Construction of Fully CCA-Secure Predicate Encryptions from Pair Encoding Schemes*

Johannes Blömer and Gennadij Liske**

Paderborn University, Germany
bloemer@upb.de, gennadij.liske@upb.de

Abstract. This paper presents a new framework for constructing fully CCA-secure predicate encryption schemes from pair encoding schemes. Our construction is the first in the context of predicate encryption which uses the technique of well-formedness proofs known from public key encryption. The resulting constructions are simpler and more efficient compared to the schemes achieved using known generic transformations from CPA-secure to CCA-secure schemes. The reduction costs of our framework are comparable to the reduction costs of the underlying CPA-secure framework. We achieve this last result by applying the dual system encryption methodology in a novel way.

Keywords: predicate encryption schemes, chosen-ciphertext security, key-encapsulation mechanisms, full security, pair encoding schemes

---

* A preliminary version of this paper appears in Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, Lecture Notes in Computer Science Vol. 9610, Kazue Sako ed., Springer-Verlag, 2016. This is the full version.

** The authors were partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

Table of Contents

1 Introduction

Predicate encryption (PE) with public index, as a subclass of functional encryption [8], is a powerful generalization of traditional public-key encryption (PKE). In a PE system for a predicate R, data are encrypted under so-called ciphertext indices cInd, which are public. A user can decrypt such a ciphertext if she holds a secret key with a key index kInd, such that $\mathrm{R}(\mathrm{kInd}, \mathrm{cInd}) = 1$. Identity-based encryption (IBE) schemes realize the equality relation and are the simplest example of PE. In general, predicate encryption schemes provide a powerful tool for achieving fine-grained access control on confidential data.

Except for IBE, constructions of fully (also called adaptively) secure PEs have been missing for a long time. The dual system encryption methodology, introduced and extended by Waters and Lewko [21, 16], provides fundamental techniques to achieve fully secure PE schemes which withstand chosen-plaintext attacks (CPA). Based on this methodology, schemes for various predicates such as (hierarchical) identity-based encryption [21, 16], attribute-based encryption [17, 15], inner-product encryption [18, 4], spatial encryption [12, 4], and schemes for regular languages [3], to name just a few, have been constructed.

Although many PE schemes have been presented, constructions for new predicates have each been built from the ground up until the following results were published. Attrapadung [3] and Wee [22] independently introduced generic frameworks for the design and analysis of PE schemes with public index from composite-order bilinear groups. These frameworks are based on the dual system encryption methodology and define new cryptographic primitives called pair encoding and predicate encoding. Attrapadung and Wee showed that fully CPA-secure PEs can be constructed from encoding schemes in a generic fashion. This approach simplifies the development of new schemes, since the complexity of security proofs is reduced. Furthermore, the properties required to achieve secure constructions are better understood, structured, and defined in terms of security properties of encodings. Recently, both frameworks were adapted to prime-order groups in [2, 1] and in [9], respectively. Overall, the research on encodings resulted in new and efficient CPA-secure schemes for various predicates. In this paper, we extend the framework of Attrapadung [3] to achieve fully CCA-secure PE schemes. We chose this framework because of its powerful computational (rather than information theoretic) security notion which allows to capture involved predicates. Although this will be a non-trivial task, we believe that our techniques can be applied to the pair encoding framework in prime order groups [2].

Related work. Although there exist many adaptively CPA-secure PE schemes for various predicates, only a few papers consider the realization of fully secure schemes which withstand chosen-ciphertext attacks (CCA), the most desirable security notion in practice. Comparing this situation with PKE schemes and IBE schemes, we identify the following gap. Mainly two different approaches are known to achieve *efficient* CCA-secure schemes *without random oracle model* in the context of PKE and IBE (cf. discussion in [7]). The first approach goes back to the CCA-secure PKE schemes introduced in [11]. Schemes following this approach achieve CCA-security using a kind of *well-formedness proofs*, exploit specific properties of the underlying CPA-secure schemes, and sacrifice generality for efficiency. The second approach goes back to the generic transformations presented in [7] and uses one-time signatures or message authentication codes as building blocks. Whereas both approaches are well studied for PKE [11, 10, 7] and (hierarchical) IBE [13, 14, 7] this is not the case for PE with more involved predicates.

Generic transformations of CPA-secure PE schemes into CCA-secure schemes presented in [23, 24] pursue the second approach from above and use one-time signatures as a building block. However, the first approach of well-formedness proofs has not been taken into account for PEs. Indeed, only a few PE schemes are proven to be fully CCA-secure without applying the generic transformations from [23, 24]. To the best of our knowledge these are the broadcast-encryption scheme from [19] and the (index hiding) encryption for relations that are specified by non-monotone access structures combined with inner product relations [18]. The techniques from [19] are closely related to the techniques used for adaptively secure IBE schemes. The schemes from [18] achieve CCA-security using one-time signature schemes and their techniques are closely related to [24].

We can only speculate why the non-generic approach of well-formedness proofs from [11] has not been considered for fully secure predicate encryption schemes. Probably because of the complex structure of the ciphertexts in PE schemes well-formedness proofs have been assumed to be inefficient. Furthermore, the consistency checks for the ciphertexts seem to be in conflict with the dual system encryption methodology, since an essential part of this technique is based on incorrectly formed ciphertexts, i.e. semi-functional ciphertexts. In this work we show that these assumptions are premature. We show that the dual system encryption techniques can be combined with well-formedness proofs and that the resulting fully CCA-secure PE schemes require computational overhead, which is comparable to the additional overhead required by the generic transformations.

Our contribution. In this work we take a significant step to close the gap between PKE/IBE and PE w.r.t. non-generic CCA-secure constructions. Namely, given any pair encoding scheme (with natural restrictions) secure in terms of [3], we construct a fully CCA-secure key-encapsulation mechanism (KEM) for the corresponding predicate using a kind of well-formedness proofs. Surprisingly, due to the pair encoding abstraction, we achieve a semi-generic transformation and still exploit structural properties of the underlying CPA-secure schemes. Since the underlying framework of [3] is defined on composite-order groups, our construction is also build on these groups. Combined with an appropriate symmetric encryption, our framework leads to various new fully CCA-secure PE schemes through the usual hybrid construction. In fact, for efficiency reasons hybrid schemes are preferred to plain encryption schemes in practice.

Although our extensions of CPA-secure schemes are similar to those used in PKE schemes, the application to complex predicates as well as the generic nature of our construction are novel for the underlying techniques. We achieve simpler and usually more efficient constructions than those obtained from CPA-secure schemes and the generic transformations based on one-time signatures [23, 24]. Furthermore, we keep the advantage of tight reductions from the original framework of Attrapadung [3], and the reduction costs of our CCA-secure construction are comparable to the reduction costs of the underlying CPA-secure construction. This is indeed surprising and is due to our extension of the dual system encryption methodology which we describe below. The only additional cryptographic primitive required by our construction is a collision-resistant hash function, which is used to add a single redundant group element to the ciphertext. Apart from that, we add two group elements to the public parameters of the underlying CPA-secure scheme. The security of our framework is based on the same security assumptions as the security of the original CPA-secure framework.

Moving beyond the dual system encryption methodology. Security proofs in cryptography often consist of a sequence of probability experiments (or games) with small differences. The first experiment is the target security experiment (CCA-security experiment in our case) whereas the last experiment is constructed in such a way, that the adversaries cannot achieve any advantage. The task of the proof is to show that consecutive experiments are computationally indistinguishable. This proof structure is also used in dual system encryption methodology [21], but the changes between the experiments are quite special. The main idea of this technique is to define so-called semi-functional keys and semi-functional ciphertexts, which are indistinguishable from their normal counterparts. In the proof sequence, the challenge ciphertext and all generated keys are transformed from normal to semi-functional one by one. In the last experiment, when all elements are modified, the challenge can be changed to the ciphertext of a randomly chosen message.

The obvious way to apply dual system encryption methodology in the context of CCA-security is to treat keys used to answer decryption queries in the same way as keys queried by the adversary. This strategy was followed in [18] (see discussion of this work below), but our proof strategy diverges from it. We deal with decryption queries in a novel and surprisingly simple manner. As an additional advantage, the reductions of the original CPA-security proof require only a few and simple modifications. The main idea is to answer decryption queries in *all* games using separately generated *normal* keys. Our well-formedness checks ensure that this modification cannot be noticed. Moreover, we ensure that normal and semi-functional ciphertexts both pass our well-formedness checks. Mainly because of this approach, we can keep the basic structure of the original CPA-security proof of Attrapadung. We only have to add four additional experiments: three at the beginning and one before the last game. In our last game we show that by using the redundant element added to the ciphertext we can answer all decryption queries without the user secret keys. The indistinguishability for this experiment is again based on our well-formedness checks.

The main advantage of our construction and our proof strategy becomes obvious if compared to the techniques in [18], where *all keys* are changed and the security guarantees decrease linearly in the number of decryption queries and the number of corrupted keys. In our approach, the number of decryption queries influences the security guarantees only negligibly. In a realistic scenario, the number of decryption queries must be assumed to be much larger than the number of corrupted keys. Hence, our approach results in smaller security parameters, which also increases efficiency.

Organization. In Section 2 we present the preliminaries including security definitions and assumptions. Section 3 contains our formal requirements on pair encoding schemes and our fully CCA-secure framework. In Section 4 we present our main theorem and explain our proof strategy. Finally, in Section 5 we compare our resulting schemes with generic constructions and conclude.

2 Background

We denote by $\alpha := a$ the algorithmic action of assigning the value a to the variable α. For $n \in \mathbb{N}$, we denote by $[n]$ the set $\{i \in \mathbb{N} \mid 1 \leq i \leq n\}$ and by $[n]_0$ the set $[n] \cup \{0\}$. Let X be a random variable on a finite set S. We denote by $[X]$ the support of X, that is $[X] = \{s \in S \mid \Pr[X = s] > 0\}$. We write $\alpha \leftarrow X$ to denote the algorithmic action of sampling an element of S according to the distribution defined by X. We also write $\alpha \leftarrow S$ when sampling an element from S according to the uniform distribution. Furthermore, $\alpha_1, \ldots, \alpha_n \leftarrow X$ is a shortcut for $\alpha_1 \leftarrow X, \ldots, \alpha_n \leftarrow X$. This notation can be extended to probabilistic polynomial time (ppt) algorithms, since every ppt algorithm $\mathcal{A}$ on input x defines a finite output probability space denoted by $\mathcal{A}(x)$. Finally, vectors are written in bold and we do not distinguish between row and column vectors. It will be obvious from context what we mean. We usually denote the components of a vector $\boldsymbol{v}$ by $(v_1, \ldots, v_n)$, where $n = |\boldsymbol{v}|$.

2.1 Predicate Families

In general, a predicate family is just a set of relations. For our purposes we require a more specific definition. Compared to [3], we give a more fine grained definition. Namely, we split the relation indexes into two parts which play different roles.

Definition 2.1. *Let Ω, Σ be arbitrary sets. A* ***predicate family*** *$\mathcal{R}_{\Omega,\Sigma}$ (or just $\mathcal{R}$) is a set of relations*

$$\mathcal{R} = \{\mathrm{R}_{\mathrm{des},\mathrm{dom}} : \mathbb{X}_{\mathrm{des},\mathrm{dom}} \times \mathbb{Y}_{\mathrm{des},\mathrm{dom}} \to \{0,1\}\}_{\mathrm{des}\in\Omega,\mathrm{dom}\in\Sigma} \quad ,$$

where $\mathbb{X}_{\mathrm{des},\mathrm{dom}}$ and $\mathbb{Y}_{\mathrm{des},\mathrm{dom}}$ are sets called the key index space and the ciphertext index space of $\mathrm{R}_{\mathrm{des},\mathrm{dom}}$, respectively. Relation indexes will be often denoted by $\kappa = (\mathrm{des}, \mathrm{dom})$ and the corresponding relations will be denoted by R_κ.

By the definition, every predicate $\mathrm{R}_{\mathrm{des},\mathrm{dom}} \in \mathcal{R}_{\Omega,\Sigma}$ is uniquely defined by two indices. In our context, every index $\mathrm{des} \in \Omega$ specifies some general description properties of the corresponding predicates (e.g. maximal number of attributes in a key). This index will be chosen by the system administrator before the setup of the system. Index $\mathrm{dom} \in \Sigma$ specifies domain properties which will depend on the security parameter (e.g. domain of computation $\mathbb{Z}_N$) and will be determined by the corresponding setup algorithm.

