA}}^{\text{PKE}}(\lambda).\]

**Proof.** To prove this statement, we use a hybrid argument over the games \( G_0, \ldots, G_s \) as defined in Fig. 15. Note that \( G_0 \) corresponds to the game IND-CCA\(^{\text{PKE}}\) and game \( G_s \) to the game IND-CCA\(^{\text{FE}}\). By using the triangle inequality, we get:
\[\text{Adv}_{\text{IND-CCA}}^{\text{PKE.FE}, \mathcal{A}}(\lambda) \leq \sum_{k=1}^{s} |\text{Win}_{\mathcal{A}}^{G_0, k-1}(1^\lambda) - \text{Win}_{\mathcal{A}}^{G_0, k}(1^\lambda)|.\]

We conclude the proof by showing that for any \( k \in [s] \), there exists an adversary \( \mathcal{B}_k \) such that:
\[|\text{Win}_{\mathcal{A}}^{G_0, k-1}(1^\lambda) - \text{Win}_{\mathcal{A}}^{G_0, k}(1^\lambda)| \leq \text{Adv}_{\text{PKE.FE.B_k}}^{\text{IND-CCA}}(\lambda).\]

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 \( G_{0, k-1+\beta} \) to \( \mathcal{A} \), when interacting with the underlying IND-CCA\(^{\text{PKE}}\) experiment.

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

Whenever \( \mathcal{A} \) asks for a functional key \( \text{sk}_{f_i} \), with \( i \in [s] \setminus \{k\} \), \( \mathcal{B}_k \) outputs \( \text{sk}_i \) to \( \mathcal{A} \). If \( \mathcal{A} \) asks for the functional key \( \text{sk}_{f_k} \) the adversary \( \mathcal{B}_k \) outputs a random value \( \alpha \leftarrow \{0, 1\} \).
let $\mathcal{A}$ be an IND-CPA secure public-key encryption scheme, then the functional encryption scheme $FE = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ in Fig. 14 is not setup 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

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

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 $f_1 : X_\lambda \rightarrow \{0, 1\}$ and $f_2(x) := \overline{f_1(x)}$, where $\overline{}$ denotes the bit complement. The adversary $\mathcal{A}$ queries $\text{QDec}(f_1, ct)$ with $i \in [s] \setminus \{k\}$, $\mathcal{B}_k$ computes $f_i(x) = \text{PKE}.\text{Dec}(\text{sk}_i, ct)$ and sends $f_i(x)$ to $\mathcal{A}$. In the case $\mathcal{A}$ queries $\text{QDec}(f_2, ct)$, $\mathcal{B}_k$ 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 $\beta'$ returned by $\mathcal{A}$. Since $\mathcal{B}_k$ perfectly emulates $\mathcal{G}_{0,k-1,\beta}$ 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 $\mathcal{G}_{0,k-1,\beta}$, 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 $\text{PKE} = (\text{PKE}.\text{Setup}, \text{PKE}.\text{Enc}, \text{PKE}.\text{Dec})$ be an IND-CCA secure public-key encryption scheme, then the functional encryption scheme $FE = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ 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

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

Proof. We consider a functionality class that contains two functions $(s = 2)$, i.e. $\mathcal{F} = \{f_1, f_2\}$, with $f_1 : X_\lambda \rightarrow \{0, 1\}$ and $f_2(x) := \overline{f_1(x)}$, where $\overline{}$ denotes the bit complement. The adversary $\mathcal{A}$ generates a ciphertext $ct = (ct_1, ct_2) = (\text{Enc}(pk_1, 0), \text{Enc}(pk_2, 0))$, asks the $\text{KeyGen}$ 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 $f_1^{-1}(0) \cap f_2^{-1}(0) = f_1^{-1}(0) \cap f_1^{-1}(0) = \emptyset$, which contradicts the input-consistency requirement.

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

Theorem 9. Let $\text{PKE} = (\text{PKE}.\text{Setup}, \text{PKE}.\text{Enc}, \text{PKE}.\text{Dec})$ be an IND-CCA secure public-key encryption scheme, then the functional encryption scheme $FE = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ in Fig. 14 is not setup 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

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

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 $\text{PKE} = (\text{PKE}.\text{Setup}, \text{PKE}.\text{Enc}, \text{PKE}.\text{Dec})$ be an IND-CPA secure public-key encryption scheme,
then the functional encryption scheme \( FE = (\text{Setup, KeyGen, Enc, Dec}) \) in Fig. 16 is not input consistent for a concrete functionality class of size \( s = 2 \). Namely, there exists a PPT adversary \( A \) such that
\[
\Pr[\text{in-CONS}^{FE}(1^\lambda, A) = 1] = 1.
\]

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 \( PKE = (\text{PKE.Setup, PKE.Enc, PKE.Dec}) \) be an IND-CPA secure public-key encryption scheme, then the functional encryption scheme \( FE = (\text{Setup, KeyGen, Enc, Dec}) \) in Fig. 16 is not setup consistent for a concrete functionality class of size \( s = 2 \). Namely, there exists a PPT adversary \( A \) such that
\[
\Pr[\text{set-CONS}^{FE}(1^\lambda, A) = 1] = 1.
\]

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).

![Setup(1^\lambda):](image)

**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 be 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 \( \text{maul}^{\text{FE}}_R \) be (an efficiently computable) map such that for all plaintexts \( x \) and public parameters \( \text{mpk} \) in the range of \( \text{Setup} \), and for any fixed randomness \( r \), \( \text{maul}^{\text{FE}}_R(\text{mpk}, x; r), \text{mpk}) = \text{Enc}(\text{mpk}, R(x); r') \) 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 \( f \circ R : \mathcal{X} \rightarrow \mathcal{Y} \) is an injective map.

**Theorem 13.** If a functional encryption scheme \( FE = (\text{Setup, KeyGen, Enc, Dec}) \) for a functionality class \( \mathcal{F} \) admits an efficiently computable map \( \text{maul}^{\text{FE}}_R \) 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 \( ct \) as the encryption of \( m^\beta \). The adversary can now query the decryption oracle for function \( f \) on \( \text{maul}^{\text{FE}}_R(ct^\beta, \text{mpk}) \) to obtain the function value \( y^\beta \) (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 \( \beta \) can be guessed perfectly.

To prove the second part of the scheme we cast the ElGamal encryption scheme as an FE scheme with function class \( \mathcal{F} = \{\text{id, f}_0\} \) which is therefore CPA-secure under DDH [22]: Let \( G = (g) \) be a prime-order group (for a prime \( 2^{\lambda - 1} < q < 2^\lambda \)) with generator \( g \). More concretely, we let \( \text{mpk, sk} \) ← \( \text{gcd}(g, a) \) for a random exponent \( a \); an encryption of \( x \) is defined as \( (g^a, g^ax) \) for a random exponent \( r \); and finally, define \( \text{Dec}^{\text{FE}}(\text{mpk}, \text{sk} = \text{msk} = a, ct = (ct', ct'')) \) to return \( x \) if \( g^a \neq \text{mpk} \), and otherwise to return \( x' = ct_1 \cdot (ct''^{-a}) \). 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 \( \text{id} \in \mathcal{F} \).

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 \( F_{m,X,Y} \). Second, in Lemma 2, we show that the same scheme is input inconsistent for the functionality class \( F_{p} \).

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

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

**Proof.** For the computation of the final output in the decryption procedure, it is necessary to compute the discrete logarithm of \( \langle x, y \rangle \). As described in [7], we assume that the computed inner product lies in a polynomial bounded interval \( \{0, \ldots, L\} \), i.e. \( \langle x, y \rangle \in \{0, \ldots, L\} \) with a known \( L \). This ensures that the discrete logarithm computation can be performed in \( \tilde{O}(L^{1/2}) \), using Pollard’s kangaroo method [58] (or even \( \tilde{O}(L^{1/3}) \), by precomputing a table of size \( \tilde{O}(L^{1/3}) \) [19]). Due to correctness, we assume that for every encrypted vector \( x \), with \( ||x||_\infty < X \), and every functional key corresponding to \( y \), with \( ||y||_\infty < Y \), the decryption gives us the right output \( \langle x, y \rangle \). 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 \( A \) against the input consistency of the scheme. After the challenger has generated the parameters, \((mpk, sk) \leftarrow \text{Setup}(1^\lambda)\) and has sent \( mpk \) to the adversary, the adversary generates a ciphertext \( ct \), by encrypting the vector \( x := (L+1) \cdot e_1 \) after the rules defined in the encryption procedure. In the next step, \( A \) queries the key generation oracle for the vectors \( e_1 \) and \( e_m \), receives \( sk_{e_1} \) and \( sk_{e_m} \) as a reply and sends \( ct \) to the challenger.

After receiving \( ct \), the challenger computes \( y_1 = \text{Dec}(mpk, e_1, sk_{e_1}, ct) \) and \( y_2 = \text{Dec}(mpk, e_m, sk_{e_m}, ct) \). We consider the computation of \( y_1 \) and \( y_2 \) in more detail. In the decryption \( \text{Dec}(mpk, e_1, sk_{e_1}, ct) \), the decryptor computes \( g^{(L+1) \cdot e_1} = g^{y_1} \) 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 \( \text{Dec}(mpk, e_m, sk_{e_m}, ct) \), the decryptor computes \( g^{(L+1) \cdot e_m} = g^{y_2} = 1 \), 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. \( f_{e_1}^{-1}(\perp) \) and \( f_{e_m}^{-1}(0) \). The first preimage is defined as \( f_{e_1}^{-1}(\perp) = \{ \perp \} \) (due to Definition 2) and the second preimage as \( f_{e_m}^{-1}(0) = \{ x \in \mathbb{Z}_m^X : \langle x, e_m \rangle = 0 \} = \{ (\pi_0) \} \), with \( x \in \mathbb{Z}_m^m \). For the final step in the consistency check, we compute the intersection of the two preimages \( f_{e_1}^{-1}(\perp) \cap f_{e_m}^{-1}(0) = \{ \perp \} \cap \{ (\pi_0) \} = \emptyset \).

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 \( F_{p} \), with \( p \) prime, described in Fig. 7, then the scheme FE is not input consistent. Namely, there exists a PPT adversary \( A \) such that:

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

**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 \( g^z \) for all \( z \in \mathbb{Z}_p \) 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**

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 \( sk_u \), queried by the adversary \( A \) 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 \( (mpk, sk) \leftarrow \text{Setup}(1^\lambda) \). In the next step, the adversary \( A \) receives mpk and has access to a key generation oracle \( \text{KeyGen}(mpk, sk_m) \). Whenever \( A \) queries the key generation oracle with a vector \( y_1 \), the challenger generates \( sk_{y_1} = \text{KeyGen}(mpk, sk_m, y_1) \), adds \( (sk_{y_1}, y_1) \) to the list \( F \) and sends \( sk_{y_1} \) to \( A \). At some point in the game, \( A \) sends \( ct \) to the challenger and the challenger computes \( [z_{y_1}] := \text{Dec}(mpk, y_1, sk_{y_1}, ct) \) for all \( (sk_{y_1}, y_1) \in F \). We consider how the decryption works in more detail and determine \( [z_{y_1}] \) specifically corresponding to \( y_1 \).

For the ciphertext, output by the adversary \( A \), we write \( ct = (c_0, c_1) \), with \( c_0 := [c_0]_g \in \mathbb{G}_k^{k+1} \) and \( c_1 := [c_1]_g \in \mathbb{G}_m ^m \). To be more specific, we also write \( c_0 \in \mathbb{G}_k^{k+1} \) and \( c_1 \in \mathbb{G}_m ^m \) as explicit group elements, i.e. \( c_0 := (g^{c_0}_0, \ldots, g^{c_0}_{k+1}) \) and \( c_1 := (e_1, \ldots, e_m) \in \mathbb{Z}_p ^m \). To show that the
decryption procedure always decrypts to one underlying element, we compute the decryption procedure for an arbitrary honestly generated key \( \mathbf{sW}_{y} \). We denote the master secret key as \( \mathbf{W} := \left( \begin{array}{c} 1 \\ \vdots \\ 1 \end{array} \right) \), and correspondingly \( \mathbf{W}^\top := \left( \begin{array}{c} \mathbf{w}_1 \\ \vdots \\ \mathbf{w}_{k+1} \end{array} \right) \). Using the matrix description, the functional key is defined as \( \mathbf{W}^\top \cdot \mathbf{y} = \left( \begin{array}{c} \langle \mathbf{w}_1, \mathbf{y} \rangle \\ \vdots \\ \langle \mathbf{w}_{k+1}, \mathbf{y} \rangle \end{array} \right) \in \mathbb{Z}_p^{k+1} \).

For the decryption, we need to compute two different components: \( c_{0}^\top \mathbf{sW}_{y} \) and \( c_{1}^\top \mathbf{y} \).

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

For the final step of the decryption procedure, the discrete logarithm computation happens. We denote the final decryptions with respect to the different \( \mathbf{y} \)'s by \( z_i := \log (g^{\langle \mathbf{W} \cdot \mathbf{c}_0 + c_{1}^i, \mathbf{y} \rangle}) \).

To prove the input consistency, we show that the preimage of \( z_i \) contains the value \( \mathbf{W} \cdot \mathbf{c}_0 + c_{1}^i \) for the case that \( z_i = \text{oob} \) and the case that \( z_i \neq \text{oob} \). This leads to the fact that \( \mathbf{W} \cdot \mathbf{c}_0 + c_{1}^i \in f_{y}^{-1}(\text{oob}) \) for all \( z_i \in \{0, \ldots, L\} \cup \{\text{oob}\} \), 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 \( f_{y}^{-1}(\text{oob}) \) contains all the vectors \( \mathbf{x} \) such that \( \langle \mathbf{x}, \mathbf{y} \rangle > L \). For the case that \( z_i = \text{oob} \) it holds that \( \langle \mathbf{W} \cdot \mathbf{c}_0 + c_{1}^i, \mathbf{y} \rangle < L \), after the analysis above, and therefore \( \mathbf{W} \cdot \mathbf{c}_0 + c_{1}^i \in f_{y}^{-1}(\text{oob}) \). For the case that \( z_i \in \{0, \ldots, L\} \) it holds that the preimage contains all the vectors \( \mathbf{x} \) such that \( \langle \mathbf{x}, \mathbf{y} \rangle = z_i \). Therefore, again with the analysis above, it follows that \( \mathbf{W} \cdot \mathbf{c}_0 + c_{1}^i \in f_{y}^{-1}(z_i) \) for \( z_i \in \{0, \ldots, L\} \). Overall, this leads to the fact that \( \mathbf{W} \cdot \mathbf{c}_0 \in \bigcap_{i \in [n]} f_{y}^{-1}(z_i) \) for all \( i \in [n] \) with \( z_i \in \{0, \ldots, L\} \cup \{\text{oob}\} \).