For every $\mathrm{des} \in \Omega$ we denote by $\mathcal{R}_{\mathrm{des}}$ the following subfamily of predicates

$$\mathcal{R}_{\mathrm{des}} = \{\mathrm{R}_{\mathrm{des},\mathrm{dom}} : \mathbb{X}_{\mathrm{des},\mathrm{dom}} \times \mathbb{Y}_{\mathrm{des},\mathrm{dom}} \to \{0,1\}\}_{\mathrm{dom}\in\Sigma} \subseteq \mathcal{R} \quad .$$

Furthermore, if $\mathrm{des} \in \Omega$ is fixed and obvious from context, we will simply write $\mathrm{R}_{\mathrm{dom}}$, $\mathbb{X}_{\mathrm{dom}}$ and $\mathbb{Y}_{\mathrm{dom}}$.

Our framework is defined over composite order groups and hence, we have to take care of zero-divisors in $\mathbb{Z}_N$ for composite $N \in \mathbb{N}$. The following definition is adapted from [3] to our notation and specifies the properties of the predicate families which are required for the framework.

Definition 2.2. *A predicate family $\mathcal{R}_{\Omega,\Sigma}$ is called* ***domain-transferable*** *if $\Sigma \subseteq \mathbb{N}$, for every $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$, and every $p \in \mathbb{N}^{>1}$ with $p \mid N$ it holds $\kappa' = (\mathrm{des}, p) \in \Omega \times \Sigma$, and $\mathbb{X}_{\kappa'} \subseteq \mathbb{X}_\kappa$, $\mathbb{Y}_{\kappa'} \subseteq \mathbb{Y}_\kappa$. Furthermore, there must exist a ppt algorithm* Factor *and projection maps $f_1 : \mathbb{X}_\kappa \mapsto \mathbb{X}_{\kappa'}$ and $f_2 : \mathbb{Y}_\kappa \mapsto \mathbb{Y}_{\kappa'}$ such that for all $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ it holds:*

Completeness: *If $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, then $\mathrm{R}_{\kappa'}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd})) = 1$.*

Soundness: *If $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 0$ and $\mathrm{R}_{\kappa'}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd})) = 1$, then a non-trivial factor F of N can be computed by $F := \mathrm{Factor}(\kappa, \mathrm{kInd}, \mathrm{cInd})$.*

2.2 Predicate Key-Encapsulation Mechanisms

In this subsection we present the definition of predicate key-encapsulation mechanisms (P-KEMs) and the definition of full security against adaptively chosen-ciphertext attacks (also called CCA2 attacks) for these schemes. P-KEMs combined with appropriate symmetric encryption schemes lead to fully functional predicate encryptions through the usual hybrid construction. For the sake of completeness we present this construction and the corresponding security proof in Appendix C.

Definition 2.3. *Let $\mathcal{K} = \{\mathbb{K}_\lambda\}$ be a family of finite sets indexed by security parameter λ and possibly some further parameters. A* ***predicate key-encapsulation mechanism*** Π *for predicate family $\mathcal{R}_{\Omega,\Sigma}$ and a family of key spaces $\mathcal{K}$ consists of four ppt algorithms:*

Setup $\left(1^\lambda, \mathrm{des}\right) \to (\mathrm{msk}, \mathrm{pp}_\kappa)$ *: takes as input security parameter λ, $\mathrm{des} \in \Omega$, and outputs a master secret key and public parameters. The algorithm determines among other elements $\mathrm{dom} \in \Sigma$ and the relation index $\kappa = (\mathrm{des}, \mathrm{dom})$ is (implicitly) included in pp_κ.*

KeyGen $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right) \to \mathrm{sk}$ *: takes as input the master secret key* msk *and a key index* $\mathrm{kInd} \in \mathbb{X}_\kappa$*. It generates a user secret key* sk *for* kInd.

Encaps $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}\right) \to (\mathrm{K}, \mathrm{CT})$ *: takes as input a ciphertext index* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *and outputs a key* $\mathrm{K} \in \mathbb{K}_\lambda$*, and an encapsulation* CT *of this key.*

Decaps $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT}\right) \to \mathrm{K}$ *: takes as input a secret key* sk *and an encapsulation. It outputs a key* $\mathrm{K} \in \mathbb{K}_\lambda$ *or an error symbol* $\perp \notin \mathbb{K}_\lambda$*.*

Correctness: *For every security parameter λ, every $\mathrm{des} \in \Omega$, every $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, every $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ with $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, every $\mathrm{sk} \in \left[\mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right)\right]$ and $(\mathrm{K}, \mathrm{CT}) \in \left[\mathrm{Encaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}\right)\right]$ it must hold that*

$$\Pr\left[\mathrm{Decaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT}\right) = \mathrm{K}\right] = 1 \ .$$

We will leave out 1^λ and pp_κ from the input of the algorithms, if these are obvious from the context. Furthermore, for every $\mathrm{kInd} \in \mathbb{X}_\kappa$ and every $\mathrm{cInd} \in \mathbb{Y}_\kappa$ we denote by $\mathbb{SK}_{\mathrm{kInd}}$ and by $\mathbb{C}_{\mathrm{cInd}}$ the sets of syntactically correct secret keys and encapsulations, respectively. These sets are certain supersets of corresponding correctly generated elements and represent their syntactic structure, which can be easily checked (e.g. the correct number of group elements).

CCA Security Definition for P-KEMs. Whereas in the context of traditional PKE there is only a single secret key in question, in PE schemes there are many user secret keys generated from the master secret key. Actually, several users may have different keys for the same key index. In order to model this issue, we have to give the adversary the possibility to specify not only the key index, but also the keys which have to be used for answering decapsulation queries. Similar to [19], we model this using so-called covered key generation queries.

Let Π be a P-KEM for predicate family $\mathcal{R}_{\Omega,\Sigma}$ and family $\mathcal{K} = \{\mathbb{K}_\lambda\}$ of key spaces. The CCA-security experiment $\text{aP-KEM}^{\text{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ between challenger $\mathcal{C}$ and adversary $\mathcal{A}$ is defined next. In this experiment, index i denotes the number of a covered key generation query and kInd_i denotes the key index used in the query with number i. W.l.o.g. we assume that $\mathcal{A}$ uses index i in the oracle queries only after the i'th query to the covered key generation oracle. In the security proof we will change this experiment step by step. Those parts of the experiment, which will be changed later, are framed and numbered.

The advantage of $\mathcal{A}$ in security experiment $\text{aP-KEM}^{\text{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ is defined as

$$\text{Adv-aP-KEM}^{\text{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) := \Pr\left[\text{aP-KEM}^{\text{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - 1/2 \ .$$

Definition 2.4. *A predicate key encapsulation mechanism Π for predicate family $\mathcal{R}_{\Omega,\Sigma}$ is called* ***fully (or adaptively) secure against adaptively chosen-ciphertext attacks (or CCA2 secure)*** *if for every $\mathrm{des} \in \Omega$ and every ppt adversary $\mathcal{A}$ the function $\text{Adv-aP-KEM}^{\text{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ is negligible in λ.*

$\textbf{aP-KEM}^{\mathrm{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$:

Setup : $\mathcal{C}$ generates ⟨1⟩ $\boxed{(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}(1^\lambda, \mathrm{des})}$ and starts $\mathcal{A}$ on input $(1^\lambda, \mathrm{pp}_\kappa)$.

Phase I : $\mathcal{A}$ has access to the following oracles:

CoveredKeyGen (kInd_i) for $\mathrm{kInd}_i \in \mathbb{X}_\kappa$: Challenger $\mathcal{C}$ generates a secret key for kInd_i ⟨2⟩ $\boxed{\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)}$, stores $(\mathrm{kInd}_i, \mathrm{sk}_i)$, and returns nothing.

Open (i) for $i \in \mathbb{N}$: $\mathcal{C}$ returns ⟨3⟩ $\boxed{\mathrm{sk}_i}$. We call the corresponding key index kInd_i a corrupted key index.

Decapsulate (CT, i) with $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ for some $\mathrm{cInd} \in \mathbb{Y}_\kappa$, and $i \in \mathbb{N}$: $\mathcal{C}$ returns the decapsulation ⟨4⟩ $\boxed{\mathrm{Decaps}(\mathrm{sk}_i, \mathrm{CT})}$. [a]

Challenge : $\mathcal{A}$ submits a target ciphertext index $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$ under the restriction that for every corrupted key index kInd it holds $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}^*) = 0$. $\mathcal{C}$ computes ⟨5⟩ $\boxed{(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{Encaps}(\mathrm{cInd}^*)}$, chooses $\mathrm{K}_1 \leftarrow \mathbb{K}_\lambda$, flips a bit $b \leftarrow \{0,1\}$, sets ⟨6⟩ $\boxed{\mathrm{K}^* := \mathrm{K}_b}$, and returns the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.

Phase II : $\mathcal{A}$ has access to the following oracles:

CoveredKeyGen (kInd_i) for $\mathrm{kInd}_i \in \mathbb{X}_\kappa$: As before, challenger $\mathcal{C}$ generates a secret key ⟨7⟩ $\boxed{\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)}$, stores $(\mathrm{kInd}_i, \mathrm{sk}_i)$, and returns nothing.

Open (i) for $i \in \mathbb{N}$: $\mathcal{C}$ returns ⟨8⟩ $\boxed{\mathrm{sk}_i}$ under the restriction that $\mathrm{R}_\kappa(\mathrm{kInd}_i, \mathrm{cInd}^*) = 0$.

Decapsulate (CT, i) with $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ for some $\mathrm{cInd} \in \mathbb{Y}_\kappa$, and $i \in \mathbb{N}$: $\mathcal{C}$ returns the error symbol $\perp$ if $\mathrm{CT} = \mathrm{CT}^*$. [b] Otherwise, $\mathcal{C}$ returns ⟨9⟩ $\boxed{\mathrm{Decaps}(\mathrm{sk}_i, \mathrm{CT})}$.

Guess : $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. If one of the restrictions is violated, the output of the experiment is 0. ⟨10⟩ $\boxed{\text{The output of the experiment is 1 iff } b' = b}$.

[a] For schemes, where cInd is not efficiently computable from CT, the decapsulation oracle requires the ciphertext index as additional input.

[b] For schemes, where the decapsulation oracle requires cInd in addition, a query on CT^* is allowed if $\mathrm{cInd} \neq \mathrm{cInd}^*$.

2.3 Composite Order Bilinear Groups

In this section we briefly recall the main properties of composite order bilinear groups (cf. [16]). We define these groups using a group generation algorithm $\mathcal{G}$, a ppt algorithm which takes as input a security parameter 1^λ and outputs a description $\mathbb{GD}$ of bilinear groups. We require that $\mathcal{G}$ outputs

$$\mathbb{GD} = (p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_{\mathrm{T}}, \mathrm{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathrm{T}}) \quad ,$$

where p_1, p_2, p_3 are distinct primes of length λ, $\mathbb{G}$ and $\mathbb{G}_{\mathrm{T}}$ are cyclic groups of order $N = p_1 p_2 p_3$, g is a generator of $\mathbb{G}$, and function e is a non-degenerate bilinear map: i.e., $\mathrm{e}(g^a, g^b) = \mathrm{e}(g, g)^{a \cdot b}$ for all $a, b \in \mathbb{Z}_N$, and $\mathrm{e}(g, g)$ is a generator of $\mathbb{G}_{\mathrm{T}}$. Furthermore, we denote by $\mathbb{GD}_N$ a restricted group description corresponding to $\mathbb{GD}$, where the prime numbers are replaced by N. We require that the group operations as well as the bilinear map e are computable in polynomial time with respect to λ when the restricted group description $\mathbb{GD}_N$ is given.