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 \( \mathcal{E} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}) \) be the IND-CPA secure functional encryption scheme for the functionality class \( \mathcal{F}_{p,l}^m \), with \( p \) prime, described in Fig. 7. Then the functional encryption scheme \( \mathcal{E}' = (\text{Setup}', \text{KeyGen}', \text{Enc}', \text{Dec}') \) 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

\[
\text{Adv}_{\mathcal{I}, \mathcal{E}, \mathcal{A}}^{\text{IND-CPA}}(\lambda) \leq \text{Adv}_{\mathcal{I}, \mathcal{E}, \mathcal{B}}^{\text{IND-CPA}}(\lambda).
\]

**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 \( g^{\langle x, y \rangle} \). 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 \( \mathbf{cT} \) under different functional keys \( \mathbf{sW}_{y} \) 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 \((\mathbf{mpk}, \mathbf{msk}) = ((\mathbf{G}, [\mathbf{A}]_{g}, [\mathbf{WA}]_{g}), \mathbf{W}) \leftarrow \text{Setup}(\lambda)) \). In the next step, the adversary \( \mathcal{A} \) receives \( \mathbf{mpk} \) and has access to a key generation oracle \( \text{KeyGen} (\mathbf{mpk}, \mathbf{msk}, \cdot) \). Whenever \( \mathcal{A} \) queries the key generation oracle with a vector \( \mathbf{y} \), the challenger generates \( \mathbf{sW}_{y} = \text{KeyGen}(\mathbf{mpk}, \mathbf{msk}, \mathbf{y}) \), adds \( (\mathbf{sW}_{y}, \mathbf{y}) \) to the list \( \mathcal{F} \) and sends \( \mathbf{sW}_{y} \) to \( \mathcal{A} \). At some point in the game, \( \mathcal{A} \) sends \( \mathbf{cT} \) to the challenger and the challenger computes \( [z]_{g} := \text{Dec}(\mathbf{mpk}, \mathbf{y}, \mathbf{sW}_{y}, \mathbf{ct}) \) for all \( (\mathbf{sW}_{y}, \mathbf{y}) \in \mathcal{F} \). We consider how the decryption works in more detail and determine \( [z]_{g} \) specifically corresponding to \( \mathbf{y} \).

For the ciphertext, output by the adversary \( \mathcal{A} \), we write \( \mathbf{cT} = (c_{0}^i, c_{1}^i) \), with \( c_{0}^i := [c_{0}]_{g} \in \mathbb{G}^{k+1} \) and \( c_{1}^i := [c_{1}]_{g} \in \mathbb{G}^{m} \).

To be more specific, we also write \( c_{0}^i \in \mathbb{G}^{k+1} \) and \( c_{1}^i \in \mathbb{G}^{m} \) as explicit group elements, i.e. \( c_{0}^i := \left( \begin{array}{c} g_{0,0}^\lambda \\ \vdots \\ g_{0,k+1}^\lambda \end{array} \right) \) and \( c_{1}^i := \left( \begin{array}{c} g_{1,0}^\lambda \\ \vdots \\ g_{1,m}^\lambda \end{array} \right) \) with the generator \( g \), \( c_{0} := (c_{0,0}, \ldots, c_{0,k+1}) \in \mathbb{Z}_p^{k+1} \) and \( c_{1} := (c_{1,0}, \ldots, c_{1,m}) \in \mathbb{Z}_p^{m} \). To show that the decryption procedure always decrypts to one underlying
element, we compute the decryption procedure for an arbitrary honestly generated key $sk_y$. We denote the master secret key as $W := \begin{pmatrix} w_0^1 & \cdots & w_{k+1}^1 \end{pmatrix}$, and correspondingly $W^\top := \left( \begin{array}{ccc} -w_0 \vdots & \cdots & -w_{k+1} \end{array} \right)$. Using the matrix description, the functional key is defined as $W^\top \cdot y = \begin{pmatrix} \langle w_0^1, y \rangle \\ \vdots \\ \langle w_{k+1}^1, y \rangle \end{pmatrix} \in \mathbb{Z}_p^{k+1}$.

For the decryption, we need to compute two different components: $[c_0^\top sk_y]_g$ and $[c_1^\top y]_g$.

First, we describe how to compute $[c_0^\top sk_y]_g$: We exponentiate all of the components of $c_0$ with the components of $sk_y$ and compute the product of the resulting vector components. In more detail, we compute $\prod_{i \in [k+1]} g^{c_0 \cdot i} \cdot \langle w_i^\top, y \rangle = g^{\sum_{i \in [k+1]} (c_0 \cdot i) \cdot w_i^\top} = g^{\langle W \cdot c_0, y \rangle}$. We proceed in the same way for the second component $\prod_{i \in [m]} g^{c_1 \cdot i} \cdot y_i = g^{\langle W \cdot c_1 + c_1, y \rangle}$.

To generate the final output, we need to multiply the two components, which results in $g^{\langle W \cdot c_0, y \rangle} \cdot g^{\langle c_1, y \rangle} = g^{\langle W \cdot c_0 + c_1, y \rangle}$.

Coming back to the initial description of the game, the decryption procedure outputs $[z_i]_g := \langle [W \cdot c_0 + c_1, y] \rangle_g$. Due to the fact that the vectors $c_0$ and $c_1$ are set by the adversary and the matrix $W$ is fixed after the setup procedure, the decryption relies only on the value $y$. This results in the decryptions $g^{\langle W \cdot c_0 + c_1, y \rangle}, \ldots, g^{\langle W \cdot c_0 + c_1, y \rangle}$ for all the different secret key queries $y_i$. Consequently, $W \cdot c_0 + c_1 \in \bigcap_{i \in [m]} f_{g,y_i}^{-1}([z_i]_g)$, which further implies that $W \cdot c_0 + c_1 \in \bigcap_{i \in [m]} f_{g,y_i}^{-1}([z_i]_g)$.

This makes the intersection non-empty for every possible ciphertext $ct$ generated by $A$. 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 $FE = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ 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 $FE' = (\text{Setup}', \text{KeyGen}', \text{Enc}', \text{Dec}')$ for the functionality class $\mathcal{P}_p^m$, with $p$ prime, described in Fig. 7 is IND-CPA secure. Namely, for any PPT adversary $A$, there exists a PPT adversary $B$ such that

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

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

**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.

![Fig. 17: Input consistency compiler](image)

**Theorem 16.** Let $FE = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ be a functional encryption scheme and $NIZK = (\text{NIZK.Setup}, \text{NIZK.Prove}, \text{NIZK.Verify})$ a NIZK proof system for relation $R_{in}$, then the construction $FE' = (\text{Setup}', \text{KeyGen}', \text{Enc}', \text{Dec}')$ defined in Figure 17 satisfies input consistency.
Namely, for any PPT adversary $A$, there exists a PPT adversary $B$ such that:

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

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

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

Whenever $A$ asks a query to $B$, $B$ computes the key $s_k \leftarrow \text{KeyGen}(\text{msk}, f)$, adds $(s_k, f)$ to the list $F$ and sends $s_k, f$ to $A$.

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

Let us analyze the output of $A$ 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 $A$ performs a consistency attack under the assumption that $(\text{mpk}, \text{ct}) \in L$ 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 $y_i \neq \bot$ for all $i \in [n]$ (this is covered by event $E_1$). Furthermore we show that if $y_i \neq \bot$ then there exists an $x$ such that $x \in \bigcap_{i \in [n]} f^{-1}_i(y_i)$ (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 $\bot$, i.e. $y_i \neq \bot$. We distinguish between two cases:

1. It holds that $y_i = \bot$ for all $i \in [n]$. In this case, the adversary $A$ did not perform a consistency attack. In more detail, the intersection $\bigcap_{i \in [n]} f^{-1}_i(y_i)$ will contain the $\bot$ value.
2. At least one, but not all, of the decryptions are equal to $\bot$, i.e. $y_i = \bot$. Since $\bot$ 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 $((\text{mpk}, \text{ct}), w) \in R_{in}$ (i.e. it cannot be a valid instance). This is a contradiction to the assumption that $(\text{mpk}, \text{ct}) \in L$.

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

For the analysis of event $E_2$, we assume, for the sake of contradiction, that the intersection $\bigcap_{i \in [n]} f^{-1}_i(y_i)$ is empty and it holds (with respect to event $E_1$) that $y_i \neq \bot$ for all $i \in [n]$. In this case, the adversary $A$ has generated a valid proof $\pi$ for an $x \notin L_{in}$. Again, by the perfect correctness of the FE scheme, the adversary $B$ broke the soundness of the NIZK scheme, because it has found a ciphertext $\text{ct}$ 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 $(\text{mpk}, \text{ct}) \in L$, it is not possible for an adversary $A$ 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 $(\text{mpk}, \text{ct}) \notin L$.

This yields the bound

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

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

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

**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$_0^{\text{FE}}(1^\lambda, A)$ and $G_3$ to the game IND-CPA$_1^{\text{FE}}(1^\lambda, A)$. This results in:

$$\text{Adv}^{\text{IND-CPA}}_{\text{FE}',A}(1^\lambda) = |\text{Win}^{G_0}_{A}(1^\lambda) - \text{Win}^{G_3}_A(\lambda)|.$$
Putting everything together, we obtain the theorem.

**Lemma 3** (Transition from \(G_0\) to \(G_1\).) For any PPT adversary \(A\), there exists a PPT adversary \(B_0\) such that

\[
|\text{Win}^{G_0}_A(1^\lambda) - \text{Win}^{G_1}_A(1^\lambda)| \leq Adv^{\text{IND-CPA}}_{\text{NIZK},B_0}(\lambda) .
\]

**Proof.** We build an adversary \(B_0\) that simulates \(G_0\) towards \(A\) when interacting with the underlying ZK\_NIZK experiment.

In the beginning of the reduction, \(B_0\) receives CRS from the ZK\_NIZK experiment. It generates a functional encryption instance \((mpk,msk) \leftarrow \text{Setup}(1^\lambda)\), sets \(mpk' = (\text{CRS}, mpk)\) and gives \(mpk'\) to the adversary.

Whenever \(A\) asks an encryption query \((x^0,x^1)\), \(B_0\) generates the ciphertext \(ct = \text{Enc}(mpk,x^0;\tau)\) with \(r \leftarrow \{0,1\}^\lambda\) and sends \(y = (mpk,ct)\) and \(w = (x,\tau)\) as a statement-witness pair to its challenger. As an answer, \(B_0\) receives a proof \(\pi\) for \(R_m\). It sets \(ct' = (ct,\pi)\) and sends it to \(A\).

For a key generation query \(f\), \(B_0\) generates \(sk_f \leftarrow \text{KeyGen}(mpk,msk,f)\) for and sends \(sk_f' = sk_f\) as a reply to \(A\).

This covers the simulation of the game \(G_\beta\). Finally \(B_0\) outputs the same bit \(\beta'\) returned by \(A\). It follows, from the perfect simulation, that the advantage of \(B_0\) is the same as the advantage of \(A\).

**Lemma 4** (Transition from \(G_1\) to \(G_2\).) For any PPT adversary \(A\), there exists a PPT adversary \(B_1\) such that

\[
|\text{Win}^{G_1}_A(1^\lambda) - \text{Win}^{G_2}_A(1^\lambda)| \leq Adv^{\text{IND-CPA}}_{\text{FE,B}_1}(\lambda) .
\]

**Proof.** We build an adversary \(B_1\) that simulates \(G_{1+\beta}\) towards \(A\) when interacting with the underlying IND-CPA\_FE experiment.

In the beginning of the reduction, \(B_1\) receives \(mpk\) from the experiment. It simulates a CRS, i.e. \((\text{CRS},\tau) \leftarrow S_1(1^\lambda)\), sets \(mpk' = (\text{CRS}, mpk)\) and gives \(mpk'\) to the adversary.

Whenever \(A\) asks an encryption query \((x^0,x^1)\), \(B_1\) forwards it to its own encryption oracle to receive \(ct \leftarrow \text{Enc}(mpk,x^0)\), simulates a proof for the relation \(R_m\), i.e. \(\pi \leftarrow S_2(\text{CRS},\tau,x^0)\) and sends \(ct' = (ct,\pi)\) to \(A\).

For a key generation query \(f\), \(B_1\) queries its own key generation oracle on \(f\) to receive \(sk_f' \leftarrow \text{KeyGen}(mpk,msk,f)\), sets \(sk_f' = sk_f\) and sends \(sk_f'\) to \(A\).

This covers the simulation of the game \(G_{1+\beta}\). Finally \(B_1\) outputs the same bit \(\beta'\) returned by \(A\). It follows, from the perfect simulation, that the advantage of \(B_1\) is the same as the advantage of \(A\).
say $c_{t_i}$, it internally runs $A_2^*$ on input $(c_{t_i}, \pi)$, where
$\pi$ is a simulated proof for the relation $R_{ni}$. Finally, $A_2$ outputs whatever $A_2^*$ outputs. We see that the output distribution of $\text{Real}^{\text{FE}}(1^\lambda, A)$ is identical to the output of $A'$ in experiment $G_1$ and the output distribution of $\text{Ideal}^{\text{FE}}(1^\lambda, A, S)$ is identical to the output distribution of $G_2$, the ideal experiment with $\text{FE}'$ and simulator $S'$. 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 $c_{t_i}$ 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 $\text{FE} = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec})$ be an IND-CPA secure functional encryption scheme and $\text{NIZK} = (\text{NIZK.Setup}, \text{NIZK.Prove}, \text{NIZK.Verify})$ a NIZK proof system satisfying one-time simulation soundness, then the construction $\text{FE}' = (\text{Setup}', \text{KeyGen}', \text{Enc}', \text{Dec}')$, defined in Figure 20, is IND-CCA secure. Namely, for any PPT adversary $A$, there exist PPT adversaries $B, B'$ and $B''$, such that:

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

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 IND-CCA$^{0}_A(1^\lambda, \mathcal{A})$ and $G_4$ to the game IND-CCA$^{1}_A(1^\lambda, \mathcal{A})$. This results in:

$$\text{Adv}_{\text{FE}', A}^{\text{IND-CCA}}(\lambda) = |\text{Win}_{\text{A}}^{G_0}(1^\lambda) - \text{Win}_{\text{A}}^{G_1}(1^\lambda)|.$$

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 $B_0$ such that:

$$|\text{Win}_{\text{A}}^{G_0}(1^\lambda) - \text{Win}_{\text{A}}^{G_1}(1^\lambda)| \leq \text{Adv}_{\text{NIZK}, B_0}^{\text{ZK}}(\lambda).$$

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. $\text{ct} = (\text{Enc}(\text{mpk}_1, x^0), \text{Enc}(\text{mpk}_2, x^0), \text{Enc}(\text{mpk}_1, x^1), \text{Enc}(\text{mpk}_2, x^1))$. 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 6, we exhibit PPT adversaries $B_1$ and $B_2$ such that:

$$|\text{Win}_{\text{A}}^{G_1}(1^\lambda) - \text{Win}_{\text{A}}^{G_2}(1^\lambda)| \leq \text{Adv}_{\text{NIZK}, B_1}^{\text{SIM-SOUND}}(\lambda) + \text{Adv}_{\text{FE}', B_2}^{\text{IND-CPA}}(\lambda).$$

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. $\text{ct} = (\text{Enc}(\text{mpk}_1, x^1), \text{Enc}(\text{mpk}_2, x^1), \text{Enc}(\text{mpk}_1, x^0), \text{Enc}(\text{mpk}_2, x^0))$. The transition from $G_2$ to $G_3$ is almost symmetric to the transition from $G_1$ 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 $B_3$ such that:

$$|\text{Win}_{\text{A}}^{G_2}(1^\lambda) - \text{Win}_{\text{A}}^{G_3}(1^\lambda)| \leq \text{Adv}_{\text{FE}, B_3}^{\text{IND-CPA}}(\lambda).$$

Putting everything together, we obtain the theorem. □

Lemma 5 (Transition from $G_0$ to $G_1$). For any PPT adversary $A$, there exists a PPT adversary $B_0$ such that:

$$|\text{Win}_{\text{A}}^{G_0}(1^\lambda) - \text{Win}_{\text{A}}^{G_1}(1^\lambda)| \leq \text{Adv}_{\text{NIZK}, B_0}^{\text{ZK}}(\lambda).$$

Proof. We build an adversary $B_0$ that simulates $G_0$ towards $A$ when interacting with the underlying ZK$^{\text{NIZK}}_B$ experiment.

In the beginning of the reduction, $B_0$ receives CRS from the ZK$^{\text{NIZK}}_B$ experiment. It generates two functional encryption instance $(\text{mpk}_i, \text{msk}_i) \leftarrow \text{Setup}(1^\lambda)$ for $i \in [2]$, sets $\text{mpk}' = (\text{CRS}, \{\text{mpk}_i\}_{i \in [2]})$ and gives $\text{mpk}'$ to the adversary.

Whenever $A$ asks an encryption query $(x^0, x^1)$, $B_0$ generates the ciphertext $c_{t_i} = (\text{Enc}(\text{mpk}_i, x^0; r_i), \text{Enc}(\text{mpk}_i, x^1; r_i))$ with $r_i \triangleright \left\{0, 1\right\}^\lambda$ for $i \in [2]$ and sends $y = (\text{mpk}'_i, \text{ct}_i)_{i \in [2]}$ and $w = (x, \{r_i\}_{i \in [2]})$ as a statement-witness pair to its challenger. As an answer, $B_0$ receives a proof $\pi$ for the relation $R_{\text{FE}'}$. It sets $\text{ct}' = (\{\text{ct}_i\}_{i \in [2]}, \pi)$ and sends it to $A$.

For a key generation query $f$, $B_0$ generates $\text{sk}_{f,i} \leftarrow \text{KeyGen}(\text{mpk}_i, \text{msk}_i, f)$ for $i \in [2]$ and sends $\text{sk}'_f = \{\text{sk}_{f,i}\}_{i \in [2]}$ as a reply to $A$. 

30
Whenever \( A \) submits a decryption query \((f, ct')\), with \( ct' = (\{ct_i\}_{i \in [2]}, \pi')\), \( B_0 \) generates the functional key \( sk_{f,1} \leftarrow \text{KeyGen}(\text{mpk}_1, \text{msk}_1, f) \) and executes \( \text{NIZK.Verify}(\text{CRS}, (\text{mpk}_1, ct_i)_{i \in [2]}) = 1 \). \( B_0 \) computes \( y := \text{Dec}(\text{mpk}_1, f, sk_{f,1}, ct_1) \) and sends \( y \) to \( A \). Otherwise, \( B_0 \) sends \( \perp \) to \( A \).

This covers the simulation of the game \( G_\beta \). Finally \( B_0 \) outputs the same bit \( \beta' \) returned by \( A \). It follows, from the perfect simulation, that the advantage of \( B_0 \) is the same as the advantage of \( A \).

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 \( A \) participating in \( G_{1+\beta} \) for \( \beta \in \{0,1\} \), the probability that, during the experiment, \( A \) queries its decryption oracle \( \text{QDec} \) with a function-ciphertext-pair that is not explainable but has an accepting proof is negligible. Namely, we exhibit a PPT adversary \( B_1 \), such that

\[
\Pr \left[ \exists (f, \{ct_i\}_{i \in [2]}, \pi') \in Q : \\
\{(ct'_i)_{i \in [2]}, \pi'\} \neq (\{ct_i\}_{i \in [2]}, \pi), \\
\text{NIZK.Verify}(\text{CRS}, (\{ct_i\}_{i \in [2]}, \pi') = 1 \text{ and} \\
\text{Dec}(\text{mpk}_1, sk_{f,1}, ct'_1) \neq \text{Dec}(\text{mpk}_2, sk_{f,2}, ct'_2) \right] \\
\leq \text{Adv}^{\text{Sim-Sound}}_{\text{NIZK.B1}}(\lambda),
\]

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

**Proof.** We build an adversary \( B_1 \) that simulates \( G_{1+\beta} \) towards \( A \) when interacting with the underlying one time simulation-soundness experiment.

After the adversary \( B_1 \) has received \( \text{CRS} \) from the underlying experiment, it generates \((\text{mpk}_1, \text{msk}_1) \leftarrow \text{Setup}(\lambda)\) for \( i \in [2] \), sets \( \text{mpk}' := (\text{CRS}, \{\text{mpk}_i\}_{i \in [2]}) \) and sends \( \text{mpk}' \) to \( A \). Whenever \( A \) submits a key generation query \( f \), \( B_1 \) generates the functional keys \((sk_{f,1} \leftarrow \text{KeyGen}(\text{mpk}_1, \text{msk}_1, f))\) for \( i \in [2] \), sets \( sk'_f = (sk_{f,i})_{i \in [2]} \) and sends it to \( A \).

For the challenge query \((x^0, x^1)\) asked by \( A \), \( B_1 \) computes \( ct_1 = \text{Enc}(\text{mpk}_1, x^0) \) and \( ct_2 = \text{Enc}(\text{mpk}_2, x^1) \) (where \( \beta = 0 \) in game \( G_1 \) and \( \beta = 1 \) in \( G_2 \)) and asks its experiment for a simulated proof \( \pi \) of the statement \((\text{mpk}_1, ct_i)_{i \in [2]} \). It sets \( ct' := (\{ct_i\}_{i \in [2]}, \pi) \) and sends \( ct' \) to \( A \).

Whenever \( A \) outputs a decryption query \((f, ct' := (\{ct_i\}_{i \in [2]}, \pi))\), \( B_1 \) verifies the proof. If the output of the verification is 1, \( B_1 \) computes \( y_{f,1} := \text{Dec}(\text{mpk}_1, sk_{f,1}, ct_1) \) and \( y_{f,2} := \text{Dec}(\text{mpk}_2, sk_{f,2}, ct_2) \). If \( y_{f,1} \neq y_{f,2} \), \( B_1 \) sends \((\{ct_i\}_{i \in [2]}, \pi)\) as a proof forgery to its challenger. Otherwise it sends \( y_{f,1} \) to \( A \). If the verification outputs 0, \( B_1 \) sends \( \perp \) to \( A \).

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 \( A \), there exist PPT adversaries \( B_1 \) and \( B_2 \), such that

\[
|\text{Win}_{A}^{G_1}(\lambda) - \text{Win}_{A}^{G_2}(\lambda)| \leq \text{Adv}^{\text{Sim-Sound}}_{\text{NIZK.B1}}(\lambda) + \text{Adv}^{\text{IND-CPA}}_{\text{FE.B2}}(\lambda).
\]

**Proof.** We build an adversary \( B_2 \) that simulates \( G_{1+\beta} \) to \( A \) when interacting with the underlying IND-CPA experiment.

In the beginning of the reduction, \( B_2 \) receives \( \text{mpk}_1 \) from the underlying experiment. It simulates a \( \text{CRS} \), i.e. \((\text{CRS}, \tau) \leftarrow S_1(\lambda)\), generates a functional encryp-
Fig. 21: Relation used in the advanced input consistency compiler.

Relation $R_{in}^{CCCA}$:

Instance: $z = (mpk_i, ct_i)_{i \in [2]}$
Witness: $w = (x, \{r_i\}_{i \in [2]}, x \in X$, random coins $r_i$

$R_{in}^{CCCA}(z, w) = 1$ if and only if:

$ct_i = Enc(mpk_i, x; r_i)$, for both $i \in [2]$

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

<table>
<thead>
<tr>
<th>Setup'(1^\lambda)</th>
<th>CRS ← NIZK.Setup(1^\lambda)</th>
</tr>
</thead>
<tbody>
<tr>
<td>For $i \in [2]$</td>
<td>$(mpk_i, sk_i) ← Setup(1^\lambda)$</td>
</tr>
<tr>
<td>Return $(mpk', sk')$</td>
<td>$= ((mpk_i)<em>{i \in [2]}, {msk_i}</em>{i \in [2]})$</td>
</tr>
<tr>
<td>KeyGen'(mpk', sk', f)</td>
<td>Parse $mpk' := (mpk_i)<em>{i \in [2]}$, $msk' := {msk_i}</em>{i \in [2]}$</td>
</tr>
<tr>
<td></td>
<td>For $i \in [2]$ $sk_{f,i} = KeyGen(mpk_i, sk_i, f)$</td>
</tr>
<tr>
<td></td>
<td>Return $sk_f' := {sk_{f,i}}_{i \in [2]}$</td>
</tr>
<tr>
<td></td>
<td>$Enc'(mpk', x)$ :</td>
</tr>
<tr>
<td></td>
<td>Parse $mpk' := (mpk_i)_{i \in [2]}$</td>
</tr>
<tr>
<td></td>
<td>For $i \in [2]$ $ct_i = Enc(mpk_i, x; r_i) \text{ with } r_i \leftarrow {0, 1}$</td>
</tr>
<tr>
<td></td>
<td>If $\exists i \in [2] : ct_i = \bot$, return $err$</td>
</tr>
<tr>
<td></td>
<td>$\pi ← NIZK.Prove(CRS, (mpk_i, ct_i)<em>{i \in [2]}, (x, {r_i}</em>{i \in [2]}))$</td>
</tr>
<tr>
<td></td>
<td>if $R_{in}^{CCA}(\text{Fig. 21})$</td>
</tr>
<tr>
<td></td>
<td>If NIZK.Verify$(CORS, (mpk_i, ct_i)_{i \in [2]}, \pi) = 0$, return $err$</td>
</tr>
<tr>
<td></td>
<td>Return $ct' = ({ct_i}_{i \in [2]}, \pi)$</td>
</tr>
<tr>
<td></td>
<td>Dec$(mpk', f, sk_f', ct')$ :</td>
</tr>
<tr>
<td></td>
<td>Parse $mpk' := (mpk_i)<em>{i \in [2]}$, $sk_f' := {sk</em>{i,f}}<em>{i \in [2]}$, $ct' := {ct_i}</em>{i \in [2]}$, $\pi$</td>
</tr>
<tr>
<td></td>
<td>If NIZK.Verify$(CORS, (mpk_i, ct_i)<em>{i \in [2]}, \pi) = 1$, return $Dec(mp</em>{k1}, f, sk_{f,1}, ct_1)$</td>
</tr>
<tr>
<td></td>
<td>Else Return $\bot$</td>
</tr>
</tbody>
</table>

Fig. 21: Relation used in the advanced input consistency compiler.
We note that it is an interesting research direction to verify functional encryption (VFE) extends standard functional encryption by two additional requirements. Due to this modular reduction, we also directly inherit their compiler, and in general any compiler that achieves verifiable functional encryption 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 \( VFE = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{VerifyCT}, \text{VerifySK}) \) be a verifiable functional encryption scheme then the construction \( VFE' = (\text{Setup}', \text{KeyGen}', \text{Enc}', \text{Dec}') \) defined in Figure 23 is strongly input consistent. Namely, for any PPT adversary \( A \), it holds that:

\[
\Pr[\text{st-in-CONS}^{VFE'}(1^{\lambda}, A) = 1] = 0 .
\]

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

In the beginning of the reduction, \( B \) receives a master public key \( mpk \), two ciphertexts \( ct_1, ct_2 \), and a tuple of secret keys with the corresponding functions \( \{(sk_j, f_j)\}_{j \in [n]} \) from \( A \). In the next step, \( B \) computes:

\[
y_{j,i} := \text{Dec}(mpk', f_j, sk_j, ct_i) \quad \text{for all} \ j \in [n], i \in \{1, 2\}
\]

and defines the set \( F \) as all the functional keys that do not output \( \diamond \), i.e., \( F := \{(sk_j, f_j)\}_{j \in [n]} \setminus \{(y_{j,i} \neq 0 \lor y_{j,i} 
eq \diamond)\} \). Let \( E \) denote the event that the intersection \( \bigcap_{j \in [n]} \{y_{j,i} \neq 0 \lor y_{j,i} \neq \diamond\} \) 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