$\mathbb{G}$ can be decomposed as $\mathbb{G}_{p_1} \times \mathbb{G}_{p_2} \times \mathbb{G}_{p_3}$, where for every $p_i \mid N$ we denote by $\mathbb{G}_{p_i}$ the unique subgroup of $\mathbb{G}$ of order p_i. Let g_i be an arbitrary but fixed generator of $\mathbb{G}_{p_i}$. Every $h \in \mathbb{G}$ can be expressed as $g_1^{a_1} g_2^{a_2} g_3^{a_3}$, where a_i are uniquely defined modulo p_i. Hence, we will call $g_i^{a_i}$ the $\mathbb{G}_{p_i}$ component of h. Note that, e.g., $g^{p_1 p_2}$ generates $\mathbb{G}_{p_3}$ and hence, given the factorization of N, we can pick random elements from every subgroup. A further important property of composite order bilinear groups is that for $p_i \neq p_j$ and $g_i \in \mathbb{G}_{p_i}$, $g_j \in \mathbb{G}_{p_j}$ it holds $\mathrm{e}(g_i, g_j) = 1_{\mathbb{G}_{\mathrm{T}}}$.

We will also use the following common shortcuts for vectors of group elements. Let $g, h, r \in \mathbb{G}$, $\boldsymbol{v}, \boldsymbol{w}, \boldsymbol{u} \in \mathbb{Z}_N^k$, and $\boldsymbol{E} \in \mathbb{Z}_N^{k \times d}$ for $k, d \in \mathbb{N}$. We denote by $g^{\boldsymbol{v}}$ the vector $(g^{v_1}, g^{v_2}, \ldots, g^{v_k}) \in \mathbb{G}^k$. Furthermore, we define $g^{\boldsymbol{v}} \cdot g^{\boldsymbol{w}} := g^{\boldsymbol{v}+\boldsymbol{w}}$, $(g^{\boldsymbol{v}})^{\boldsymbol{E}} := g^{\boldsymbol{v} \cdot \boldsymbol{E}}$, and $\mathrm{e}(g^{\boldsymbol{v}}, h^{\boldsymbol{w}}) := \prod_{i=1}^{k} \mathrm{e}(g^{v_i}, h^{w_i})$. Hence, it also holds $\mathrm{e}(g^{\boldsymbol{v}}, h^{\boldsymbol{w}} \cdot r^{\boldsymbol{u}}) = \mathrm{e}(g^{\boldsymbol{v}}, h^{\boldsymbol{w}}) \cdot \mathrm{e}(g^{\boldsymbol{v}}, r^{\boldsymbol{u}})$. Furthermore, given $g^{\boldsymbol{v}}$ and $\boldsymbol{E}$ one can efficiently compute components of $(g^{\boldsymbol{v}})^{\boldsymbol{E}} \in \mathbb{G}^d$.

Remark 2.1. We will often pick group elements from $\mathbb{G}$ and its subgroups uniformly at random. Thereby we will always assume, that a generator of the corresponding group is chosen. This is only a technical convention, since the opposite happens only with negligible probability.

2.4 Security Assumptions

In this subsection we define three so-called Subgroup Decision Assumptions used to prove the security of our construction. We use exactly the same assumptions as the original CPA-secure framework [3]. See also [16] for validity of these assumptions in the generic group model. Let $\mathcal{G}$ be a group generation algorithm. Each of the following probability experiments starts with

$$\mathbb{GD} = (p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_{\mathrm{T}}, \mathrm{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathrm{T}}) \leftarrow \mathcal{G}\left(1^{\lambda}\right) :$$

$$\begin{array}{ll}
\mathbf{SD1}(\lambda): & g_1 \leftarrow \mathbb{G}_{p_1}, \quad g_3 \leftarrow \mathbb{G}_{p_3} \ , \\
& D := (\mathbb{GD}_N, g_1, g_3), \quad Z_0 \leftarrow \mathbb{G}_{p_1}, \quad Z_1 \leftarrow \mathbb{G}_{p_1 p_2} \ . \\
\mathbf{SD2}(\lambda): & g_1, X_1 \leftarrow \mathbb{G}_{p_1}, \quad X_2, Y_2 \leftarrow \mathbb{G}_{p_2}, \quad g_3, Y_3 \leftarrow \mathbb{G}_{p_3} \ , \\
& D := (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3), \quad Z_0 \leftarrow \mathbb{G}_{p_1 p_3}, \quad Z_1 \leftarrow \mathbb{G} \ . \\
\mathbf{SD3}(\lambda): & g_1 \leftarrow \mathbb{G}_{p_1}, \quad g_2, X_2, Y_2 \leftarrow \mathbb{G}_{p_2}, \quad g_3 \leftarrow \mathbb{G}_{p_3}, \quad \alpha, s \leftarrow \mathbb{Z}_N \\
& D := (\mathbb{GD}_N, g_1, g_1^{\alpha} X_2, g_1^{s} Y_2, g_2, g_3), \ Z_0 \leftarrow \mathbb{G}_{\mathrm{T}}, \ Z_1 := \mathrm{e}(g_1, g_1)^{\alpha \cdot s} \ .
\end{array}$$

The advantage of $\mathcal{A}$ in breaking experiment SDi (λ) is defined as

$$\mathrm{Adv}_{\mathcal{A}}^{\mathrm{SD}i}(\lambda) := |\Pr[\mathcal{A}(D, Z_0) = 1] - \Pr[\mathcal{A}(D, Z_1) = 1]| \ .$$

Assumptions. We say that $\mathcal{G}$ satisfies Assumption SDi if for every ppt algorithm $\mathcal{A}$ the function $\mathrm{Adv}_{\mathcal{A}}^{\mathrm{SD}i}(\lambda)$ is negligible.

The following lemma was implicitly proven in [16] (see the proof of Lemma 5). This lemma implies, that under Assumption SD2, it is computationally infeasible to compute a non-trivial factor of N.

Lemma 2.1. *There exists a ppt algorithm $\mathcal{A}$ with*

$$|\Pr[\mathcal{A}(D, Z_0, F) = 1] - \Pr[\mathcal{A}(D, Z_1, F) = 1]| = 1 \ ,$$

where D, Z_0, Z_1 are distributed as defined in Experiment SD2, and F is a non-trivial factor of N (N is defined by $\mathbb{GD}_N \in D$).

Proof. See the proof in Appendix F.2.

3 Framework for CCA-Secure P-KEMs

In this section we recall the definition of pair encoding schemes and define two additional properties, which are required for our CCA-secure framework. Our framework is presented in Subsection 3.3.

3.1 Pair Encoding Schemes

In this subsection we first recall the formal definition of *pair encodings* presented by Attrapadung [3] and slightly adapted to our notation. This cryptographic primitive is used to construct predicate encryption schemes. Actually, pair encodings are multivariate polynomials which are evaluated during the key generation and during the encryption. The elements, used to evaluate the polynomials, will be chosen according to certain probability distributions. We separate the notation of polynomial variables and the notation of corresponding elements in the schemes, which is different from [3]. Every polynomial variable gets an index, which in turn will be the name of the corresponding element in the schemes. For example random element s corresponds to the polynomial variable X_s. This justifies the unusual names of variables.

Definition 3.1. *Let $\mathcal{R}_{\Omega,\Sigma}$ be a domain-transferable predicate family, $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$ be a predicate index, $\mathrm{kInd} \in \mathbb{X}_{\kappa}$ and $\mathrm{cInd} \in \mathbb{Y}_{\kappa}$ be a key index and a ciphertext index respectively. A* ***pair encoding scheme*** P *for $\mathcal{R}_{\Omega,\Sigma}$ consists of four ppt algorithms:*

Param $(\kappa) =: n$: *outputs $n \in \mathbb{N}$, which defines the number of so-called common variables denoted by $(\mathrm{X}_{h_1}, \ldots, \mathrm{X}_{h_n}) = \mathbf{X}_{\boldsymbol{h}}$.*

Enc1 $(\kappa, \mathrm{kInd}) =: (\boldsymbol{k}, m_2)$: *outputs* $m_2 \in \mathbb{N}$ *and a vector* $\boldsymbol{k} = (k_1, \ldots, k_{m_1})$ *of* m_1 *multivariate polynomials* $k_1, \ldots, k_{m_1} \in \mathbb{Z}_N [\mathrm{X}_\alpha, \mathbf{X}_{\boldsymbol{r}}, \mathbf{X}_{\boldsymbol{h}}]$. *The variable* X_α *is called the master secret key variable and the variables* $\left(\mathrm{X}_{r_1}, \ldots, \mathrm{X}_{r_{m_2}}\right) = \mathbf{X}_{\boldsymbol{r}}$ *are called key-specific variables. The polynomials in* $\boldsymbol{k}$ *are restricted to linear combinations of monomials* $\left\{\mathrm{X}_\alpha,\ \mathrm{X}_{r_i},\ \mathrm{X}_{h_j}\mathrm{X}_{r_i}\right\}_{i\in[m_2], j\in[n]}$.

Enc2 $(\kappa, \mathrm{cInd}) =: (\boldsymbol{c}, w_2)$: *outputs* $w_2 \in \mathbb{N}$ *and a vector* $\boldsymbol{c} = (c_1, \ldots, c_{w_1})$ *of* w_1 *multivariate polynomials* $c_1, \ldots, c_{w_1} \in \mathbb{Z}_N [\mathrm{X}_s, \mathbf{X}_{\boldsymbol{s}}, \mathbf{X}_{\boldsymbol{h}}]$. *The variable* X_s *and the variables* $\left(\mathrm{X}_{s_1}, \ldots, \mathrm{X}_{s_{w_2}}\right) = \mathbf{X}_{\boldsymbol{s}}$ *are called ciphertext-specific variables.*[1] *The polynomials in* $\boldsymbol{c}$ *are restricted to linear combinations of monomials* $\left\{\mathrm{X}_s,\ \mathrm{X}_{s_i},\ \mathrm{X}_{h_j}\mathrm{X}_s,\ \mathrm{X}_{h_j}\mathrm{X}_{s_i}\right\}_{i\in[w_2], j\in[n]}$.

Pair $(\kappa, \mathrm{kInd}, \mathrm{cInd}) \to \boldsymbol{E}$: *outputs a matrix* $\boldsymbol{E} \in \mathbb{Z}_N^{m_1\times w_1}$, *where* m_1 *and* w_1 *are defined by* $\mathrm{Enc1}(\kappa, \mathrm{kInd})$ *and* $\mathrm{Enc2}(\kappa, \mathrm{cInd})$, *respectively.*

Correctness*: Let* $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *be arbitrary. Let* $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$, *and* $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$. *The following three properties must be fulfilled:*

1. *If* $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}) = 1$, *then for every* $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ *it holds symbolically*

$$\sum_{\tau\in[m_1]} \sum_{\tau'\in[w_1]} e_{\tau,\tau'} \cdot k_\tau \cdot c_{\tau'} = \mathrm{X}_\alpha \cdot \mathrm{X}_s \quad .$$

2. *(Trivially holds for prime* N*) For every* $\mathrm{kInd} \in \mathbb{X}_\kappa$ *and every* $p \in \mathbb{N}^{>1}$ *with* $p \mid N$ *it holds*

$$\boldsymbol{k}' = \boldsymbol{k} \pmod p \qquad \textit{and} \qquad m_2 = m_2' \ ,$$

where $(\boldsymbol{k}', m_2') = \mathrm{Enc1}(\kappa, f_1(\mathrm{kInd}))$ *and* f_1 *is the projection map from domain-transferable property of* $\mathcal{R}_{\Omega,\Sigma}$.

3. *(Trivially holds for prime* N*) For every* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *and every* $p \in \mathbb{N}^{>1}$ *with* $p \mid N$ *it holds*

$$\boldsymbol{c}' = \boldsymbol{c} \pmod p \qquad \textit{and} \qquad w_2 = w_2' \ ,$$

where $(\boldsymbol{c}', w_2') = \mathrm{Enc2}(\kappa, f_2(\mathrm{cInd}))$ *and* f_2 *is the projection map from domain-transferable property* $\mathcal{R}_{\Omega,\Sigma}$.

As a notational convention, whenever a particular relation index κ, a key index $\mathrm{kInd} \in \mathbb{X}_\kappa$, and a ciphertext index $\mathrm{cInd} \in \mathbb{Y}_\kappa$ are under consideration, the following values are also implicitly defined: $n = \mathrm{Param}(\kappa)$, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$, and $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$. Note that differently from [3] we allow the algorithm Pair to be probabilistic. The results from [3] still hold with our definition.

Security Notions for Pair Encoding Schemes. We prove the security of our framework based on the computational security notions of pair encoding schemes presented in [3], i.e. selectively master-key hiding (SMH) and co-selectively master-key hiding (CMH). These security notions make the pair encoding framework so powerful. We refer to Appendix A for both definitions.