---

### Table

<table>
<thead>
<tr>
<th>Game</th>
<th>CRS &amp; ( \pi )</th>
<th>ct</th>
<th>justification/remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G_0 )</td>
<td>( \text{CRS} \leftarrow \text{NIZK.Setup}(1^\lambda) ) ( \pi \leftarrow \text{NIZK.Prove}((\text{CRS}, w)) ) ( \text{Enc}(mpk_1, x^0) ) ( \text{Enc}(mpk_2, x^0) )</td>
<td></td>
<td>Zero-knowledge of NIZK</td>
</tr>
<tr>
<td>( G_1 )</td>
<td>( \text{CRS} \leftarrow S_1(1^\lambda) ) ( \pi \leftarrow S_2((\text{CRS}, x, w)) ) ( \text{Enc}(mpk_1, x^0) ) ( \text{Enc}(mpk_2, x^0) )</td>
<td></td>
<td>IND-CPA of FE and simulation-soundness of NIZK</td>
</tr>
<tr>
<td>( G_2 )</td>
<td>( \text{CRS} \leftarrow S_1(1^\lambda) ) ( \pi \leftarrow S_2((\text{CRS}, x, w)) ) ( \text{Enc}(mpk_1, x^1) ) ( \text{Enc}(mpk_2, x^1) )</td>
<td></td>
<td>IND-CPA of FE</td>
</tr>
<tr>
<td>( G_4 )</td>
<td>( \text{CRS} \leftarrow \text{NIZK.Setup}(1^\lambda) ) ( \pi \leftarrow \text{NIZK.Prove}((\text{CRS}, w)) ) ( \text{Enc}(mpk_1, x^1) ) ( \text{Enc}(mpk_2, x^1) )</td>
<td></td>
<td>Zero-knowledge of NIZK</td>
</tr>
</tbody>
</table>

---

**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:

- **VerifyCT(mpk, ct):** A predicate on ciphertexts (w.r.t. the public key) that decides whether ct is valid.
- **VerifySK(mpk, f, 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 VerifyCT(mpk, ct) = VerifySK(mpk, f, sk) = 1, we have Pr[Dec(mpk, f, sk, ct) = f(x)] = 1 (where the implication holds as stated. Now, we analyze the different scenarios in...
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 $y_{j,i} \neq \bot$ for a single $j \in [n]$ and a fixed $i \in \{0,1\}$, then $y_{j,i} \neq \bot$ for all $j \in [n]$ and the same fixed $i \in \{0,1\}$. Event $E_2$ denotes the case that if $y_{j,i} \neq \circ$ for a single $i \in \{0,1\}$ and a fixed $j \in [n]$, then $y_{j,i} \neq \circ$ for both $i \in \{0,1\}$ and the same fixed $j \in [n]$. The final event, event $E_3$, denotes the case that if $y_{j,i} \notin \{\bot, \circ\}$, then $y_{j,i} = f_j(x_i)$.

We start by analyzing event $E_1$. Let one of the $y_{j,i} = \bot$ for a single $j \in [n]$ and a fixed $i \in \{1,2\}$. Now, if $\text{VerifyCT}$ was satisfied by $ct_i$, then by the verifiability condition, $\bot$ is a valid output and thus by definition of the special symbol $\bot$, the only “preimage” explaining the output is $\bot$ 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 $\bot$.

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

For event $E_2$, let one of the $y_{j,i} = \circ$ for a single $i \in \{1,2\}$ and a fixed $j \in [n]$. If this case occurs, then the $\text{VerifySK}$ algorithm must have output 0 for the functional key $sk_j$ for a fixed $j$ as otherwise, since by definition of the special symbol $\circ$, there is no “preimage” explaining the output (and thus the verifiability property violated). As before, the $\text{VerifySK}$ algorithm also outputs 0 if it gets queried using another ciphertext $ct_i$ but the same functional key $sk_j$. This yields that $y_{j,i'} = \circ$ 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 $y_{j,i} \notin \{\bot, \circ\}$, then both of the verify algorithms, $\text{VerifyCT}$ and $\text{VerifySK}$, output 1 in the decryption procedure. This ensures, together with the verifiability property, i.e. $\Pr[\text{Dec}(mpk, f_j, sk, ct)] = f_j(x_i) = 1$, 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 $sk_{j'}$, is unequal to $\circ$ (due to event $E_4$ and the definition of $F$) and unequal to $\bot$ (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 do the same analysis for all the remaining $y_{j',i}$ with $j'' \in [n] \setminus \{j, j'\}$ and therefore it follows that $\bigcap_{j \in [n], f_j \in F} F_{j'}^{-1}(y_{j,i}) \neq \emptyset$. The same analysis also needs to be done for the second ciphertext $ct_i$ 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 $\text{Dec}$ 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 $\text{Dec}$ with access to at least one secret key. Thus, no guaranteed verifiability algorithm of the form $\text{VerifyCT}(mpk, ct)$ (or $\text{VerifySK}(mpk, f, sk_j)$) 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 potentially be violated even though finding these values by an efficient adversary $A$—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 $B$ that successfully generates a valid proof for a statement not in the language by assuming an adversary $A$ for the setup consistency experiment Set-CONS.

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

Let us analyze the output of $A$ 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 $A$ performs a consistency attack under the assumption that $(\{\text{mpk}'_i\}_{i \in [\lambda]}, \{\text{sk}_i\}_{i \in [\lambda]}, f) \in L$ and show that the occurrence of $E$ would contradict the assumption. Let us compute $\text{ct}_i' := (\text{mpk}', \{\text{ct}_j\}_{j \in [\lambda]} : \text{Enc}''(\text{mpk}', x_i))$ for $i \in [2]$ and $y_i = \text{Dec}'(\text{mpk}'', f, \text{sk}, \text{ct}_i')$ for $i \in [2]$. For concreteness, assume that both ciphertexts are not equal to $\text{err}$ (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 $y_i \neq \text{err}$, we denote this by event $E_1$, enforces that $y_i = f(x_i)$ for all $i \in [2]$ and furthermore that $y_i \neq \text{err}$ for all $i \in [2]$.

In the case of event $E_1$, we assume $y_i \neq \text{err}$ (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 $(\{\text{mpk}'_i\}_{i \in [\lambda]}, \{\text{sk}_i\}_{i \in [\lambda]}, f) \in L$ 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 $\text{sk}_i$ and $\text{sk}_j$, for $i \neq j$ are correctly generated and matching to the master public keys $\text{mpk}_i$ and $\text{mpk}_j$ in the encryption (note that by the definition of $\text{Dec}'$ that decryption is only performed if both $\text{Enc}'$ and $\text{Dec}'$ use the same triple $(\text{mpk}_1, \text{mpk}_2, \text{mpk}_3)$) and thus $y_{f,1}^{(i)} \leftarrow \text{Dec}(\text{mpk}_1, f, \text{sk}_i, \text{ct}_1)$ and $y_{f,1}^{(j)} \leftarrow \text{Dec}(\text{mpk}_2, f, \text{sk}_j, \text{ct}_1)$ are equal to $f(x_1)$. Therefore also the majority of the decryption for $ct_1$ is equal to $f(x_1)$ and the final decryption outputs $f(x_1)$. Hence assuming $y_1 \neq \text{err}$ implies $y_1 = f(x_1)$.

For event $E_2$, we need to show that $y_1 \neq \text{err}$ and $y_2 \neq \text{err}$ in the case of a consistency attack and under the assumption that $(\{\text{mpk}'_i\}_{i \in [\lambda]}, \{\text{sk}_i\}_{i \in [\lambda]}, f) \in L$. We start by considering the case that $y_1 = y_2 = \text{err}$. If $y_1 = y_2 = \text{err}$, then the adversary $A$ 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 $y_1 \neq \text{err}$ and $y_2 \neq \text{err}$ (or $y_1 = \text{err}$ and $y_2 \neq \text{err}$ respectively). If $y_1 \neq \text{err}$, then follows, with the analysis for $E_1$, that $y_1 = f(x_1)$ and that at least two of the functional keys $\text{sk}_i$ and $\text{sk}_j$ are correctly generated and matching the master public keys $\text{mpk}_i$ and $\text{mpk}_j$. But this would also lead, due to perfect correctness of the functional encryption scheme, to a correct decryption of the ciphertext $ct_1'$, which yields $y_2 = f(x_2)$. This shows that the case $y_1 \neq \text{err}$ and $y_2 = \text{err}$ (or $y_1 = \text{err}$ and $y_2 \neq \text{err}$) cannot occur.

By combining the events $E_1$ and $E_2$, we proved that event $E$ cannot occur. To recap, whenever $(\{\text{mpk}'_i\}_{i \in [\lambda]}, \{\text{sk}_i\}_{i \in [\lambda]}, f) \in L$, it is not possible for an adversary $A$ 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 $(\{\text{mpk}'_i\}_{i \in [\lambda]}, \{\text{sk}_i\}_{i \in [\lambda]}, f) \notin L$.

This yields the bound

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

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

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

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 IND-CPA$_0^{\text{FE}}(1^\lambda, \text{A})$ and $G_5$ to the game IND-CPA$_5^{\text{FE}}(1^\lambda, \text{A})$. This results in:

$$\text{Adv}^{\text{IND-CPA}}_{\text{FE'}A}(1^\lambda) = |\text{Win}^{G_0}_A(1^\lambda) - \text{Win}^{G_5}_A(1^\lambda)|.$$

We describe the different games in more detail:

Game $G_0$: This game is the IND-CPA$_0^{\text{FE}}(1^\lambda, \text{A})$ 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 $\text{mpk}_1$ from $x^0$ to $x^1$. The
Putting everything together, we obtain the theorem.

**Game G₂:** In this game, we change the indices that are used in the generation of the NIWI proof in the key generation procedure. 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 $B_1$ such that:

$$|\text{Win}_{G_0}^G(1^\lambda) - \text{Win}_{G_1}^G(1^\lambda)| \leq \text{Adv}_{\text{NIWI}, B_1}^\text{WI}(\lambda).$$

**Game G₃:** In this game, we change the indices that are used in the generation of the master public key $\text{mpk}_2$ from $x^0$ to $x^3$. The transition from $G_2$ to $G_3$ is almost symmetric to the transition from game $G_0$ to $G_1$ except that the ciphertexts under the first master public key $\text{mpk}_1$ contains an encryption of $x^5$ and $x^6$ in addition to $x^3$. As in Lemma 8, the transition is justified by the IND-CPA property of $\text{FE}$. Namely, we can exhibit a PPT adversary $B_0$ such that:

$$|\text{Win}_{G_0}^G(1^\lambda) - \text{Win}_{G_1}^G(1^\lambda)| \leq \text{Adv}_{\text{FE}, B_0}^\text{IND-CPA}(\lambda).$$

**Game G₄:** In this game, we change the indices that are used in the generation of the NIWI proof in the key generation procedure. The transition from $G_3$ to $G_4$ is almost symmetric to the transition from game $G_0$ to $G_1$ except that the ciphertexts under the first two master public keys $\text{mpk}_1$ and $\text{mpk}_2$ contain encryptions of $x^3$ and $x^5$. As in Lemma 8, the transition is justified by the witness-hiding property of NIWI. Namely, we can exhibit a PPT adversary $B_0$ such that:

$$|\text{Win}_{G_0}^G(1^\lambda) - \text{Win}_{G_1}^G(1^\lambda)| \leq \text{Adv}_{\text{NIWI}, B_1}^\text{WI}(\lambda).$$

**Game G₅:** This is the IND-CPA($\text{FE}(1^\lambda)$) 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 $\text{mpk}_1$ and $\text{mpk}_2$ contain encryptions of $x^3$. As in Lemma 7, the transition is justified by the IND-CPA property of $\text{FE}$. Namely, we can exhibit a PPT adversary $B_0$ such that:

$$|\text{Win}_{G_0}^G(1^\lambda) - \text{Win}_{G_1}^G(1^\lambda)| \leq \text{Adv}_{\text{FE}, B_0}^\text{IND-CPA}(\lambda).$$

Putting everything together, we obtain the theorem.

**Lemma 7 (Transition from G₀ to G₁).** For any PPT adversary $A$, there exists a PPT adversary $B₀$ such that:

$$|\text{Win}_{A}^{G₀}(1^\lambda) - \text{Win}_{A}^{G₁}(1^\lambda)| \leq \text{Adv}_{\text{FE}, B₀}^\text{IND-CPA}(\lambda).$$

**Proof.** We build an adversary $B₀$ that simulates $G₁$ towards $A$ when interacting with the underlying IND-CPA experiment.

In the beginning of the reduction, $B₀$ receives $\text{mpk}_₁$ from the experiment. It generates two other functional encryption instances $(\text{mpk}₀, \text{msk}_₀) ← \text{Setup}(1^\lambda)$, for $i ∈ [3] \setminus \{1\}$, sets $\text{mpk}' = \{\text{mpk}_i\}_{i ∈ [3]}$ and gives $\text{mpk}'$ to the adversary. Whenever $A$ asks an encryption query $(x^0, x^1)$, $B₀$ forwards it to its own encryption oracle to receive $ct₁ ← Enc(\text{mpk}_₁, x^0)$, generates $ct₂ ← Enc(\text{mpk}_₂, x^1)$, for $i ∈ [3] \setminus \{1\}$ on its own and sends $ct' = \{ct_i\}_{i ∈ [3]}$ to $A$. For a key generation query $f$, $B₀$ queries its own key encryption oracle on $f$ to receive $sk'₁ ← \text{KeyGen}(\text{mpk}_₁, \text{msk}_₁, f)$, generates $sk'ᵢ ← \text{KeyGen}(\text{mpk}_ᵢ, \text{msk}_ᵢ, f)$ for $i ∈ [3] \setminus \{1\}$ on its own and generates a proof $π ← \text{NIWI.Prove}(1^\lambda, z, w)$ with $z = (\{\text{mpk}_i\}_{i ∈ [3]}, \{sk'ᵢ\}_{i ∈ [3]}, f, w) = ((\text{msk}_i)_{i ∈ [3] \setminus \{1\}}, r_i)_{i ∈ [3] \setminus \{1\}}, s_i)_{i ∈ [3] \setminus \{1\}}$ for the relation $R_{\text{set}}$, by using its information of two-out-of-three different instances. As a reply for the key generation query, $B₀$ sends $sk'_{i} = (\{sk'ᵢ\}_{i ∈ [3]}; π)$. This covers the simulation of the game $G₁$. Finally, $B₀$ outputs the same bit $β$ returned by $A$. It follows, from the perfect simulation, that the advantage of $B₀$ is the same as the advantage $A$.