3.2 Additional Requirements of CCA-Secure Framework

In this subsection we formalize two properties of pair encoding schemes, which are sufficient to achieve CCA-secure P-KEMs using our framework. As in [5] we require *normality* of pair encoding P, a very natural restriction (this is also one of the restrictions of regular encodings from [2]).

Definition 3.2. *A pair encoding* P *for* $\mathcal{R}_{\Omega,\Sigma}$ *is called* ***normal****, if for every predicate index* $\kappa \in \Omega\times\Sigma$ *and every* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *there exists an integer* $\hat{\tau} \in [w_1]$ *such that it holds* $c_{\hat{\tau}} = \mathrm{X}_s$, *where* $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$. *W.l.o.g, we will assume that* $c_1 = \mathrm{X}_s$.

Next, we formally define the second required property, which we call verifiability property. As mentioned before, we have to check the form of the encapsulation to a certain extent in order to achieve CCA-security. The verifiability property itself does not ensure the CCA-security and has to be considered in the context of our extended framework. Hence, for the intuition behind this property we refer

[1] The variable X_s is separated, because it plays a special role in the algorithms. We will also denote it by X_{s_0}.

to the discussion in the next subsection. In Theorem 4.2 we will furthermore state that all regular pair encodings schemes are verifiable. The definition of regular encodings and a constructive proof of this theorem are given in Appendix E.

Let $\mathcal{R}_{\Omega,\Sigma}$ be a domain-transferable predicate family, $\mathcal{G}$ be a group generator and λ be a security parameter. Let $\mathbb{GD} \in [\mathcal{G}(1^\lambda)]$ and $\mathbb{GD}_N$ be the corresponding restricted group description. (We call $\mathcal{G}$ an appropriate group generator for $\mathcal{R}_{\Omega,\Sigma}$ if $N \in \Sigma$ for every $\mathbb{GD} \in [\mathcal{G}(1^\lambda)]$.) Furthermore, let $\mathrm{des} \in \Omega$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ be arbitrary but fixed such that $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, where $\kappa = (\mathrm{des}, N)$. Let $n = \mathrm{Param}(\kappa)$.

Definition 3.3. (Verifiability) P *is called* ***verifiable with respect to*** $\mathcal{G}$ *if it is normal and there exists a deterministic polynomial-time algorithm* Vrfy*, which given* des, $\mathbb{GD}_N$, $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, kInd*,* cInd*,* $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$*, and* $\boldsymbol{C} = (C_1, \ldots, C_{w_1}) \in \mathbb{G}^{w_1}$ *outputs* 0 *or* 1 *such that:*

Completeness: *The output is 1 if there exist* $s \in \mathbb{Z}_N$ *and* $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$ *such that the* $\mathbb{G}_{p_1}$ *components of the elements in* $\boldsymbol{C}$ *are equal to* $g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}$*, where* $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$.

Soundness: *If the output is 1, then for every* $\alpha \in \mathbb{Z}_N$*,* $\boldsymbol{r} \in \mathbb{Z}_N^{m_2}$ *it holds:*

$$\mathrm{e}\left(g_1^{\boldsymbol{k}(\alpha,\boldsymbol{r},\boldsymbol{h})\cdot\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}(g_1, C_1)^\alpha \quad , \tag{1}$$

where $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$.

Remark 3.1. Suppose that the verification algorithm outputs 1 *if and only if* there exist $s \in \mathbb{Z}_N$ and $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$ such that the $\mathbb{G}_{p_1}$ components of $\boldsymbol{C}$ are equal to $g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}$. Then, both required properties are satisfied due to the correctness of the pair encoding scheme, which ensures that for every $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ it holds $\boldsymbol{k}(\alpha, \boldsymbol{r}, \boldsymbol{h}) \cdot \boldsymbol{E} \cdot \boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h}) = \alpha \cdot s$.

Collision-Resistant Hash Functions. Our construction requires a collision-resistant hash function in order to hash elements from $\mathbb{Y}_\kappa$ and a restricted number of elements from $\mathbb{G}_\mathrm{T}$ into $\mathbb{Z}_N$. Such a function can be realized using an appropriate injective encoding function and a cryptographic hash function. The notion of collision-resistance is common and we refer to Appendix B for a formal definition. We denote by $\mathrm{H} \leftarrow \mathcal{H}_\kappa$ the random choice of such a function.

3.3 Fully CCA-Secure Framework

In this section we present our framework for constructing fully CCA-secure P-KEMs from pair encoding schemes. Let P be a verifiable pair encoding scheme for domain-transferable predicate family $\mathcal{R}_{\Omega,\Sigma}$ and Vrfy be the algorithm from Definition 3.3. Let $\mathcal{G}$ be a composite order group generator, and $\mathcal{H}$ be a family of appropriate collision-resistant hash functions. A P-KEM Π for $\mathcal{R}_{\Omega,\Sigma}$ is defined as follows:

Setup $(1^\lambda, \mathrm{des})$: If $\mathrm{des} \in \Omega$, generate $\mathbb{GD} \leftarrow \mathcal{G}(1^\lambda)$, $g_1 \leftarrow \mathbb{G}_{p_1}$ and $g_3 \leftarrow \mathbb{G}_{p_3}$. Set $\kappa := (\mathrm{des}, N)$, where $N = p_1 p_2 p_3$. Compute $n := \mathrm{Param}(\kappa)$, pick $\boldsymbol{h} \leftarrow \mathbb{Z}_N^n$, and compute $g_1^{\boldsymbol{h}}$. Choose $\alpha, u, v \leftarrow \mathbb{Z}_N$ and set $Y := \mathrm{e}(g_1, g_1)^\alpha$, $U_1 := g_1^u$, and $V_1 := g_1^v$. Choose $\mathrm{H} \leftarrow \mathcal{H}_\kappa$ and output $\mathrm{msk} := \alpha$ and $\mathrm{pp}_\kappa := \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$.

KeyGen $(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})$: If $\mathrm{kInd} \in \mathbb{X}_\kappa$, compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$ (let $m_1 = |\boldsymbol{k}|$). Pick $\boldsymbol{r} \leftarrow \mathbb{Z}_N^{m_2}$, $\boldsymbol{R}_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$, and compute $\boldsymbol{K} := g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot \boldsymbol{R}_3$. Output $\mathrm{sk} := (\mathrm{kInd}, \boldsymbol{K})$.

The key space for $\mathrm{kInd} \in \mathbb{X}_\kappa$ is $\mathbb{SK}_{\mathrm{kInd}} := \{\mathrm{kInd}\} \times \mathbb{G}^{m_1}$.

Encaps $(\mathrm{pp}_\kappa, \mathrm{cInd})$: If $\mathrm{cInd} \in \mathbb{Y}_\kappa$, compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd})$ (let $w_1 = |\boldsymbol{c}|$). Pick $s \leftarrow \mathbb{Z}_N$, $\boldsymbol{s} \leftarrow \mathbb{Z}_N^{w_2}$, and compute $\boldsymbol{C} := g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})} = (C_1, \ldots, C_{w_1})$. Compute

$$t := \mathrm{H}(\mathrm{cInd}, \mathrm{e}(g_1, C_1), \ldots, \mathrm{e}(g_1, C_{w_1})) \tag{2}$$

and $C'' := (U_1^t \cdot V_1)^s$. Set $\mathrm{CT} := (\mathrm{cInd}, \boldsymbol{C}, C'')$, $\mathrm{K} := Y^s$, and output $(\mathrm{K}, \mathrm{CT})$. The ciphertext space for $\mathrm{cInd} \in \mathbb{Y}_\kappa$ is $\mathbb{C}_{\mathrm{cInd}} := \{\mathrm{cInd}\} \times \mathbb{G}^{w_1+1}$.

Note that, given $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$, the corresponding hash value can be computed efficiently. We denote by $\mathrm{HInput}(\mathrm{CT})$ the input of the hash function as defined in (2).

Decaps $(\mathrm{pp}_\kappa, \mathrm{sk}, \mathrm{CT})$: It must hold $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'') \in \mathbb{C}_{\mathrm{cInd}}$ for $\mathrm{cInd} \in \mathbb{Y}_\kappa$ and $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K}) \in \mathbb{SK}_{\mathrm{kInd}}$ for $\mathrm{kInd} \in \mathbb{X}_\kappa$. Output $\perp$ if $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}) \neq 1$. Compute $t := \mathrm{H}(\mathrm{HInput}(\mathrm{CT}))$ and $\boldsymbol{E} \leftarrow \mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})$. Output $\perp$, if one of the following checks fails:

$$\mathrm{e}(C'', g_1) \stackrel{?}{=} \mathrm{e}(C_1, U_1^t \cdot V_1) \ , \tag{3}$$

$$\mathrm{e}(C'', g_3) \stackrel{?}{=} 1 \text{ and } \forall_{i \in [w_1]} : \mathrm{e}(C_i, g_3) \stackrel{?}{=} 1 \ , \tag{4}$$

$$\mathrm{Vrfy}\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, \mathrm{kInd}, \mathrm{cInd}, \boldsymbol{E}, \boldsymbol{C}\right) \stackrel{?}{=} 1 \ . \tag{5}$$

Output $\mathrm{K} := \mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right)$.

Correctness is based mainly on the correctness of pair encoding and the completeness of the verification algorithm (see the proof in Appendix F.1). Compared to the original CPA-secure framework of [3] we add only the hash function H and the group elements $U_1, V_1 \in \mathbb{G}$ to the public parameter. The user secret keys are not changed at all. The encapsulation is extended by a single group element $C'' \in \mathbb{G}$. The checks in (3), (4) and (5) are new in the decapsulation algorithm. We call these checks *consistency checks* and explain them in more detail below.

Semi-functional algorithms. The following semi-functional algorithms are basically from [3] and are essential to prove adaptive security of the original and our extended framework. The main idea is to extend the keys and the ciphertexts with components from $\mathbb{G}_{p_2}$ subgroup. These modifications cannot be noticed by a ppt adversary mainly due to the subgroup decision assumptions and since the public parameters do not contain a generator of $\mathbb{G}_{p_2}$. We extended the algorithms from [3] by semi-functional components for our additional elements in the public parameters (U_1, V_1) and in the encapsulation (C'').

SFSetup $(1^\lambda, \mathrm{des})$: Generate $(\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}(1^\lambda, \mathrm{des})$, $g_2 \leftarrow \mathbb{G}_{p_2}$, $\hat{\boldsymbol{h}} \leftarrow \mathbb{Z}_{p_2}^n$ and $\hat{u}_2, \hat{v}_2 \leftarrow \mathbb{Z}_{p_2}$. Output $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)$.

SFKeyGen $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, \mathrm{type}, \hat{\alpha}, g_2, \hat{\boldsymbol{h}}\right)$ for $\hat{\alpha} \in \mathbb{Z}_N$: Generate a normal secret key $(\mathrm{kInd}, \boldsymbol{K}_1) \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd})$, pick $\hat{\boldsymbol{r}} = (\hat{r}_1, \ldots, \hat{r}_{m_2}) \leftarrow \mathbb{Z}_N^{m_2}$, and compute

$$\widehat{\boldsymbol{K}} := \begin{cases} g_2^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 1 \\ g_2^{\boldsymbol{k}\left(\hat{\alpha}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 2 \\ g_2^{\boldsymbol{k}(\hat{\alpha}, 0, \boldsymbol{0})} & \text{if type} = 3 \ . \end{cases}$$

Set $\boldsymbol{K} := \boldsymbol{K}_1 \cdot \widehat{\boldsymbol{K}}$ and output a semi-functional key $\mathrm{sk} := (\mathrm{kInd}, \boldsymbol{K}) \in \mathbb{SK}_{\mathrm{kInd}}$.

SFEncaps $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)$: Generate $(\mathrm{K}, (\mathrm{cInd}, \boldsymbol{C}_1, \_)) \leftarrow \mathrm{Encaps}(\mathrm{cInd})$. Let $s \in \mathbb{Z}_N$ be the corresponding random exponent. Choose $\hat{s} \leftarrow \mathbb{Z}_N$, $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_N^{w_2}$ and compute $\widehat{\boldsymbol{C}} := g_2^{\boldsymbol{c}\left(\hat{s}, \hat{\boldsymbol{s}}, \hat{\boldsymbol{h}}\right)}$. Next, set $\boldsymbol{C} := \boldsymbol{C}_1 \cdot \widehat{\boldsymbol{C}}$, compute the hash value $t = \mathrm{H}(\mathrm{HInput}(\mathrm{cInd}, \boldsymbol{C}, \_))$ and $\widehat{C''} := (U_1^t \cdot V_1)^s \cdot \left(g_2^{\hat{u}_2 \cdot t} \cdot g_2^{\hat{v}_2}\right)^{\hat{s}}$. Output key K and $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'') \in \mathbb{C}_{\mathrm{cInd}}$.

Remark 3.2. Note, that differently from [3], we define the semi-functional elements $\hat{h}_1, \ldots, \hat{h}_n$, $\hat{u}_2$, and $\hat{v}_2$ as uniformly distributed elements in $\mathbb{Z}_{p_2}$ instead of $\mathbb{Z}_N$. All these elements are used only in the exponents of $g_2 \in \mathbb{G}_{p_2}$ and hence, by Chinese Remainder Theorem, we did not change the distributions of the user secret keys and the distributions of encapsulations. But, this simplifies argumentation on several places in the proofs.

Intuition behind the Consistency Checks. In this subsection we provide a high-level explanation of why the consistency checks render the decapsulation oracle useless to any ppt adversary. Our explanation leaves out many important details of the formal proof.

Assume that $\mathcal{A}$ queries the decapsulation oracle with $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'') \in \mathbb{C}_{\mathrm{cInd}}$ such that the group elements of CT contain only the $\mathbb{G}_{p_1}$ components. If CT passes (5), then by the verifiability property $\mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}(g_1, C_1)^{\mathrm{msk}}$. Next, our additional element C'' and the check in (3) guarantee that the $\mathbb{G}_{p_1}$ component of C_1 is of the form g_1^s and s is known to $\mathcal{A}$. Hence, the output of the decapsulation is $\mathrm{e}(g_1, C_1)^{\mathrm{msk}} = Y^s$. Since $\mathcal{A}$ knows Y and s anyway, this can be computed by $\mathcal{A}$ itself and the decapsulation oracle is useless for $\mathcal{A}$.

We still have to justify the assumption that the elements in CT contain only the $\mathbb{G}_{p_1}$ components. The checks in (4) guarantee that the elements of CT contain no $\mathbb{G}_{p_3}$ components. Then, the subgroup decision assumptions ensures that CT does not also contain $\mathbb{G}_{p_2}$ components.

The following lemma is important for the following security proof. It shows that, due to the consistency checks in (4) and in (5), there is no difference which *normal key* is used in the decapsulation algorithm. This lemma provides further intuition behind the consistency checks.

Lemma 3.1. *For every security parameter* λ*, every* $\mathrm{des} \in \Omega$*, every* $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \mathrm{Setup}\left[\left(1^\lambda, \mathrm{des}\right)\right]$*, every* $\mathrm{kInd} \in \mathbb{X}_\kappa$*, every* $\mathrm{sk}_1, \mathrm{sk}_2 \in [\mathrm{KeyGen}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})]$ *and every* $\mathrm{CT} \in \{0,1\}^*$ *it holds*

$$\Pr\left[\mathrm{K} : \mathrm{K} \leftarrow \mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_1, \mathrm{CT})\right] = \Pr\left[\mathrm{K} : \mathrm{K} \leftarrow \mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_2, \mathrm{CT})\right] \quad .$$

Proof. Let λ, $\mathrm{des} \in \Omega$, $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, $\mathrm{sk}_1, \mathrm{sk}_2 \in [\mathrm{KeyGen}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd})]$, and $\mathrm{CT} \in \{0,1\}^*$ be arbitrary, but fixed. We consider only the case that there exists an index $\mathrm{cInd} \in \mathbb{Y}_\kappa$ such that $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ and $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}) = 1$, since otherwise the decapsulation algorithm will output $\perp$ for both keys. Furthermore, let $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$, and $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$. We denote $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$, where $\boldsymbol{C} \in \mathbb{G}^{w_1}$ and $C'' \in \mathbb{G}$.

Both probability distributions are over the random choice of $\boldsymbol{E} \leftarrow \mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})$. The choice of $\boldsymbol{E} \in \mathbb{Z}_N^{m_1 \times w_1}$ depends on kInd, but is independent of the concrete secret key for kInd. Hence, every $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ is chosen with the same probability in both cases. Let $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ be arbitrary, but fixed. We claim, that independently of the concrete secret key, the result of the decapsulation algorithm using $\boldsymbol{E}$ will be the same. This will immediately prove the lemma.

It is important to notice, that for a fixed $\boldsymbol{E}$ the consistency checks are deterministic. In particular, Vrfy is a deterministic algorithm by definition. Hence, if one of the consistency checks fails, the output of the decapsulation algorithm will be $\perp$ independently of the concrete secret key. Hence, it remains to consider the case that $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$ passes all consistency checks.

Since the keys sk_1 and sk_2 are normal, there exist $\boldsymbol{r}_1, \boldsymbol{r}_2 \in \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}_{3,1}, \boldsymbol{R}_{3,2} \in \mathbb{G}_{p_3}^{m_1}$ such that $\mathrm{sk}_1 = \left(\mathrm{kInd}, \boldsymbol{K}_1 = g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}_1, \boldsymbol{h})} \cdot \boldsymbol{R}_{3,1}\right)$ and $\mathrm{sk}_2 = \left(\mathrm{kInd}, \boldsymbol{K}_2 = g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}_2, \boldsymbol{h})} \cdot \boldsymbol{R}_{3,2}\right)$, where $g_1^{\boldsymbol{h}} \in \mathrm{pp}_\kappa$. Hence, by construction of Decaps and since the elements in $\boldsymbol{C}$ do not have $\mathbb{G}_{p_3}$ components (due to the consistency check in (4)) it holds:

$$\begin{aligned}\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_1, \mathrm{CT}) &= \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}_1, \boldsymbol{h})} \cdot \boldsymbol{R}_{3,1}\right)^{\boldsymbol{E}}, \boldsymbol{C}\right) \\ &= \mathrm{e}\left(g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}_1, \boldsymbol{h}) \cdot \boldsymbol{E}}, \boldsymbol{C}\right) \quad ,\end{aligned}$$

and

$$\begin{aligned}\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_2, \mathrm{CT}) &= \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}_2, \boldsymbol{h})} \cdot \boldsymbol{R}_{3,2}\right)^{\boldsymbol{E}}, \boldsymbol{C}\right) \\ &= \mathrm{e}\left(g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}_2, \boldsymbol{h}) \cdot \boldsymbol{E}}, \boldsymbol{C}\right) \quad .\end{aligned}$$

Furthermore, CT passes the consistency check in (5). Hence, by the soundness property of Vrfy, for every $\alpha' \in \mathbb{Z}_N$, $\boldsymbol{r} \in \mathbb{Z}_N^{m_2}$ it holds:

$$\mathrm{e}\left(g_1^{\boldsymbol{k}(\alpha', \boldsymbol{r}, \boldsymbol{h}) \cdot \boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}(g_1, C_1)^{\alpha'} \quad .$$

We deduce that for a fixed $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ it holds

$$\mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_1, \mathrm{CT}) = \mathrm{e}(g_1, C_1)^{\mathrm{msk}} = \mathrm{Decaps}(\mathrm{pp}_\kappa, \mathrm{sk}_2, \mathrm{CT}) \quad .$$

This finally proves the lemma, since every $\boldsymbol{E}$ is chosen with the same probability in both probability distributions, as explained above. $\square$

Fig. 1. Proof Structure

$$\mathrm{G}_{\mathrm{Real}} \xrightarrow[\mathrm{CR}_{\mathcal{H}}]{} \mathrm{G}_{\mathrm{resH}} \xrightarrow[\mathrm{SD2}]{} \mathrm{G}_{\mathrm{resQ}} \xrightarrow[\mathrm{SD1}]{} \mathrm{G}'_0 \xrightarrow[\mathrm{Vrfy}]{} \mathrm{G}_{0,3} \cdots\rightarrow \mathrm{G}_{k-1,3} \xrightarrow[\mathrm{SD2}]{} \mathrm{G}_{k,1} \xrightarrow[\mathrm{CMH}]{} \mathrm{G}_{k,2} \xrightarrow[\mathrm{SD2}]{} \mathrm{G}_{k,3} \cdots\rightarrow \mathrm{G}_{q_1,3} \xrightarrow[\mathrm{SD2}]{} \mathrm{G}_{q_1+1} \xrightarrow[\mathrm{SMH}]{} \mathrm{G}_{q_1+2} \xrightarrow[\mathrm{SD2}]{} \mathrm{G}_{q_1+3} \xrightarrow[\mathrm{SD2}]{} \mathrm{G}'_{q_1+3} \xrightarrow[\mathrm{SD3}]{} \mathrm{G}_{\mathrm{Final}}$$

Extension of our construction. Our framework requires additional computational overhead during the computation of the hash value. Namely, a pairing is computed for every group element in the ciphertext. We can avoid this computation by hashing the original ciphertext. Formally, we only change the definition of the function HInput defined in (2). Then, our last reduction must be adapted in order to prove the security for this variant. The other reductions require only minor modifications. We decided to present the given less efficient construction in order to explicitly show which parts of the ciphertext are important for the well-formedness proofs, when the dual system encryption methodology is used to achieve CCA-secure schemes. We will present the formal proof for this variant in the next version.

4 Main Theorem and Extended Proof Technique

In this section we present our main theorem and explain the proof technique. We also state that all known pair encodings satisfy our verifiability property.

Theorem 4.1. *Let* Π *be the P-KEM from Section 3.3. Suppose that the subgroup decision assumptions from Section 2.4 are correct, the underlying pair encoding scheme* P *is selectively and co-selectively master key hiding, and the family of collision-resistant hash functions* $\mathcal{H}$ *is secure. Then,* Π *is fully CCA-secure with respect to Definition 2.4. Furthermore, for every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$*, there exists a negligible function* negl *and there exist ppt algorithms* $\mathcal{B}_1, \ldots, \mathcal{B}_6$ *with essentially the same running time as* $\mathcal{A}$ *such that for sufficiently large* λ *it holds*

$$\begin{aligned}\text{Adv-aP-KEM}^{\mathrm{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) \le\ & \mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{B}_1}(\lambda, \mathrm{des}) + \mathrm{Adv}^{\mathrm{SD1}}_{\mathcal{B}_2}(\lambda) + \mathrm{Adv}^{\mathrm{SD3}}_{\mathcal{B}_4}(\lambda) \\ & + (2q_1 + 4) \cdot \mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}_3}(\lambda) + \mathrm{Adv}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{B}_6}(\lambda, \mathrm{des}) \\ & + q_1 \cdot \mathrm{Adv}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{B}5}(\lambda, \mathrm{des}) + {q_{\mathrm{dec1}}}/{p_1} + \mathrm{negl}(\lambda) \quad ,\end{aligned}$$

where q_1 *is the number of keys that are corrupted in Phase I and* q_{dec1} *is the number of decapsulation queries in Phase I of experiment* $\mathrm{aP\text{-}KEM}^{\mathrm{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$.

For simplicity, we collected some negligible terms such as $1/p_1$ in $\mathrm{negl}(\lambda)$. It is important to notice that the number of decapsulation queries from Phase I only appears in the term q_{dec1}/p_1 and decreases the security guarantees only negligibly. Furthermore, compared to the CPA-secure framework of [3] we only loose the additional terms $\mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{B}_1}(\lambda, \mathrm{des})$ and $\mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}_3}(\lambda)$.

The structure for the proof of Theorem 4.1 is presented in Fig. 1. The nodes represent different probability experiments. In Table 1 the modifications between the probability experiments are defined (these will be explained in detail in the corresponding proofs). The first experiment is the target experiment $\mathrm{aP\text{-}KEM}^{\mathrm{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})$ from page 8 and the last experiment is constructed in such a way, that the advantage of every adversary is zero. The edges represent reduction steps and their labels the underlying security assumptions, except for the edge labeled with Vrfy. The corresponding proof is based on the verifiability property of the pair encoding scheme. In the proof we show that no ppt algorithm can distinguish between any pair of consecutive experiments. The formal proof of Theorem 4.1 is given in Appendix D. Here, we explain the main steps of the proof and the proof technique.