**Lemma 8 (Transition from G₁ to G₂).** For any PPT adversary $A$, there exists a PPT adversary $B₁$ such that:

$$|\text{Win}_{A}^{G₁}(1^\lambda) - \text{Win}_{A}^{G₂}(1^\lambda)| \leq \text{Adv}_{\text{NIWI}, B₁}^\text{WI}(\lambda).$$

**Proof.** We build an adversary $B₁$ that simulates $G₁ + β$ towards $A$ when interacting with the underlying $\text{WI}_{\text{NIWI}}$ experiment.

In the beginning of the reduction, $B₁$ samples $s_i ← \{0, 1\}^λ$, generates the keys for the three functional encryption instances $(\text{mpk}_i, \text{msk}_i) ← \text{Setup}(1^\lambda, s_i)$, for $i ∈ [3]$, sets $\text{mpk}' = \{\text{mpk}_i\}_{i ∈ [3]}$, saves $(s_i)_{i ∈ [3]}$ and gives $\text{mpk}'$ to the adversary. Whenever $A$ asks an encryption query $(x^0, x^1)$, $B₁$ encrypts $x^1$ using the first public key and $x^0$ using the second and third public key, i.e $ct₁ ← Enc(\text{mpk}_₁, x^1)$ and $ct₂ ← Enc(\text{mpk}_₂, x^0)$, for $i ∈ [3] \setminus \{1\}$. Afterwards, it sends $ct' = \{ct_i\}_{i ∈ [3]}$ to $A$. For a key generation query $f$, $B₁$ samples $r_i ← \{0, 1\}^λ$ and generates $sk'₁ ← \text{KeyGen}(\text{mpk}_₁, \text{msk}_₁, f, r_i)$ for $i ∈ [3]$. Afterwards, $B₁$ submits $(z, w₀, w₁)$ with $z = (\{\text{mpk}_i\}_{i ∈ [3]}, sk'₁, i ∈ [3], f)$, $w₀ = ((\text{msk}_i)_{i ∈ [3] \setminus \{1\}}, r_i)_{i ∈ [3] \setminus \{1\}}, s_i)_{i ∈ [3] \setminus \{1\}}$ and $w₁ = ((\text{msk}_i)_{i ∈ [3] \setminus \{2\}}, r_i)_{i ∈ [3] \setminus \{2\}}, s_i)_{i ∈ [3] \setminus \{2\}}$ as a challenge query to its challenger and receives $π$ as a reply. Finally, $B₁$ sends $(\{sk'₁\}_{i ∈ [3]}; π)$ to $A$ as a reply to the key generation query.

This covers the simulation of the game $G₁ + β$. Finally, $B₁$ outputs the same bit $β$ returned by $A$. It follows, from the perfect simulation, that the advantage of $B₁$ is the same as the advantage $A$. 

□
<table>
<thead>
<tr>
<th>Game</th>
<th>ct</th>
<th>NIWI Witness</th>
<th>justification/remark</th>
</tr>
</thead>
</table>
| G₀   | Enc(mpk₁, x₀) | j₁ = 2  
Enc(mpk₂, x₀) | j₂ = 3  |
| G₁   | Enc(mpk₁, x₁) | j₁ = 2  
Enc(mpk₂, x₀) | j₂ = 3  |
| G₂   | Enc(mpk₁, x₁) | j₁ = 2  
Enc(mpk₂, x₀) | j₂ = 3  |
| G₃   | Enc(mpk₁, x₁) | j₁ = 2  
Enc(mpk₂, x₀) | j₂ = 3  |
| G₄   | Enc(mpk₁, x₁) | j₁ = 2  
Enc(mpk₂, x₀) | j₂ = 3  |
| G₅   | Enc(mpk₁, x₁) | j₁ = 2  
Enc(mpk₂, x₀) | j₂ = 3  |

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 S₁ (for setup generation), that outputs a simulated mpk and an initial (joint) state s = s₀, S₂ for the simulation of functional keys (based on the joint state s which might be update in this process), and finally S₃ 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 secrecy preservation for.

More formally, we require S₁ to output mpk and maintain state s such that (mpk, msk) = Setup(1²; s) holds with probability 1 in Ideal²FE(1²; A, S). Additionally, we require that for any adversary A any functional key skf output by S₂ on input f satisfies skf = KeyGen(mpk, msk, f) with probability 1 in Ideal²FE(1²; A, S) where mpk and msk are the values obtained by Setup(1²; s) where s is the initial private state output by S₁.

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 FE = (Setup, KeyGen, Enc, Dec) be a CFE secure functional encryption scheme with respect to simulators (S₁, S₂, S₃) that satisfy the above condition in Ideal²FE(1³; A, S) (for any adversary A). Let further NIWI = (NIWI.Prove, 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 S₁’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 S₁, S₂, S₃ be the simulators for FE. Then the composite simulator S′ = (S₁′, S₂′, S₃′) works as follows: S₁′ runs S₁ three times to obtain (mpk, sk₁). To answer key-generation queries for functions f, S₂′ runs S₂ three times on the respective joint state (and the function values of all previous queries) to obtain skf,i. Note that by the assumption on the simulation for the underlying scheme, for (mpk₁, msk₁) = Setup(1²; i) we have that skf,i = KeyGen(mpk₁, msk₁, f). By the theorem assumption, KeyGen is deterministic and hence we have all witnesses to simulate a genuine proof π ← NIWI.Prove(1², z, w) with z = {(mpk₁,i) ∈ [3] : (skf,i) ∈ [3]} and w = {(s₁,i) ∈ [3] : (r₁,i) ∈ [3]}. Finally, simulating a ciphertext is done by invoking S₃ 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 FE = (Setup, KeyGen, Enc, Dec) be an IND-CCA secure functional encryption scheme, NIWI = (NIWI.Prove, NIWI.Verify) a NIWI proof system and NIZK = (NIZK.Setup, NIZK.Prove, NIZK.Verify) a NIZK proof system satisfying one-time simulation soundness, then the construction FE’ defined in Figure 25 is IND-CCA secure. Namely, for any PPT adversary A, there exist PPT adversaries B, B’ and B” such that:

\[
\text{Adv}^{\text{IND-CCA}}(\lambda) \leq 2 \cdot \text{Adv}^{\text{NIZK.B’}}(\lambda) + 5 \cdot \text{Adv}^{\text{Sim-Sound}}(\lambda) + 3 \cdot \text{Adv}^{\text{IND-CPA}}(\lambda).
\]

**Proof.** To prove this statement, we use a hybrid argument with the games defined in Fig. 27. Note that G₀ corresponds to the game IND-CCA²FE(1³; A) and G₇ to the game IND-CCA₁FE(1³, A). This results in:

\[
\text{Adv}^{\text{IND-CCA}}(\lambda) = |\text{Win}^G_0(\lambda) - \text{Win}^G_7(\lambda)|.
\]

We describe the different games in more detail:
Setup'(1^\lambda):
CRS ← NIZK.Setup(1^\lambda)

For i ∈ [3]:
(mpk_i, msk_i) ← Setup(1^\lambda; s_i) with s_i ∈ \{0, 1\}^\lambda
Return (mpk', msk')
= (({mpk_i}_{i∈[3]}, CRS), ({msk_i, s_i})_{i∈[3]})

KeyGen'(mpk', msk', f):
Parse mpk' := (({mpk_i}_{i∈[3]}, CRS), msk' := (({msk_i, s_i})_{i∈[3]})

For i ∈ [3]:
sk_{f,i} = KeyGen(mpk_i, msk_i, f; r_i) with r_i ∈ \{0, 1\}^\lambda

Generate π_{sk} ← NIWI.Prove(1^\lambda, z, w) with
z = (({mpk_i}_{i∈[3]}, {sk_{f,i}}_{i∈[3]}, f)
w = (({msk_i}_{i∈[3]}, {r_i}_{i∈[3]}, {s_i}_{i∈[3]})
where L is defined corresponding to R_{set} (Fig. 9)

Return sk'_f = (({sk_{f,i}}_{i∈[3]}, π_{sk})
Enc'(mpk', x):
Parse mpk' := (({mpk_i}_{i∈[3]}, CRS)

For i ∈ [3]:
ct_i ← Enc(mpk_i, x; u_i) with u_i ∈ \{0, 1\}^\lambda

If \exists i ∈ [3] : ct_i = err return err

π_{ct} ← NIWI.Prove(CRS, (mpk_i, ct_i)_{i∈[3]}, (x, {u_i}_{i∈[3]})),
for R_{set} (Fig. 26)

If NIWI.Verify(CRS, (mpk_i, ct_i)_{i∈[3]}, π_{ct}) = 0 return err

Return ct' = (mpk', {ct_i}_{i∈[3]}, π_{ct})
Dec'(mpk', sk'_f, f, ct'):
Parse mpk' := (({mpk_i}_{i∈[3]}, sk'_f := (({sk_{f,i}}_{i∈[3]}, π_{sk}),
ct' := (({ct_i}_{i∈[3]}, π_{ct})

If mpk' = mpk''

If NIWI.Verify(CRS, (mpk_i, ct_i)_{i∈[3]}, π_{ct}) = 1
If NIWI.Verify(1^\lambda, (\{mpk_i\}_{i∈[3]}, \{sk_{f,i}\}_{i∈[3]}), f),
π_{sk}) = 1
y_{f,i} := Dec(mpk_i, sk_{f,i}, f, ct_i), for i ∈ [3]:
If there are indices a, b ∈ [3], a ≠ b s.t.

y_{f,a} = y_{f,b}

Return y ← MajVal(y_{f,1}, y_{f,2}, y_{f,3})

Return ⊥

Fig. 25: Advanced setup consistency compiler. MajVal(·) calculates and returns the majority value of the input values, if there is a clear majority and ⊥ otherwise. Shaded instructions again indicate the difference to the simpler setup compiler.

Relation R_{set}^\text{CCA}:
Instance: z = (({mpk_i}_{i∈[3]}, {ct_i}_{i∈[3]})
Witness: w = (x, {u_i}_{i∈[3]})
R_{set}^\text{CCA}(z, w) = 1 if and only if:
ct_i = Enc(mpk_i, x; u_i) for all i ∈ [3]

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

Game G_0:
This game is the IND-CCA_{\text{A_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 B_0 such that:

|Win_{\text{A_0}}(1^\lambda) − Win_{\text{A_0}}(1^\lambda)| ≤ Adv_{\text{NIZK},B_0}^\text{ZK}(\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 B_1 and B_2 such that:

|Win_{\text{A_1}}(1^\lambda) − Win_{\text{A_1}}(1^\lambda)| ≤ Adv_{\text{NIZK},B_1}^\text{Sim-Sound}(\lambda)
+ Adv_{\text{FE},B_2}^\text{IND-CPA}(\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 B_1 and B_3 such that:

|Win_{\text{A_1}}(1^\lambda) − Win_{\text{A_1}}(1^\lambda)| ≤ Adv_{\text{NIZK},B_1}^\text{Sim-Sound}(\lambda)
+ Adv_{\text{NIWI},B_3}^\text{Wit-Hiding}(\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 B_1 and B_2 such that:
Fig. 27: Overview of the games to prove the IND-CCA security of the advanced setup consistency compiler described in Fig. 25.

<table>
<thead>
<tr>
<th>Game</th>
<th>CRS &amp; π</th>
<th>ct</th>
<th>NIWI Witness</th>
<th>justification/remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>G₀</td>
<td>CRS ← NIZK.Setup(1^λ) π ← NIZK.Prove(CRS, x, w)</td>
<td>Enc(mpk₁, x₀), Enc(mpk₂, x₀), Enc(mpk₃, x₀)</td>
<td>j₁ = 2, j₂ = 3</td>
<td></td>
</tr>
<tr>
<td>G₁</td>
<td>CRS ← S₁(1^λ) π ← S₂(CRS, τ, x)</td>
<td>Enc(mpk₁, x₀), Enc(mpk₂, x₀), Enc(mpk₃, x₀)</td>
<td>j₁ = 2, j₂ = 3</td>
<td>Zero-knowledge of NIZK</td>
</tr>
<tr>
<td>G₂</td>
<td>CRS ← S₁(1^λ) π ← S₂(CRS, τ, x)</td>
<td>Enc(mpk₁, x¹), Enc(mpk₂, x₀), Enc(mpk₃, x₀)</td>
<td>j₁ = 2, j₂ = 3</td>
<td>IND-CPA of FE and one-time simulation-soundness of NIZK</td>
</tr>
<tr>
<td>G₃</td>
<td>CRS ← S₁(1^λ) π ← S₂(CRS, τ, x)</td>
<td>Enc(mpk₁, x¹), Enc(mpk₂, x₀), Enc(mpk₃, x₀)</td>
<td>j₁ = 1, j₂ = 3</td>
<td>Witness-hiding of NIWI and one-time simulation-soundness of NIZK</td>
</tr>
<tr>
<td>G₄</td>
<td>CRS ← S₁(1^λ) π ← S₂(CRS, τ, x)</td>
<td>Enc(mpk₁, x¹), Enc(mpk₂, x¹), Enc(mpk₃, x₀)</td>
<td>j₁ = 1, j₂ = 3</td>
<td>IND-CPA of FE and one-time simulation-soundness of NIZK</td>
</tr>
<tr>
<td>G₅</td>
<td>CRS ← S₁(1^λ) π ← S₂(CRS, τ, x)</td>
<td>Enc(mpk₁, x¹), Enc(mpk₂, x¹), Enc(mpk₃, x₀)</td>
<td>j₁ = 1, j₂ = 2</td>
<td>Witness-hiding of NIWI and one-time simulation-soundness of NIZK</td>
</tr>
<tr>
<td>G₆</td>
<td>CRS ← S₁(1^λ) π ← S₂(CRS, τ, x)</td>
<td>Enc(mpk₁, x¹), Enc(mpk₂, x¹), Enc(mpk₃, x¹)</td>
<td>j₁ = 1, j₂ = 2</td>
<td>IND-CPA of FE and one-time simulation-soundness of NIZK</td>
</tr>
<tr>
<td>G₇</td>
<td>CRS ← NIZK.Setup(1^λ) π ← NIZK.Prove(CRS, x, w)</td>
<td>Enc(mpk₁, x¹), Enc(mpk₂, x¹), Enc(mpk₃, x¹)</td>
<td>j₁ = 1, j₂ = 2</td>
<td>Zero-knowledge of NIZK</td>
</tr>
</tbody>
</table>