The structure of the proof for our CCA-secure construction is similar to the structure of the proof for the CPA-secure construction of [3]. Experiments $\mathrm{G}_{\mathrm{resH}}$, $\mathrm{G}_{\mathrm{resQ}}$, G'_0, and G'_{q_1+3} as well as the four reduction steps denoted by bold edges in Fig. 1 are new. The remaining experiments and reductions are from the original CPA-security proof from [3] and require only simple extensions.

Our first reduction $\mathrm{G}_{\mathrm{Real}} \to \mathrm{G}_{\mathrm{resH}}$ is based on the security of the family of collision-resistant hash functions. In the second reduction $\mathrm{G}_{\mathrm{resH}} \to \mathrm{G}_{\mathrm{resQ}}$ we separate failure events which enable us to find a non-trivial factor of N, which violates Assumption SD2 by Lemma 2.1. This reduction is an extension of the first reduction step from [3]. These two steps are of a technical nature. Our additional games G'_0 and G'_{q_1+3} and the corresponding new reductions $\mathrm{G}'_0 \to \mathrm{G}_{0,3}$ and $\mathrm{G}_{q_1+3} \to \mathrm{G}'_{q_1+3}$ are the most important

Table 1. The probability experiments from security proof.

$\mathrm{G}_{\mathrm{resH}}$:	Modify $^{\langle 10\rangle}$	Output is 0 if there is a collision for H
$\mathrm{G}_{\mathrm{resQ}}$:	Modify $^{\langle 10\rangle}$	Output 0 if $\mathcal{A}$ implicitly found a factor of N.
G'_0:	Modify $^{\langle 1\rangle}$ Modify $^{\langle 5\rangle}$	$\left(\mathrm{msk}, \mathrm{pp}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \leftarrow \mathrm{SFSetup}\left(1^{\lambda}, \mathrm{des}\right)$ $(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SFEncaps}\left(\mathrm{cInd}^*, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)$
$\mathrm{G}_{0,3}$:	Modify$^{\langle 4\rangle,\ \langle 9\rangle}$ Change	$\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i\right)$, $\mathrm{Decaps}\left(\mathrm{sk}'_i, \mathrm{CT}\right)$ Generate keys in Open oracle.
$\mathrm{G}_{k,1}$:	Modify $^{\langle 3\rangle}$	$\hat{\alpha}_j \leftarrow \mathbb{Z}_N,$ $\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}_j, g_2, \_\right) & \text{if } j < k \\ \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 1, \_, g_2, \hat{\boldsymbol{h}}\right) & \text{if } j = k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
$\mathrm{G}_{k,2}$:	Modify $^{\langle 3\rangle}$	$\hat{\alpha}_j \leftarrow \mathbb{Z}_N,$ $\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}_j, g_2, \_\right) & \text{if } j < k \\ \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 2, \hat{\alpha}_j, g_2, \hat{\boldsymbol{h}}\right) & \text{if } j = k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
$\mathrm{G}_{k,3}$:	Modify $^{\langle 3\rangle}$	$\hat{\alpha}_j \leftarrow \mathbb{Z}_N,$ $\mathrm{sk}_j \leftarrow \begin{cases} \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}_j, g_2, \_\right) & \text{if } j \leq k \\ \mathrm{KeyGen}\left(\mathrm{msk}, \mathrm{kInd}\right) & \text{if } j > k \end{cases}$
G_{q_1+1}:	Modify $^{\langle 8\rangle}$	$\mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 1, \_, g_2, \hat{\boldsymbol{h}}\right)$
G_{q_1+2}:	Insert Modify $^{\langle 8\rangle}$	$\hat{\alpha} \leftarrow \mathbb{Z}_N$ at the beginning of Phase II $\mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 2, \hat{\alpha}, g_2, \hat{\boldsymbol{h}}\right)$
G_{q_1+3}:	Modify $^{\langle 8\rangle}$	$\mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}, g_2, \_\right)$
G'_{q_1+3}:	Insert Modify $^{\langle 4\rangle}, ^{\langle 9\rangle}$	$X_2 \leftarrow \mathbb{G}_{p_2}$ in the Setup phase Check consistency, return $\mathrm{e}\left(g_1^{\mathrm{msk}} \cdot X_2, C_1\right)$
$\mathrm{G}_{\mathrm{Final}}$:	Modify $^{\langle 6\rangle}$	$\mathrm{K}^* \leftarrow \mathbb{G}_{\mathrm{T}}$

parts of the CCA-security proof and enable us to deal with decapsulation queries in an elegant way. The major modification in $\mathrm{G}_{0,3}$ is that the decapsulation queries are answered using separately generated *normal keys* which we denote by sk'_i. We do not change these keys to semi-functional in the following games. In particular, using consistency check (5) we show that for every (unconditional) $\mathcal{A}$, experiments G'_0 and $\mathrm{G}_{0,3}$ are indistinguishable. The next important observation is that in all reductions between $\mathrm{G}_{0,3}$ and G_{q_1+3}, the master secret key is known to the reduction algorithm. Hence, the normal keys for the decapsulation queries can be generated by the key generation algorithm. The final challenge is to answer decapsulation queries without the user secret keys in the last experiment $\mathrm{G}_{\mathrm{Final}}$. Experiment G'_{q_1+3} and the corresponding new reduction step $\mathrm{G}_{q_1+3} \rightarrow \mathrm{G}'_{q_1+3}$ allow us to deal with this problem. In the proof of this reduction step we use our additional group element from the encapsulation in order to answer the decapsulation queries. To prove that this modification can not be noticed, again the consistency checks are crucial (see the proof of Lemma D.17).

Verifiability of pair encoding schemes. In this paragraph we explain how to construct verification algorithms for pair encoding schemes according to Definition 3.3. Together with our framework, this provides new, fully CCA-secure PE schemes for various predicates. Among these are an IBE scheme, the scheme for regular languages and its dual, new and reviewed key-policy and ciphertext-policy attribute-base schemes, spatial and negated spatial encryption, key-policy over doubly spatial encryption, as well as dual-policy attribute-based schemes. All (nineteen) pair encoding schemes from [3, 5] satisfy the verifiability property according to Definition 3.3. Almost all these encoding schemes are *regular*. The following theorem leads to verification algorithms for all these schemes. We refer to Appendix E for the definition of regular pair encodings and for the constructive proof of this theorem.

Theorem 4.2. *Suppose $\mathcal{R}_{\Omega,\Sigma}$ is a domain-transferable predicate family and* P *is a regular pair encoding scheme for $\mathcal{R}_{\Omega,\Sigma}$. Then,* P *satisfies the verifiability property according to Definition 3.3.*

Two ciphertext-policy attribute-based encryption schemes from [3] achieved using dual scheme conversion are not regular. These schemes were improved in [5] and the resulting schemes are regular. Anyway, verification algorithms can be constructed using a slightly adapted technique also for these schemes.

5 Comparison with Generic Constructions and Conclusion

In this section we compare the efficiency of our construction to the efficiency of generic constructions for fully CCA-secure PEs from [23, 24]. On the one hand we look at the size of public parameters, user secret keys and ciphertexts. On the other hand we look at the efficiency of the encapsulation (encryption) and the decapsulation (decryption) algorithms.

All generic transformations from above use one-time signature schemes as a building block and integrate the verification key vk into the ciphertexts. This results in non-trivial extensions of public parameters, user secret keys and ciphertexts. For example, keys and ciphertexts of PE for the dual of regular languages are extended by $6 \cdot |\mathrm{vk}|$ and by $2 \cdot |\mathrm{vk}|$ group elements. In contrast to this, we only add two group elements to the public parameters and a single group element to the ciphertext independently of the predicate. Hence, with respect to the size of public parameters, secret keys, and ciphertexts our construction is more efficient.

Considering the efficiency of the encapsulation and the decapsulation, we further need to distinguish two types of generic transformations of CPA-secure schemes into CCA-secure schemes: schemes based on verifiability, and schemes based on key delegation. CCA-secure attribute-based schemes achieved from key delegation [23] require derandomization and delegation of the user secret keys in every decryption. Depending on the predicate, on kInd and on cInd this can be more efficient or more costly compared to the schemes achieved using our construction. Generic constructions based on verifiability require a verification algorithm which ensures that decryption of a ciphertext under every secret key for kInd and every secret key corresponding to vk will be the same. In our construction we require that decapsulation using every secret key for kInd will be the same. Hence, schemes from generic constructions have to check in addition those parts of the ciphertext, that correspond to the verification key included in the ciphertext ($2 \cdot |\mathrm{vk}|$ group elements in the example from above). This results in more costly verification algorithms compared to ours. Furthermore, these additional elements have to be computed in the encryption algorithm together with the one-time signature, whereas we only use a hash function and have to compute a single group element in addition.

Summarizing, we presented a semi-generic framework to construct fully CCA-secure PEs in composite-order groups from any verifiable pair encoding schemes including regular pair encoding schemes. From this point of view our framework is as generic as the underlying CPA-secure framework of [3]. Our security proofs are based on a small but significant modification of the dual system encryption methodology, i.e. we do not change decryption keys to semi-functional. This results in a reduction of CCA-security to the security of pair encodings which is almost as tight as the reduction of CPA-security to the security of pair encodings given by Attrapadung [3].

References

[1] Agrawal, S., Chase, M.: A study of pair encodings: Predicate encryption in prime order groups. Cryptology ePrint Archive, Report 2015/413

[2] Attrapadung, N.: Dual system encryption framework in prime-order groups. Cryptology ePrint Archive, Report 2015/390

[3] Attrapadung, N.: Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In: Advances in Cryptology - EUROCRYPT'14. LNCS, vol. 8441, pp. 557–577 (2014)

[4] Attrapadung, N., Libert, B.: Functional encryption for public-attribute inner products: Achieving constant-size ciphertexts with adaptive security or support for negation. Journal of Mathematical Cryptology 5(2), 115–158 (2012)

[5] Attrapadung, N., Yamada, S.: Duality in ABE: converting attribute based encryption for dual predicate and dual policy via computational encodings. In: Topics in Cryptology - CT-RSA'15. LNCS, vol. 9048, pp. 87–105 (2015)

[6] Bentahar, K., Farshim, P., Malone-Lee, J., Smart, N.P.: Generic constructions of identity-based and certificateless kems. Journal of Cryptology 21(2), 178–199 (2008)

[7] Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM Journal on Computing 36(5), 1301–1328 (2007)
[8] Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Theory of Cryptography - TCC'11. LNCS, vol. 6597, pp. 253–273 (2011)
[9] Chen, J., Gay, R., Wee, H.: Improved dual system ABE in prime-order groups via predicate encodings. In: Advances in Cryptology - EUROCRYPT'15 Part II. LNCS, vol. 9057, pp. 595–624 (2015)
[10] Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Advances in Cryptology - EUROCRYPT'02. LNCS, vol. 2332, pp. 45–64 (2002)
[11] Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM Journal on Computing 33(1), 167–226 (2003)
[12] Hamburg, M.: Spatial encryption. Cryptology ePrint Archive, Report 2011/389
[13] Kiltz, E., Galindo, D.: Direct chosen-ciphertext secure identity-based key encapsulation without random oracles. Theoretical Computer Science 410(47-49), 5093–5111 (2009)
[14] Kiltz, E., Vahlis, Y.: CCA2 secure IBE: standard model efficiency through authenticated symmetric encryption. In: Topics in Cryptology - CT-RSA'08. LNCS, vol. 4964, pp. 221–238 (2008)
[15] Lewko, A.B., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: Advances in Cryptology - EUROCRYPT'10. LNCS, vol. 6110, pp. 62–91 (2010)
[16] Lewko, A.B., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: Theory of Cryptography - TCC'10. LNCS, vol. 5978, pp. 455–479 (2010)
[17] Lewko, A.B., Waters, B.: Unbounded HIBE and attribute-based encryption. In: Advances in Cryptology - EUROCRYPT'11. LNCS, vol. 6632, pp. 547–567 (2011)
[18] Okamoto, T., Takashima, K.: Fully secure functional encryption with general relations from the decisional linear assumption. In: Advances in Cryptology - CRYPTO'10. LNCS, vol. 6223, pp. 191–208 (2010)
[19] Phan, D.H., Pointcheval, D., Shahandashti, S.F., Strefler, M.: Adaptive CCA broadcast encryption with constant-size secret keys and ciphertexts. International Journal of Information Security 12(4), 251–265 (2013)
[20] Shoup, V.: Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332 (2004), `https://eprint.iacr.org/2004/332`
[21] Waters, B.: Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In: Advances in Cryptology - CRYPTO'09. LNCS, vol. 5677, pp. 619–636 (2009)
[22] Wee, H.: Dual system encryption via predicate encodings. In: Theory of Cryptography - TCC'14. LNCS, vol. 8349, pp. 616–637 (2014)
[23] Yamada, S., Attrapadung, N., Hanaoka, G., Kunihiro, N.: Generic constructions for chosen-ciphertext secure attribute based encryption. In: Public Key Cryptography - PKC'11. LNCS, vol. 6571, pp. 71–89 (2011)
[24] Yamada, S., Attrapadung, N., Santoso, B., Schuldt, J.C.N., Hanaoka, G., Kunihiro, N.: Verifiable predicate encryption and applications to CCA security and anonymous predicate authentication. In: Public Key Cryptography - PKC'12. LNCS, vol. 7293, pp. 243–261 (2012)

A Security Notions of Pair Encodings

In this section we recall the computational security notions of pair encoding schemes presented in [3]. Suppose $\mathcal{G}$ is a group generation algorithm and P is a pair encoding scheme for domain-transferable predicate family $\mathcal{R}_{\Omega,\Sigma}$. First define the following generic probability experiment between challenger $\mathcal{C}$ and adversary $\mathcal{A}$, which is parametrized with Type $\in \{\text{SMH}, \text{CMH}\}$ and $\nu \in \{0,1\}$.

$\mathbf{Exp}^{\text{Type}}_{\text{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \text{des})$:

Setup : $\mathcal{C}$ chooses $\mathbb{GD} \leftarrow \mathcal{G}(1^\lambda)$, sets $\kappa := (\text{des}, N)$ and $n := \text{Param}(\kappa)$. $\mathcal{C}$ picks $g_1 \leftarrow \mathbb{G}_{p_1}$, $g_2 \leftarrow \mathbb{G}_{p_2}$, $g_3 \leftarrow \mathbb{G}_{p_3}$, $\hat{\alpha} \leftarrow \mathbb{Z}_N$, $\hat{\boldsymbol{h}} \leftarrow \mathbb{Z}_N^n$, and simulates adversary $\mathcal{A}$ on $(\text{des}, \mathbb{GD}_N, g_1, g_2, g_3)$.
Phase I : $\mathcal{A}$ is allowed to query oracle $\mathcal{O}^1_{\text{Type},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\cdot)$.
Phase II : $\mathcal{A}$ is allowed to query oracle $\mathcal{O}^2_{\text{Type},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\cdot)$.
Guess : $\mathcal{A}$ outputs a guess $\nu' \in \{0,1\}$, which is the output of the experiment.

Based on this generic experiment we define two following security experiments for the corresponding security notions of pair encoding schemes.

$\mathbf{Exp}^{\text{SMH}}_{\text{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \text{des})$ is an instantiation of $\text{Exp}^{\text{Type}}_{\text{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \text{des})$ with:

$\mathcal{O}^1_{\text{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\text{cInd}^*)$ for $\text{cInd}^* \in \mathbb{Y}_\kappa$: Can be queried only once. $\mathcal{C}$ computes $(\boldsymbol{c}, w_2) := \text{Enc2}(\kappa, \text{cInd}^*)$, picks $\hat{s} \leftarrow \mathbb{Z}_N$, $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_N^{w_2}$ and returns $\widehat{\boldsymbol{C}} := g_2^{\boldsymbol{c}(\hat{s},\hat{\boldsymbol{s}},\hat{\boldsymbol{h}})}$.
$\mathcal{O}^2_{\text{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\text{kInd})$ for $\text{kInd} \in \mathbb{X}_\kappa$: Can be queried polynomially many times. Challenger $\mathcal{C}$ returns $\perp$ if $\text{R}_{p_2}(f_1(\text{kInd}), f_2(\text{cInd}^*)) = 1$. Otherwise, $\mathcal{C}$ computes $(\boldsymbol{k}, m_2) := \text{Enc1}(\kappa, \text{kInd})$, picks $\hat{\boldsymbol{r}} \leftarrow \mathbb{Z}_N^{m_2}$ and returns $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(0,\hat{\boldsymbol{r}}_i,\hat{\boldsymbol{h}})}$ if $\nu = 0$ and $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(\hat{\alpha},\hat{\boldsymbol{r}}_i,\hat{\boldsymbol{h}})}$ if $\nu = 1$.

The advantage of $\mathcal{A}$ in this experiment is defined as:

$$\text{Adv}^{\text{SMH}}_{\text{P},\mathcal{A}}(\lambda, \text{des}) := \left|\Pr\left[\text{Exp}^{\text{SMH}}_{\text{P},\mathcal{G},0,\mathcal{A}}(\lambda, \text{des}) = 1\right] - \Pr\left[\text{Exp}^{\text{SMH}}_{\text{P},\mathcal{G},1,\mathcal{A}}(\lambda, \text{des}) = 1\right]\right| .$$

Definition A.1. *Pair encoding* P *is called* ***selectively master key hiding*** *with respect to* $\mathcal{G}$ *if for all* $\text{des} \in \Omega$*, all* λ *and all ppt (in* λ*) adversaries* $\mathcal{A}$*, the function* $\text{Adv}^{\text{SMH}}_{\text{P},\mathcal{A}}(\lambda, \text{des})$ *is negligible in* λ*.*

$\mathbf{Exp}^{\text{CMH}}_{\text{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \text{des})$ is an instantiation of $\text{Exp}^{\text{Type}}_{\text{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \text{des})$ with:

$\mathcal{O}^1_{\text{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\text{kInd})$ for $\text{kInd} \in \mathbb{X}_\kappa$: Can be queried only once. $\mathcal{C}$ computes $(\boldsymbol{k}, m_2) := \text{Enc1}(\kappa, \text{kInd})$, $\hat{\boldsymbol{r}} \leftarrow \mathbb{Z}_{p_2}^{m_2}$ and returns $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(0,\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})}$ if $\nu = 0$ and $\widehat{\boldsymbol{K}} := g_2^{\boldsymbol{k}(\hat{\alpha},\hat{\boldsymbol{r}},\hat{\boldsymbol{h}})}$ if $\nu = 1$.
$\mathcal{O}^2_{\text{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\text{cInd}^*)$ for $\text{cInd}^* \in \mathbb{Y}_\kappa$: The oracle can be queried only once. Challenger $\mathcal{C}$ returns $\perp$ if $\text{R}_{p_2}(f_1(\text{kInd}), f_2(\text{cInd}^*)) = 1$. Otherwise, $\mathcal{C}$ computes $(\boldsymbol{c}, w_2) := \text{Enc2}(\kappa, \text{cInd}^*)$, picks $\hat{s} \leftarrow \mathbb{Z}_N$ and $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_N^{w_2}$ and returns $\widehat{\boldsymbol{C}} := g_2^{\boldsymbol{c}(\hat{s},\hat{\boldsymbol{s}},\hat{\boldsymbol{h}})}$.

The advantage of $\mathcal{A}$ in this experiment is defined as:

$$\text{Adv}^{\text{CMH}}_{\text{P},\mathcal{A}}(\lambda, \text{des}) := \left|\Pr\left[\text{Exp}^{\text{CMH}}_{\text{P},\mathcal{G},0,\mathcal{A}}(\lambda, \text{des}) = 1\right] - \Pr\left[\text{Exp}^{\text{CMH}}_{\text{P},\mathcal{G},1,\mathcal{A}}(\lambda, \text{des}) = 1\right]\right| .$$

Definition A.2. *Pair encoding* P *is called* ***co-selectively master key hiding*** *with respect to* $\mathcal{G}$ *if for all* $\text{des} \in \Omega$*, all* λ *and all ppt (in* λ*) adversaries* $\mathcal{A}$*, the function* $\text{Adv}^{\text{CMH}}_{\text{P},\mathcal{A}}(\lambda, \text{des})$ *is negligible in* λ*.*

B Families of Collision-Resistant Hash Function

Let $\mathcal{G}$ be a group generation algorithm and $\mathcal{R}_{\Omega,\Sigma}$ be a predicate family with $\Sigma \subseteq \mathbb{N}$. In our constructions we will hash elements from $\mathbb{Y}_\kappa$ together with a restricted number of group elements of $\mathbb{G}_T$. Hence, we require a collision resistant hash function, which will be formalized as in [11]. Let λ be a security parameter, $\text{des} \in \Omega$.

Definition B.1. *A* ***family of collision-resistant hash functions*** $\mathcal{H}$ *associated with* $\mathcal{G}$ *and* $\mathcal{R}_{\Omega,\mathbb{N}}$ *is specified by:*

- *A family of key spaces* $\mathcal{S} = \{S_{\lambda,\mathbb{GD}_N,\mathrm{des}}\}$ *indexed by security parameter* λ*, restricted group description* $\mathbb{GD}_N$ *of* $\mathbb{GD} \in \left[\mathcal{G}\left(1^\lambda\right)\right]$*, and* $\mathrm{des} \in \Omega$*. Each* $S_{\lambda,\mathbb{GD}_N,\mathrm{des}}$ *is a probability space and there must be a ppt algorithm* Sample*, which given* 1^λ*,* $\mathbb{GD}_N$*, and* des *outputs a key* s *according to* $S_{\lambda,\mathbb{GD}_N,\mathrm{des}}$*. We write* $s \leftarrow \mathrm{Sample}\left(1^\lambda, \mathbb{GD}_N, \mathrm{des}\right)$*.*
- *A family of efficiently computable functions*

$$\left\{\mathrm{H}_{\lambda,\mathbb{GD}_N,\mathrm{des},s} : \mathbb{Y}_{\mathrm{des},N} \times \left(\mathbb{G}_\mathrm{T}\right)^{\leq m_{\mathrm{des},N}} \mapsto \mathbb{Z}_N\right\}$$

indexed by security parameter λ*, restricted group description* $\mathbb{GD}_N$ *of* $\mathbb{GD} \in \left[\mathcal{G}\left(1^\lambda\right)\right]$*,* $\mathrm{des} \in \Omega$*, and* $s \in S_{\lambda,\mathbb{GD}_N,\mathrm{des}}$*. Furthermore,* $m_{\mathrm{des},N} \in \mathbb{N}$ *only depends on* des*,* N *and the pair encoding scheme. Namely,*

$$m_{\mathrm{des},N} = \max\left(\left\{w_1 \in \mathbb{N} : (\boldsymbol{c}, w_2) = \mathrm{Enc2}\left(\left(\mathrm{des}, N\right), \mathrm{cInd}\right), w_1 = |\boldsymbol{c}|\right\}_{\mathrm{cInd} \in \mathbb{Y}_{\mathrm{des},N}}\right) \quad .$$

The security property for the collision-resistant hash functions is defined through the following probabilistic experiment.

$$\begin{aligned}\mathbf{CR}_{\mathcal{H},\mathcal{A}}\left(\lambda, \mathrm{des}\right): \quad & \mathbb{GD} \leftarrow \mathcal{G}\left(1^\lambda\right), s \leftarrow \mathrm{Sample}\left(1^\lambda, \mathbb{GD}_N, \mathrm{des}\right) \quad , \\ & \left(x_1, x_2\right) \leftarrow \mathcal{A}\left(1^\lambda, \mathbb{GD}, \mathrm{des}, s\right) \quad . \\ & \text{The output is 1 if and only if } x_1 \neq x_2 \qquad \text{and} \\ & \mathrm{H}_{\lambda,\mathbb{GD}_N,\mathrm{des},s}\left(x_1\right) = \mathrm{H}_{\lambda,\mathbb{GD}_N,\mathrm{des},s}\left(x_2\right) \pmod{N} \quad .\end{aligned}$$

We particularly note that $\mathcal{A}$ is given $\mathbb{GD}$ and not only $\mathbb{GD}_N$ in the defined experiment.