| Game G₅: | In this game, we change the indices that are used in the generation of the NIWI proof in the key generation procedure from j₁ = 1 and j₂ = 3 to j₁ = 1 and j₂ = 2. The transition from G₄ to G₅ is almost symmetric to the transition from game G₂ to G₃ except that the ciphertexts under the first two master public keys mpk₁ and mpk₂ contain encryptions of x¹. 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 B₁ and B₃ such that: |

| Game G₆: | In this game, we change the encryption under the third master public key mpk₃ from x₀ to x¹. The transition from G₅ to G₆ is almost symmetric to the transition from game G₁ to G₂ except that the ciphertext under the first two master public keys mpk₁ and mpk₂ contains encryptions of x¹. 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 B₁ and B₂ such that: |

\[
| \text{Win}^{G₅}_{A}(1^{\lambda}) - \text{Win}^{G₄}_{A}(1^{\lambda}) | \leq \text{Adv}^{\text{Sim-Sound}}_{\text{NIWI}, B₁}(\lambda) + \text{Adv}^{\text{IND-CPA}}_{\text{FE}, B₂}(\lambda) .
\]

\[
| \text{Win}^{G₆}_{A}(1^{\lambda}) - \text{Win}^{G₅}_{A}(1^{\lambda}) | \leq \text{Adv}^{\text{Sim-Sound}}_{\text{NIWI}, B₁}(\lambda) + \text{Adv}^{\text{WI}}_{\text{NIWI}, B₂}(\lambda) .
\]
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 $A$ participating in $G_k$ for $k \in \{1, \ldots, 6\}$, the probability that, during the experiment, $A$ 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 $B_1$, such that

$$\Pr \left[ \exists (f, \{ct'_i\}_{i \in [8]}, \pi') \in Q : \\
\{\{ct'_i\}_{i \in [8]}, \pi' \neq (\{ct'_i\}_{i \in [8]}, \pi'), \\
\text{NIZK.Verify}(\text{CRS}, \{ct'_i\}_{i \in [8]}, \pi') = 1 \\
\text{for all } i, j \in [3], i \neq j : \\
\text{Dec}(\text{mpk}_i, \text{sk}_{f,j}, ct'_j) \neq \text{Dec}(\text{mpk}_j, \text{sk}_{f,j}, ct'_j) \\
\right] \\
\leq \text{Adv}^{\text{Sim-Sound}}(\Lambda, \lambda).$$

where $\text{sk}_{f,i} \leftarrow \text{KeyGen}(\text{mpk}_i, \text{msk}_i, f)$ for $i \in [3]$, $\{\{ct'_i\}_{i \in [8]}, \pi'\}$ is the reply to the encryption query $(x^0, x^1)$ made by $A$ where $\{\{ct_i\}_{i \in [8]}, \pi\}$ is the reply to the encryption query $(x^0, x^1)$ made by $A$ and $Q$ the list containing all the decryption queries $(f, \{ct'_i\}_{i \in [8]}, \pi')$ asked by $A$, knowing the master public key $\text{mpk}^k := (\text{CRS}, \{\text{mpk}_i\}_{i \in [2]})$, the reply to its challenge query $\{ct_i\}_{i \in [8]}, \pi$ and by having access to the key generation oracle $\text{KeyGen}'(\text{mpk}_i, \text{msk}_i, \cdot)$, during the game.

Proof. We build an adversary $B_1$ that simulates $G_6$ towards $A$ when interacting with the underlying one-time simulation-soundness experiment.

After the adversary $B_1$ has received CRS from the underlying experiment, it generates $(\text{mpk}_k, \text{msk}_k) \leftarrow \text{Setup}(1^\lambda; s_l)$ with $s_l \leftarrow \{0,1\}^\lambda$ for $i \in [3]$ and sends $\text{mpk}' := (\text{CRS}, \{\text{mpk}_i\}_{i \in [3]})$ and gives $\text{mpk}'$ to the adversary.

Whenever $A$ asks an encryption query $(x^0, x^1)$, $B_0$ generates the ciphertext $ct_i' = (\text{Enc}(\text{mpk}_i, x^0, u_i), u_i)_{i \in [8]}$ with $u_i \leftarrow \{0,1\}^\lambda$ for $i \in [3]$ and sends $y = (\text{mpk}_i, ct_i')_{i \in [8]}$ to $B$. As in Lemma 9, the transition is justified by the zero-knowledge property of NIZK. Namely, we can exhibit a PPT adversary $B_0$ such that:

$$|\text{Win}^{G_0}_A(1^\lambda) - \text{Win}^{G_0}_A(1^\lambda)| \leq \text{Adv}^{\text{Sim-Sound}}(\Lambda, \lambda).$$

Putting everything together, we obtain the theorem.

Lemma 9 (Transition from $G_0$ to $G_1$). For any PPT adversary $A$, there exists a PPT adversary $B_0$ such that:

$$|\text{Win}^{G_0}_A(1^\lambda) - \text{Win}^{G_0}_A(1^\lambda)| \leq \text{Adv}^{\text{Sim-Sound}}(\Lambda, \lambda).$$

Proof. We build an adversary $B_0$ that simulates $G_8$ towards $A$ when interacting with the underlying ZK$^N_{\text{NIZK}}$ experiment.

In the beginning of the reduction, $B_0$ receives CRS from the ZK$^N_{\text{NIZK}}$ experiment. It generates three functional encryption instances $(\text{mpk}_i, \text{msk}_i) \leftarrow \text{Setup}(1^\lambda; s_i)$ with $s_i \leftarrow \{0,1\}^\lambda$ for $i \in [3]$, sets $\text{mpk}' := (\text{CRS}, \{\text{mpk}_i\}_{i \in [3]})$ and gives $\text{mpk}'$ to the adversary.

Whenever $A$ asks an encryption query $(x^0, x^1)$, $B_0$ generates the ciphertext $ct_i' = (\text{Enc}(\text{mpk}_i, x^0, u_i), u_i)_{i \in [8]}$ with $u_i \leftarrow \{0,1\}^\lambda$ for $i \in [3]$ and sends $y = (\text{mpk}_i, ct_i')_{i \in [8]}$ and $w = (x, \{u_i\}_{i \in [8]})$ as a statement-witness pair to its challenger. As an answer, $B_0$ receives a proof $\pi_{sk}$ for the relation $\text{R}_{\text{NIZK}}$. It sets $ct_i' = (\{ct_i\}_{i \in [8]}, \pi_{ct})$ and sends it to $A$.

For a key generation query $f$, $B_0$ generates $sk_{f,i} \leftarrow \text{KeyGen}(\text{mpk}_i, \text{msk}_i, f, r_i)$ with $r_i \leftarrow \{0,1\}^\lambda$ for $i \in [3]$ and creates a NIWI proof $\pi_{sk}$ over the relation $\text{R}_{\text{NIZK}}$ for the statement-witness pair $y = (\{\text{mpk}_i\}_{i \in [8]}, \{\text{sk}_{f,i}\}_{i \in [8]}, f)$ and $w = (\{\text{msk}_i\}_{i \in [8]}, \{r_i\}_{i \in [8]}, \{s_i\}_{i \in [8]})$. Then, $B_0$ sends $sk_f' = (\{sk_{f,i}\}_{i \in [8]}, \pi_{sk})$ as a reply to $A$.

Whenever $A$ submits a decryption query $(f, ct' = (\{ct_i\}_{i \in [2]}, \pi_{ct}))$, $B_0$ generates the functional keys $sk_{f,i} \leftarrow \text{KeyGen}(\text{mpk}_i, \text{msk}_i, f)$ for $i \in [3]$ and executes NIZK.Verify(CRS, $(\text{mpk}_i, \text{ct}_i)_{i \in [3]}$). If NIZK.Verify(CRS, $(\text{mpk}_i, \text{ct}_i)_{i \in [3]}$) = 1, $B_0$ computes $y_{f,i} := \text{Dec}(\text{mpk}_i, f, \text{sk}_{f,i}, ct_i)$ for $i \in [3]$ and sends the majority vote, $y \leftarrow \text{MajVal}(y_{f,1}, y_{f,2}, y_{f,3})$, to $A$. If the verification outputs 0, $B_0$ sends ⊥ to $A$.

This covers the simulation of the game $G_6$. Finally $B_0$ outputs the same bit $\beta'$ returned by $A$. It follows, from the perfect simulation, that the advantage of $B_0$ is the same as the advantage of $A$.
Lemma 11 (Transition from G₂ to G₃). For any PPT adversary A, there exist PPT adversaries B₁ and B₂, such that
\[
|\text{Win}_A^{G_2}(1^λ) - \text{Win}_A^{G_2}(1^λ)| \leq \text{Adv}^{\text{Sim-Sound}}_{\text{NIWI},B_1}(λ) + \text{Adv}^{\text{IND-CPA}}_{\text{Enc}_B}(λ) + \text{Adv}^{\text{NIWI},B_2}(λ).
\]

Proof. We build an adversary B₂ that simulates G₂⁺β towards A when interacting with the underlying WIβₙᵢwi experiment.

In the beginning of the reduction, B₂ receives mpk₁ from the underlying experiment. It simulates a CRₕ, i.e. (CRₚ, τ) ← S₁(1^λ), generates several functional encryption instances (mpkᵢ, mski) ← Setup(1^λ; si) with si ∈ {0, 1}^λ for i ∈ [3] \ {1}, sets mpk' := (CRₚ, {mpkᵢ}, i ∈ [3]) and sends mpk' to A. Whenever A submits a key generation query f, B₂ forwards this query to its own key generation oracle KeyGen(mpkᵢ, mskiᵢ), to receive skᵢ,1 as an answer. Then, B₂ generates skᵢ,f₁ ← KeyGen(mpkᵢ, mskiᵢ, f, rᵢ) for i ∈ [3] \ {1} by itself and creates a NIWI proof πᵢ₁ over the relation Rᵢ₁ for the statement-witness pair y = ((mskiᵢ)₁∈[3], {skᵢ,f₁}₁∈[3], f) and w = ((mskiᵢ)₁∈[3], {rᵢ}₁∈[3], {sᵢ}₁∈[3]). Then, B₂ sends skᵢ,f₁ as a reply to A.

For the challenge query (x₀, x¹) asked by A, B₂ forwards it to its own encryption oracle and receives ct₁ = Enc(mpk₁, x₀) as an answer. It generates ct₀ = Enc(mpk₁, x¹; u₁) with u₁ ← {0, 1}^λ, for i ∈ [3] \ {1}, simulates a valid proof π of the relation Rᵢ₀ using the statement y = (mpk₁, ct₀)₁∈[3], i.e. πᵢ₀ ← S₂(CRS, τ, y) and sends ct'₀ := ((ct₀)₁∈[3], πᵢ₀) to A.

Whenever A asks a decryption query (f, ct') := ((ct')₁∈[3], πᵢ₀), B₂ first verifies the proof πᵢ₀, i.e. it executes NIWI.Verify(CRS, (mpk₁, ct₀)₁∈[3], πᵢ₀). If the verification outputs 1, B₂ generates skᵢ,f₁ ← KeyGen(mpki, mskiᵢ, f) for i ∈ [3] \ {1}, computes yᵢ,f₁ ← Dec(mpki, f, skᵢ,f₁, ct₀) for i ∈ [3] \ {1} and y ← MajVal(yᵢ,f₁)₁∈[3\{1}]. If the claim fails, A halts. B₂ then proceeds as follows. If the claim succeeds, B₂ generates skᵢ,f₂₁ ← KeyGen(mpki, mskiᵢ, f, rᵢ) for i ∈ [3] \ {1}, computes yᵢ,f₂₁ ← Dec(mpki, f, skᵢ,f₂₁, ct₀) for i ∈ [3] \ {1} and sends y ← MajVal(yᵢ,f₂₁)₁∈[3\{1}]. If the verification outputs 0, B₂ sends ⊥ to A.

This covers the simulation of the game G₁⁺β. Finally, B₂ outputs the same bit β' returned by A. Together with the analysis of adversary B₁, this yields the advantage mentioned in the lemma.

Lemma 10 (Transition from G₁ to G₂). For any PPT adversary A, there exist PPT adversaries B₁ and B₂, such that
\[
|\text{Win}_A^{G_2}(1^λ) - \text{Win}_A^{G_2}(1^λ)| \leq \text{Adv}^{\text{Sim-Sound}}_{\text{NIWI},B_1}(λ) + \text{Adv}^{\text{IND-CPA}}_{\text{Enc}_B}(λ) + \text{Adv}^{\text{NIWI},B_2}(λ).
\]

Proof. We build an adversary B₂ that simulates G₂⁺β towards A when interacting with the underlying WIβₙᵢwi experiment.

In the beginning of the reduction, B₂ simulates a CRₚ, i.e. (CRₚ, τ) ← S₁(1^λ), generates several functional encryption instances (mpkᵢ, mski) ← Setup(1^λ; si) with si ∈ {0, 1}^λ for i ∈ [3], sets mpk' := (CRₚ, {mpkᵢ}, i ∈ [3]) and sends mpk' to A.

Whenever A asks an encryption query (x₀, x¹), B₂ encrypts x₀ using the first public key and x¹ using the second and third public key, i.e. ct₁ ← Enc(mpki, x₀; u₁) and ctᵢ ← Enc(mpki, xᵢ; uᵢ), for i ∈ [3] \ {1}, where uᵢ ← {0, 1}^λ. Afterwards, B₂ simulates a valid proof π of the relation Rₚ using the statement y = (mpki, ctᵢ)₁∈[3], i.e. πᵢ ← S₂(CRS, τ, y) and sends ct' := ((ctᵢ)₁∈[3], πᵢ) to A.

For a key generation query f, B₂ samples rᵢ ← {0, 1}^λ and generates skᵢ,fᵢ ← KeyGen(mpki, mskiᵢ, f, rᵢ) for i ∈ [3]. Afterwards, B₂ submits (z, v₀, w₀) with z = ((mpki)₁∈[3], {skᵢ,fᵢ}₁∈[3], f), u₀ = ((mskiᵢ)₁∈[3], {rᵢ}₁∈[3], {sᵢ}₁∈[3]) and w₀ = ((mskiᵢ)₁∈[3\{2}, {rᵢ}₁∈[3\{2}, {sᵢ}₁∈[3\{2}) as a challenge query to its challenger and receives τ as a reply. Finally, B₂ sends ((skᵢ,fᵢ)₁∈[3], τ) to A as a reply to the key generation query.

Whenever A asks a decryption query (f, ct') := ((ct')₁∈[3], πᵢ₀), B₂ first verifies the proof πᵢ₀, i.e. it executes NIWI.Verify(CRS, (mpki, ct₀)₁∈[3], πᵢ₀). If the verification outputs 1, B₂ generates skᵢ,f₁ ← KeyGen(mpki, mskiᵢ, f) for i ∈ [3], computes yᵢ,f₁ ← Dec(mpki, f, skᵢ,f₁, ct₀) for i ∈ [3] \ {1} and sends y ← MajVal(yᵢ,f₁)₁∈[3\{1} to A. If the verification outputs 0, B₂ sends ⊥ to A.

This covers the simulation of the game G₂⁺β. Finally, B₂ outputs the same bit β' returned by A. Together with the analysis of adversary B₁, 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 FE = (Setup, KeyGen, Enc, Dec) be a functional encryption scheme, NIWI = (NIWI.Prove, NIWI.Verify) a NIWI proof system for Rₚ (Fig. 9) and NIZK = (NIZK.Setup, NIZK.Prove, NIZK.Verify) a NIZK proof system for Rᶜₚ (Fig. 26), then the construction FE' = (Setup', KeyGen', Enc', Dec') defined in Fig. 25 is setup consistent. Namely, for any PPT adversary A, exists a PPT adversary B such that
\[
|\text{Pr}[\text{set-CONS}_{\text{FE'}}(1^λ, A) = 1]| \leq \text{Adv}^{\text{Sound}}_{\text{NIWI},B}(λ).
\]
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 $\text{Dec}'$, as defined in the compiler. Hence, we can invoke the same analysis, based on the invariant that for each ciphertext $ct'_i = (ct_i, \pi_i)$ ($i = 1, 2$), $\pi_i$ 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).

<table>
<thead>
<tr>
<th>Functionality $\text{Func}^S_{\text{auth}}_{R_1, \ldots, R_n}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>The functionality is parameterized by sender (extended) identity $S$ and receiver (extended) identities $R_i$. The functionality initializes an empty array $M$.</td>
</tr>
<tr>
<td>• On input (SEND, sid, m) from $S$, store $M \leftarrow M</td>
</tr>
<tr>
<td>• On input (GETMSGS, sid) from the adversary (on the backdoor tape) output $M$ to the adversary.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Functionality $\text{Func}^S_{\text{sec}}_{R}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>The functionality is parameterized by sender $S$ and receiver $R$. It initializes an empty array $M$.</td>
</tr>
<tr>
<td>• On input (SEND, sid, m) from $S$, store $M \leftarrow M</td>
</tr>
<tr>
<td>• On input (GETMSGS, sid) from the adversary (on the backdoor tape) output $M$ to the adversary.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functionality $\text{Func}^A_{\text{basic-Rep}, C}$</th>
</tr>
</thead>
</table>
| The functionality is parameterized by a set $C \subseteq \{0, 1\}^*$ and the party identifiers $A, B_i, i \in [t]$, it interacts with. It maintains a lookup table $M_i$, which is initially empty.

| a) Input. Upon receiving (WRITE, sid, x) from a party with party-id $A$ do: If $c \in C$, then compute handle $h \leftarrow \text{getHandle}$ and store $M[h] \leftarrow c$. Return $h$ to the calling party. |

| b) Output. Upon receiving (READ, sid, h) from a party with party-id $B_i$, $i \in [t]$, do: If $M[h] = 0$ then return noData. Otherwise, i.e., if $M[h] \in C$, return $M[h]$ to the calling party. |

B. The FE protocol

The UC protocol $\pi^A_{\text{FE}}$ (based on a functional encryption scheme FE) to realize $\text{Func}^A_{\text{Rep}, (F^+, f_0)}$ 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 $\text{Func}^A_{\text{Rep}, (F^+, f_0)}$ and only then prove the necessary direction. Thereafter, we turn our attention to $\text{Func}^A_{\text{Rep}, (F^+, f_0, f)}$.

Consistency implies the UC realization: We first describe the simulator $S$ for the dummy UC adversary $D$, which basically means that $S$ receives the instructions by the environment $Z$. 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 (SETUP, sid) from $\text{Func}^A_{\text{Rep}, (F^+, f_0)}$, the simulator $S$ executes $(\text{msk}, \text{mpk}) \leftarrow \text{Setup}()$ to obtain the master public and private key and provides $\text{mpk}$ to the environment as the message received by the dishonest input provider and decryptors (or leaked by the authenticated broadcast channel). Upon receiving (ASSIGNED, sid, f, i) from $\text{Func}^A_{\text{Rep}, (F^+, f_0)}$ then $S$ computes $F_0 \leftarrow F_0 \cup \{f\}$ (where $F_0$ is initially empty) and evaluates $\text{sk}_f \leftarrow \text{KeyGen}(\text{mpk}, \text{msk}, f)$ and provides $\text{sk}_f$ to the environment when asked (READ, sid) (meant for the secure channel between the setup generator and some corrupted decryptor that receives the functional key for $f$). Finally, update $F \leftarrow F \cup \{(\text{sk}_f, f)\}$ (again $F$ is initially empty).

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

Upon receiving (READ, sid, h, f) from $\text{Func}^A_{\text{Rep}, (F^+, f_0)}$, recall the ciphertext $ct$ associated to $h$ (previously input by the corrupted input provider). Then compute $y \leftarrow \text{Dec}(\text{mpk}, f, \text{sk}_f, ct)$ and provide the input (READ, sid, h, f, (unknown, y)) to $\text{Func}^A_{\text{Rep}, (F^+, f_0)}$.

If any receiver is corrupted, the adversary can instruct them to input (READ, sid, h) to directly obtain the cipher-
Upon each invocation, protocol $\pi_{FE}^{A,B,C,t}$ first verifies that this ITIs party identifier matches $\text{pid} \in \{A, B, C\}$ (for some $i \in [t]$) and rejects the message otherwise. This means that the ITI running the protocol must have the extended identity $\text{eid}_{\text{pid}} = (\pi_{FE}^{A,B,C,t}, \text{sid} | \text{pid})$ for some $\text{sid}$ and $\text{pid} \in \{A, B, C\}$ (for some $i \in [t]$).

Depending on the encoded $\text{pid}$, match the input to the following commands:

$\text{pid} = A$:

- On input $(\text{SETUP}, \text{sid})$, execute $(\text{mpk}, \text{msk}) \leftarrow \text{Setup}()$, store the pair internally and send $\text{mpk}$ via $\text{Func}_\text{auth}^{\text{eid, eid}_{1, \ldots, \text{sid}}, \text{sid}}$ to the input provider and via $\text{Func}_\text{auth}^{\text{eid, eid}_{1, \ldots, \text{sid}}, \text{sid}}$ to the decryptor.

- On input $(\text{ASSIGN}, \text{sid}, f, i)$, ensure that $f \in F$ and otherwise ignore the input. Execute $sk_f = \text{KeyGen}(\text{mpk}, \text{msk}, f)$, send $sk_f$ via $\text{Func}_\text{sec}^{\text{eid, eid}_{1, \ldots, \text{sid}}, \text{sid}}$ to the decryptor.

$\text{pid} = B$:

- On input $(\text{WRITE}, \text{sid}, x)$, ensure that $x \in \mathcal{X}$ and that an $\text{mpk}$ has been received (otherwise ignore the input). Then, execute $ct \leftarrow \text{Enc}(\text{mpk}, x)$ and if $ct \neq \text{err}$ output $(\text{WRITE}, \text{sid}, ct)$ to $\text{Func}_{\text{basic-rep}, \mathcal{C}}$ and return the obtained handle $h$ from the basic repository back to the caller by returning $(\text{WRITTEN}, \text{sid}, h)$. (Give up activation if an error occurs).

- On receiving the master public key $\text{mpk}$ from $\text{Func}_\text{auth}^{\text{eid, eid}_{1, \ldots, \text{sid}}, \text{sid}}$ and if this is the first time the key is delivered, store it internally for future reference. Ignore any future message from the channel.

- On receiving a pair $(\text{sk}, f)$ from $\text{Func}_\text{auth}^{\text{eid, eid}_{1, \ldots, \text{sid}}, \text{sid}}$ and if $f \in F$, then store the pair $(\text{sk}, f)$ 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 $Z$ (and dummy adversary) with non-negligible advantage in distinguishing the real and ideal worlds (w.r.t. simulator $S$ above) when only corrupting party $A$ (and possibly a subset of the receivers), we give a reduction $\rho_1$ to construct an adversary $A := \rho_1(Z)$ 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 $Z$ (with the protocol and dummy adversary $D$), there is a handle $h$ such that the set of associated output values $y_i(h)$ obtained by party $B$ on queries $(\text{READ}, h, f_i)$ are such that $\{x' \in \mathcal{M} | \forall i : f_i(x') = y_i(h)\} = \{\}$. As long as $-E_1$ holds in the execution, the simulator $S$ executes the real-world view perfectly, since the instruction marked with $(***)$ must never be executed in the ideal world and any value $y$ returned to an honest decryptor is explainable by an element $x \in \mathcal{X} \cup \{\bot\}$ that fulfills $f(x) = y$ for the queried function $f \in F$ (and thus also $\bot$ is never observed).

Hence, let $A := \rho_1(Z)$ be the consistency adversary that internally runs $Z$ and emulates the real-world view towards $Z$, i.e., upon any request by $Z$, $\rho_1$ emulates the actions of the protocol when generating its replies to $Z$. 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, $\rho_1$ identifies the handle $h$ that caused the event and outputs the associated ciphertext $ct_h$ stored for handle $h$. Note that $A$ is efficiently implementable by the assumption of (the efficiently implementable function) $\text{preMap}()$ which can be used to detect $E_1$ (and further events in the other cases).\footnote{We note that by picking one ciphertext at random, $\rho_1$ could avoid the dependence on $\text{preMap}()$ 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$^{\text{FE}}(1^\lambda, \mathcal{A})$ 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 $\mathcal{A}$ must define a valid base value $x \in \mathcal{X}$ and thus, upon read queries by an honest receiver/decryptor $\mathcal{B}_1$ only $2, (a)$ in the description of $\text{Func}_{\text{Rep}, (\mathcal{F}^\mathcal{A}, f_0)}$ is triggered. The main task of simulator $\mathcal{S}$ in this simulation is to translate dishonest actions of party $\mathcal{C}$ when sending values towards the other two parties and controlling the influence on Alice actions. Thus, upon receiving $(\text{SEND}, \text{sid}, m)$ for the channel from $\mathcal{C}$ to $\mathcal{A}$, $\mathcal{S}$ just remembers that $\text{mpk}_1 \leftarrow m$ (and similarly for the message sent to the receivers which is defined to be $\text{mpk}_2$). As soon as $\text{mpk}_1$ is defined, $\mathcal{C}$ sends (setup, sid) to the ideal functionality (allowing the sender to input values).

As soon as both values $\text{mpk}_1$ and $\text{mpk}_2$ 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 $(\text{SEND}, \text{sid}, m)$ for the channel from $\mathcal{C}$ to some honest receiver $\mathcal{B}_1$, the simulator first parses $m$ as $(\text{sk}, f)$ and if $x \not\in \mathcal{F}$, it gives up activation. Otherwise, it encrypts a fixed message $ct \leftarrow \text{Enc}(\text{mpk}_1, m)$ and performs a trial decryption $y \leftarrow \text{Dec}(\text{mpk}_2, f, \text{sk}, ct)$. If $y = \circ$ then give up activation. Otherwise, output $(\text{ASSIGN}, \text{sid}, f, i)$ to $\text{Func}_{\text{Rep}, (\mathcal{F}^\mathcal{A}, f_0)}$ 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 $\text{Func}_{\text{Rep}, (\mathcal{F}^\mathcal{A}, f_0)}$ with a public delayed output $(\text{WRITE}, \text{sid}, x)$ (in response to Alice’s input), the simulator performs a trial encryption $ct \leftarrow \text{Enc}(\text{mpk}_1, x)$ and sends ACK for this operation if and only if $ct \neq \text{err}$. Otherwise, the simulator activates the environment as next entity.

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 $\mathcal{C}$ at least one decryptor is dishonest: hence to simulate the ciphertext for handle $h$ the simulator first obtains the message $x$ via the command (reveal, 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 $\mathcal{C}$ (possibly alongside a subset of receivers), we give reductions $\rho_2$ and $\rho_3$ 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 $\pi'$, which is defined as $\pi_{\text{FE}}^{\mathcal{A}, \mathcal{B}, \mathcal{C}, t}$ but where party $\mathcal{A}$ provides its received master-public key $\text{mpk}_1$ to parties $\mathcal{B}_i$ via an additional covert broadcast channel and where parties $\mathcal{B}_i$, already upon receiving a functional key $(\text{sk}, f)$, perform a trial decryption and rejects the key if $\text{Dec}(\text{mpk}_2, f, \text{sk}, \text{Enc}(\text{mpk}_1, m)) = \circ$, where $\text{mpk}_2$ is the master public key sent from part $\mathcal{C}$ to parties $\mathcal{B}_i$. We observe that protocol $\pi'$ and $\pi_{\text{FE}}^{\mathcal{A}, \mathcal{B}, \mathcal{C}, t}$ have equivalent behaviors as long as the environment is not able to provide an input (WRITE, sid, $x$) to party $\mathcal{A}$ that provokes event $E_2$ defined by the condition that $\text{Dec}(\text{mpk}_2, f, \text{sk}, \text{Enc}(\text{mpk}_1, m)) = \circ$ but $\text{Dec}(\text{mpk}_2, f, \text{sk}, \text{Enc}(\text{mpk}_1, x)) \neq \circ$ (of course within an honest receiver/decryptor and where $f$ and $\text{sk}$ have been received together from party $\mathcal{C}$). In case $E_3$, the function $f$ is never evaluated upon input (READ, $h$, $f$) for any $h$ by protocol $\pi'$, whereas in $\pi_{\text{FE}}^{\mathcal{A}, \mathcal{B}, \mathcal{C}, t}$ it might. Hence, let $\mathcal{A} := \rho_2(\mathcal{Z})$ be the adversary for the setup consistency game defined as follows: $\rho_2$ internally runs $\mathcal{Z}$ and emulates the execution of protocol $\pi_{\text{FE}}^{\mathcal{A}, \mathcal{B}, \mathcal{C}, t}$ towards $\mathcal{Z}$ (by monolithically executing all required protocol steps) until event $E_3$ is observed. In this case, $\rho_2$ outputs $(\text{mpk}_1, \text{mpk}_2, \text{sk}, x, m)$, where $x$ and $\text{sk}$ are the values fulfilling the condition of event $E_3$. $\rho_2$ wins set-CONS$^{\text{FE}}(1^\lambda, \mathcal{A})$ 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 $\pi'$, $\mathcal{Z}$ provokes for some handle $h$ that a query (READ, sid, $h$) to party $\mathcal{B}_i$ following a write instruction (WRITE, sid, $x$) to party $\mathcal{A}$ that returned this handle $h$, yields an output value $y \neq f(x)$. Again, the simulation $\mathcal{S}$ interacting with the repository $\text{Func}_{\text{Rep}, (\mathcal{F}^\mathcal{A}, f_0)}$ is a perfect simulation of $\pi'$: 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 $(***)$ 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 $\mathcal{A}$ and $\mathcal{B}_i$. This includes the receipt of two message $\text{mpk}_1$ and $\text{mpk}_2$ for parties $\mathcal{A}$ and $\mathcal{B}_i$, respectively. The reduction $\rho_3$, once it detects event $E_4$ is provoked, can output $(\text{mpk}_1, \text{mpk}_2, \text{sk}, f, x, m)$, where $(\text{sk}, f)$ is defined as the key function pair provoked event $E_4$. Hence, $\mathcal{A}$ achieves set-CONS$^{\text{FE}}(1^\lambda, \mathcal{A}) = 1$ 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
generated setup parameters. That is, it defines $\text{mpk}$ as the claimed master public key that party $C$ sends to an honest party $B_i$. For the other messages ($\text{SEND}, \text{sid}, m$) for the channel from $C$ to $B_j$, the simulator again parses it as a key-function pair ($sk, f$) and does the validity tests as above and in case $f \in F$ and the trial decryption $\text{Dec}(\text{mpk}, f, sk, \text{Enc}(\text{mpk}, \tilde{m}))$ (with respect to one master public key) does not yield $\diamond$, then output $(\text{ASSIGN}, \text{sid}, f, i)$ to $\text{Func}_{\text{Rep}, (F^+, f_0)}^{A, B, C, t}$. Simulating a dishonest receiver is straightforward.

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

Finally, upon receiving $(\text{READ}, \text{sid}, h, f)$ from $\text{Func}_{\text{Rep}, (F^+, f_0)}^{A, B, C, t}$ (upon a reading instruction by honest party $B_j$), obtain the ciphertext $\text{ct}$ associated to $h$, compute $y \leftarrow \text{Dec}(\text{mpk}, f, sk_f, \text{ct})$ and $x \leftarrow \text{unknown}$ and provide the input $(\text{READ}, \text{sid}, h, f, (x, y))$ to $\text{Func}_{\text{Rep}, (F^+, f_0)}^{A, B, C, t}$.

**Lemma 14.** For any environment $Z$ (and dummy adversary) with non-negligible advantage in distinguishing the real and ideal worlds (w.r.t. simulator $S$ above) when only corrupting parties $A$ and $C$ (and possibly alongside a subset of receivers), we give reductions $\rho_4$ and $\rho_5$ to construct adversaries $A_i := \rho_i(Z)$ such that at least one of the $A_i$ violates strong input-consistency with non-negligible probability.

**Proof.** We again make a first hybrid step and consider the protocol $\pi''$, where each party $B_i$, upon receiving a functional key $sk$ together with its claimed function $f$, performs a trial decryption $\text{Dec}(\text{mpk}, f, sk, \text{ct})$, where $\text{ct} \leftarrow \text{Enc}(\text{mpk}, \tilde{m})$ (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 $\text{ct}$ and a secret key ($sk, f$) to some honest party $B_i$ such that $\text{Dec}(\text{mpk}, f, sk, \text{ct}) \neq \diamond$ and $\text{Dec}(\text{mpk}, f, sk, \text{ct}) = \diamond$. Again, $\pi_{\text{FE}}''$ and $\pi''$ have an identical behavior for any honest party $B_i$ until event $E_5$ is triggered. As above, this yields a reduction $A := \rho_4(Z)$, which emulates party $B$'s actions towards $Z$ and if $Z$ provokes $E_5$, it outputs $(\text{mpk}, \text{ct}, c, \{sk\})$, where the triple $\text{ct}, c, \text{sk}$ are the values provided by $Z$ that trigger event $E_5$.

The final reduction is obtained by defining, for an execution of $Z$ with $\pi''$ the event $E_6$ (analogous to $E_1$ above): Let $E_6$ denote the event that at any point in the execution of $Z$ (with the protocol and dummy adversary $D$), there is a handle $h$ such that the set of associated output values $y_i^{(h)}$ obtained by some honest receiver/decryptor $B_i$ on queries $(\text{READ}, h, f_i)$ are such that $\{x' \in M \mid \forall i : f_i(x') = y_i^{(h)}\} = \{\}$. As long as $E_6$ does not occur, the outputs generated by any honest party $B_i$ 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 $A := \rho_5(Z)$ for st-in-CONS$^E(1^\lambda, \mathcal{A})$ is now designed analogous to $\rho_4$: Here, $\rho_5$ only emulates the honest receivers’ actions towards an environment. When it detects event $E_6$, it outputs $(\text{mpk}, \text{ct}, c, \text{ct}, F)$ where $\text{ct}$ 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$^E(1^\lambda, \mathcal{A})$ (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 $S_1, S_2,$ and $S_3$ 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 $(\text{mpk}, s) \leftarrow S_1()$. Upon receiving $(\text{ASSIGNED}, \text{sid}, f, j)$ (for some index $i$) from $\text{Func}_{\text{Rep}, (F^+, f_0)}^{A, B, C, t}$ then $S$ computes computes $F_0 \leftarrow F_0 \cup \{f\}$ and outputs $(\text{READ}, h_i, f)$ for each available label $h_1, \ldots, h_n$ in $\text{Func}_{\text{Rep}, (F^+, f_0)}^{A, B, C, t}$ to obtain the associated value $y_i$, and execute $\text{sk}_f \leftarrow S_2(f, y_1, \ldots, y_n)[[s]]$. This key is then output whenever the simulator must simulate the transmission of this key towards a dishonest receiver.

On input $(\text{READ}, \text{sid}, h)$ 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 $\text{Func}_{\text{Rep}, (F^+, f_0)}^{A, B, C, t}$) and a ciphertext $\text{ct}_h$ has already been simulated, then return $\text{ct}_h$ to the adversary. Otherwise, the ciphertext for this handle is simulated as follows: for all already assigned functions $f_i \in F_0$, ask $(\text{READ}, h, f_i)$ to $\text{Func}_{\text{Rep}, (F^+, f_0)}^{A, B, C, t}$ 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 $\text{ct}_h \leftarrow S_3(y_0, y_1, \ldots, y_k)[[s]]$ 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 \(A_1, A_2, A_3\) such that \(Pr[\text{in-CONS}_{FE}(1^\lambda, A_1)], Pr[\text{set-CONS}_{FE}(1^\lambda, A_2)],\) or \(Pr[\text{st-in-CONS}_{FE}(1^\lambda, A_3)]\) is non-negligible, then we can distinguish a real execution of \(\pi^{AB, \text{C,F}}_{\text{Rep},(F^+, f_0)}\) from any ideal-world execution for \(\text{Func}_{\text{Rep},(F^+, f_0)}\) and an arbitrary simulator \(S'\). From the assumed adversaries we construct environments \(Z_1\) and \(Z_2\) that distinguish the real and ideal worlds, the former for the case when only \(A\) is corrupted, and for the latter when only party \(C\) is corrupted. For the sake of the argument, we only have to consider the simple setting with one honest receiver/decryptor \(B_1\) and one corrupted receiver \(B_2\).

Case in-CONS: \(Z_1\) internally runs \(A_1\) 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 \(C\) to \(B_2\) since \(B_2\) is corrupted. When \(A_1\) outputs a ciphertext \(ct, Z_1\) does the following: it corrupts party \(A\) and instructs it (all via the dummy adversary) to issue the write instruction \((\text{WRITE}, \text{sid}, ct)\) (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 \(B_1\) to obtain \(n\) values \(y_i, 1 \leq i \leq n\). If \(\{x' \in M | \forall i : f_i(x') = y_i\} = \{\}\) then \(Z_1\) outputs 1 and otherwise it outputs 0. It is clear that \(Z_1\) never outputs 1 when interacting with any simulator and functionality \(\text{Func}^{AB, \text{C,F}}_{\text{Rep},(F^+, f_0)}\) since \(\text{Func}_{\text{Rep},(F^+, f_0)}^{AB, \text{C,F}}\) will never output inconsistent values.

On the other hand, \(Z_1\) outputs 1 whenever \(A_1\) detects an inconsistency. Hence, the distinguishing advantage equals \(Pr[\text{in-CONS}_{FE}(1^\lambda, A_1)]\).

Case set-CONS: \(Z_2\) internally runs \(A_2\) until it outputs \((mpk_1, mpk_2, sk, f, x_0, x_1)\). \(Z_2\) then interacts with honest party \(A\) and \(B_1\) as follows via the party \(C\) that it corrupts: it instructs the corrupted party to send (via the channels) \(mpk_1\) and \(mpk_2\) to \(B_1\). 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 \(A\) to obtain the handles \(h_1\) and \(h_2\). Finally, it provides the input \((\text{READ}, h_1, f)\) to party \(B_1\). If it obtains an answer \(y_1\) (and the input is hence not ignored) and \(y_1 \neq f(x_j)\), then \(Z_2\) outputs 1 as its decision bit. Otherwise, it provides the input \((\text{READ}, h_2, f)\) to party \(B_1\). If it obtains an answer \(y_2\) (and the input is hence not ignored) and \(y_2 \neq f(x_{1-j})\), then \(Z_2\) outputs 1 as its decision bit. Finally, if exactly one query returned an answer, then \(Z_2\) outputs 1 as its decision bit. In any other case, \(Z_2\) outputs 0.

By assumption, the probability that at least one of the equation \(\text{Dec}(mpk_x, f, sk, \text{Enc}(mpk_1, x_j)) \neq \diamond\) holds is at least \(Pr[\text{set-CONS}(1^\lambda, A_2)]\) and therefore, with probability at least \(\frac{Pr[\text{set-CONS}(1^\lambda, A_2)]}{2}\) 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 \(S'\), then by definition of \(\text{Func}^{AB, \text{C,F}}_{\text{Rep},(F^+, f_0)}\), 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 \(Z_2\) outputs 1 when interacting with the ideal world is zero and we obtain a distinguishing advantage of at least \(\frac{Pr[\text{set-CONS}(1^\lambda, A_2)]}{2}\) for \(Z_2\).

Case st-in-CONS: \(Z_3\) internally runs \(A_3\) which outputs \((mpk, ct_0, ct_1, \{(sk_k, f_1), \ldots, (sk_n, f_n)\})\). \(Z_3\) then corrupts parties \(A\) and \(C\) and instructs party \(C\) to send \(mpk\) to (honest) party \(B_1\). \(Z_3\) then instructs party \(A\) to write each ciphertext \(ct_i\) 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 \(C\) to send \(sk_i\) to party \(B_1\), followed by a query \((\text{READ}, \text{sid}, h_1, f_i)\).

2) If the above requests got ignored, then it instructs party \(C\) to recrypt \(sk_i\). In any case, then it issues \((\text{READ}, \text{sid}, h_2, f_i)\).

3) When exactly one of the two queries gets ignored, then \(Z_3\) outputs decision bit 1 and halts. If both returned a value, it records them as \(y_i^{\diamond j}\) and proceeds with the next functional key.

If no decision has been reached, then \(Z_3\) defines the sets \(S_j := \{x' \in M | \forall i : f_i(x') = y_i^{\diamond j}\}\) 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, \(Z_3\) outputs 0. When \(Z_3\) interacts with the protocol, then the values \(y_i^{\diamond j}\) are computed exactly as in the game st-in-CONS(1^\lambda, 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 \(Z_3\)). On the other hand, if \(Z_3\) is interacting with the ideal system, then it would never output 1, as \(\text{Func}_{\text{Rep},(F^+, f_0)}^{AB, \text{C,F}}\) does either answer both queries per evaluation for \(f_i\) or none. Furthermore, the above sets \(S_j\) are non-empty as ensured by \(\text{Func}_{\text{Rep},(F^+, f_0)}^{AB, \text{C,F}}\) (ensured by instruction (++)). 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 \(A_3\), yielding a distinguishing advantage of \(Pr[\text{st-in-CONS}(1^\lambda, A_3)]\).

Universal encryption property.: We finally turn our attention to \(\text{Func}_{\text{Rep},(F^+, f_0)}^{AB, \text{C,F}}\). The only difference to \(\text{Func}_{\text{Rep},(F^+, f_0)}^{AB, \text{C,F}}\) is the case treated in Lemma 13 (dishonest setup generator, honest input provider): \(\text{Func}_{\text{Rep},(F^+, f_0)}^{AB, \text{C,F}}\) does, upon an input by Alice, not provide a public delay output revealing \(x\) (instead, \(x\) is kept private); recall that the simulator \(S\) 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 $\text{mpk}_1$ 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. \qed