The advantage of $\mathcal{A}$ in experiment $\mathrm{CR}_{\mathcal{H},\mathcal{A}}\left(\lambda, \mathrm{des}\right)$ is defined as

$$\mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{A}}\left(\lambda, \mathrm{des}\right) := \Pr\left[\mathrm{CR}_{\mathcal{H},\mathcal{A}}\left(\lambda, \mathrm{des}\right) = 1\right] \quad .$$

Definition B.2. *A family of collision-resistant hash functions* $\mathcal{H}$ *is secure, if for every ppt algorithm* $\mathcal{A}$ *there exists a negligible function* $\mathrm{negl}\left(\lambda\right)$ *such that* $\mathrm{Adv}^{\mathrm{CR}}_{\mathcal{H},\mathcal{A}}\left(\lambda, \mathrm{des}\right) \leq \mathrm{negl}\left(\lambda\right)$*.*

For simplicity, if 1^λ, $\mathbb{GD}_N$, and des are fixed and obvious from the context, we will write $\mathrm{H} \leftarrow \mathcal{H}_\kappa$ instead of $s \leftarrow \mathrm{Sample}\left(1^\lambda, \mathbb{GD}_N, \mathrm{des}\right)$ and $\mathrm{H} := \mathrm{H}_{\lambda,\mathbb{GD}_N,\mathrm{des},s}$. We will use the formalized experiment for the reduction step based on the security property of the hash family $\mathcal{H}$.

C Hybrid Construction of Predicate Encryption Schemes

In this section we show that a predicate key-encapsulation mechanisms (P-KEMs) combined with appropriate symmetric schemes lead to predicate encryption schemes (PEs) with unrestricted message space. We start by recalling common definition of data encapsulation mechanisms and key derivation functions as well as appropriate security definitions for these primitives. Then, we present the hybrid construction for PE schemes and prove the security of this construction. The proof is similar to those from PKE [11] and IBE [6] settings.

C.1 Data Encapsulation Mechanisms (DEMs)

In this subsection we recall the definition of data encapsulation mechanism, which is also called one-time symmetric-key encryption (cf. [11]).

Definition C.1. *Let* $\mathrm{KLen}\left(\lambda\right)$ *be a polynomial. A* ***data encapsulation mechanism*** Π *for the message space* $\mathcal{M} = \{0,1\}^*$ *and with key length* $\mathrm{KLen}\left(\lambda\right)$ *consists of two* deterministic *polynomial time algorithms:*

Enc $\left(1^\lambda, \mathrm{symk}, m\right) =: \mathrm{C}$: *The encryption algorithm takes as input a security parameter* λ*, a key* $\mathrm{symk} \in \{0,1\}^{\mathrm{KLen}(\lambda)}$*, and a message* $m \in \mathcal{M}$*. It outputs a ciphertext* $\mathrm{C} \in \{0,1\}^*$*.*

Dec $\left(1^\lambda, \text{symk}, \text{C}\right) =: m$: *The decryption algorithm takes as input a security parameter* λ, *a key* symk $\in \{0,1\}^{\text{KLen}(\lambda)}$, *and a ciphertext* $\text{C} \in \{0,1\}^*$. *It outputs a message* $m \in \mathcal{M}$ *or an error symbol* $\perp \notin \mathcal{M}$.

Correctness: *For every security parameter* λ, *every* symk $\in \{0,1\}^{\text{KLen}(\lambda)}$, *and every* $m \in \mathcal{M}$ *it must hold*

$$\text{Dec}\left(1^\lambda, \text{symk}, \text{Enc}\left(1^\lambda, \text{symk}, m\right)\right) = m \ .$$

Next we recall the chosen-ciphertext security definition for DEMs.

$\mathbf{DEM}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda)$:

Challenge : Adversary $\mathcal{A}\left(1^\lambda\right)$ outputs two messages $m_0, m_1 \in \mathcal{M}$ of the same length. Challenger $\mathcal{C}$ picks a key symk $\leftarrow \{0,1\}^{\text{KLen}(\lambda)}$ and a bit $b \leftarrow \{0,1\}$. It returns $\text{C}^* := \text{Enc}\left(1^\lambda, \text{symk}, m_b\right)$ to $\mathcal{A}$.

Phase II : $\mathcal{A}$ has access to the decryption oracle $\text{Dec}\left(1^\lambda, \text{symk}, \cdot\right)$ for any $\text{C} \in \{0,1\}^*$ under the restriction that $\text{C} \neq \text{C}^*$.

Guess : $\mathcal{A}$ outputs a bit b'. If one of the restrictions is violated, the output of the experiment is 0. The output of the experiment is 1 iff $b' = b$.

The advantage of $\mathcal{A}$ in experiment $\text{DEM}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda)$ is defined as

$$\text{Adv-DEM}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda) := \Pr\left[\text{DEM}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda) = 1\right] - 1/2 \ .$$

Definition C.2. *A data encapsulation mechanism* Π *is called* ***CCA-secure*** *if for every ppt adversary* $\mathcal{A}$ *the function* $\text{Adv-DEM}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda)$ *is negligible.*

C.2 Key Derivation Functions

In this subsection we recall the notion of key derivation functions. These definitions are simplified compared to the formalization in [11].

Definition C.3. *Let* $\mathcal{K} = \{\mathbb{K}_\lambda\}$ *be a family of key spaces indexed by security parameter* λ. *A* ***key derivation function family*** $\mathcal{KDF} = (\text{Sample}, \text{KDF})$ *for* $\mathcal{K}$ *with output length* $\text{OutLength}(\lambda)$ *for some polynomial* $\text{OutLength}(\lambda)$ *consists of two ppt algorithms:*

Sample $\left(1^\lambda\right) \to \text{dk}$: *The probabilistic sampling algorithm takes as input a security parameter* λ *and outputs a derivation key* dk.

KDF $\left(1^\lambda, \text{dk}, \text{K}\right) =: \text{symk}$: *The deterministic evaluation algorithm takes as input a derivation key* dk, *a security parameter* λ, *and a source key* $\text{K} \in \mathbb{K}_\lambda$. *It outputs a symmetric key* symk $\in \{0,1\}^{\text{OutLength}(\lambda)}$.

We define the security property of $\mathcal{KDF}$ through the following probability distributions.

$$\mathbf{KDF}_{\mathcal{KDF}}(\lambda): \quad \text{dk} \leftarrow \text{Sample}\left(1^\lambda\right), \quad \text{K} \leftarrow \mathbb{K}_\lambda, \quad \text{symk}_0 := \text{KDF}\left(1^\lambda, \text{dk}, \text{K}\right), \quad \text{symk}_1 \leftarrow \{0,1\}^{\text{OutLength}(\lambda)} \ .$$

The advantage of an adversary $\mathcal{A}$ against $\mathcal{KDF}$ is defined as

$$\text{Adv-KDF}_{\mathcal{KDF},\mathcal{A}}(\lambda) := \left|\Pr\left[\mathcal{A}\left(1^\lambda, \text{dk}, \text{symk}_0\right) = 1\right] - \Pr\left[\mathcal{A}\left(1^\lambda, \text{dk}, \text{symk}_1\right) = 1\right]\right| \ .$$

Definition C.4. *A family of key derivation functions* $\mathcal{KDF}$ *is secure, if for every ppt algorithm* $\mathcal{A}$ *the function* $\text{Adv-KDF}_{\mathcal{KDF},\mathcal{A}}(\lambda)$ *is negligible.*

C.3 Predicate encryption schemes

For the sake of completeness we present a formal definition of predicate based encryption schemes, which is similar to the Definition 2.3 of P-KEMs.

Definition C.5. *A* ***predicate encryption*** Π_{pe} *for predicate family* $\mathcal{R}_{\Omega,\Sigma}$ *and message space* $\mathcal{M}$ *consists of four ppt algorithms:*

Setup $\left(1^\lambda, \mathrm{des}\right) \to (\mathrm{msk}, \mathrm{pp}_\kappa)$ *: takes as input security parameter* λ*,* $\mathrm{des} \in \Omega$*, and outputs a master secret key and public parameters. The algorithm determines among other elements* $\mathrm{dom} \in \Sigma$ *and the relation index* $\kappa = (\mathrm{des}, \mathrm{dom}) \in \Omega \times \Sigma$ *is (implicitly) included in* pp_κ*.*

KeyGen $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right) \to \mathrm{sk}$ *: takes as input the master secret key* msk *and a key index* $\mathrm{kInd} \in \mathbb{X}_\kappa$*. It generates a user secret key* sk *for* kInd*.*

Enc $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\right) \to ct$ *: takes as input a ciphertext index* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *and a message* $m \in \mathcal{M}$*. It outputs a ciphertext* ct*.*

Dec $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, ct\right) \to m$ *: takes as input a secret key* sk *and a ciphertext* ct*. It outputs a message* $m \in \mathcal{M}$ *or an error symbol* $\perp \notin \mathcal{M}$*.*

Correctness: *For every security parameter* λ*, every* $\mathrm{des} \in \Omega$*, every* $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$*, every* $\mathrm{kInd} \in \mathbb{X}_\kappa$ *and* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *which satisfy* $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$*, every* $\mathrm{sk} \in \left[\mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right)\right]$*, every* $m \in \mathcal{M}$ *and every* $ct \in \left[\mathrm{Enc}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\right)\right]$ *it must hold*

$$\Pr\left[\mathrm{Dec}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, ct\right) = m\right] = 1 \ .$$

C.4 Hybrid Construction

In this subsection we put the primitives together and present the hybrid constructions for predicate encryption schemes.

Suppose $l(\lambda)$ is a polynomial. Let $\Pi_{\mathrm{DEM}} = \left(\mathrm{Enc}', \mathrm{Dec}'\right)$ be a data encapsulation mechanism for message space $\mathcal{M} = \{0,1\}^*$ and with key length $l(\lambda)$. Let $\Pi_{\mathrm{KEM}} = \left(\mathrm{Setup}', \mathrm{KeyGen}', \mathrm{Encaps}', \mathrm{Decaps}'\right)$ be a predicate key encapsulation mechanism for $\mathcal{R}_{\Omega,\Sigma}$ and key space family $\mathcal{K} = \{\mathbb{K}_\lambda\}$. At last, let $\mathcal{KDF} = (\mathrm{Sample}, \mathrm{KDF})$ be a key derivation function family for $\mathcal{K}$ and with output length $l(\lambda)$.

The hybrid predicate encryption Π_{hyb} for $\mathcal{R}_{\Omega,\Sigma}$ and $\mathcal{M}$ is as follows:

Setup $\left(1^\lambda, \mathrm{des}\right) \to (\mathrm{msk}, \mathrm{pp}_\kappa)$ for $\mathrm{des} \in \Omega$: Generate $\left(\mathrm{msk}', \mathrm{pp}'_\kappa\right) \leftarrow \mathrm{Setup}'\left(1^\lambda, \mathrm{des}\right)$ as well as $\mathrm{dk} \leftarrow \mathrm{Sample}\left(1^\lambda\right)$. Return $\mathrm{msk} = \mathrm{msk}'$ and $\mathrm{pp}_\kappa := (\mathrm{pp}'_\kappa, \mathrm{dk})$.

KeyGen $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right) \to \mathrm{sk}$ for $\mathrm{pp}_\kappa = (\mathrm{pp}'_\kappa, \mathrm{dk})$ and $\mathrm{kInd} \in \mathbb{X}_\kappa$: Generate and return a user secret key $\mathrm{sk} \leftarrow \mathrm{KeyGen}'\left(1^\lambda, \mathrm{pp}'_\kappa, \mathrm{msk}', \mathrm{kInd}\right)$, where $\mathrm{msk}' = \mathrm{msk}$.

Enc $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\right) \to ct$ for $\mathrm{pp}_\kappa = (\mathrm{pp}'_\kappa, \mathrm{dk})$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, and $m \in \mathcal{M}$: Generate an encapsulation $(\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{Encaps}'\left(1^\lambda, \mathrm{pp}'_\kappa, \mathrm{cInd}\right)$, compute $\mathrm{symk} := \mathrm{KDF}\left(1^\lambda, \mathrm{dk}, \mathrm{K}\right)$ and $\mathrm{C} := \mathrm{Enc}'\left(1^\lambda, \mathrm{symk}, m\right)$, and return $ct := (\mathrm{CT}, \mathrm{C})$.

Dec $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, ct\right)$ for $\mathrm{pp}_\kappa = (\mathrm{pp}'_\kappa, \mathrm{dk})$ and $ct = (\mathrm{CT}, \mathrm{C})$: Compute $\mathrm{K} \leftarrow \mathrm{Decaps}'\left(1^\lambda, \mathrm{pp}'_\kappa, \mathrm{sk}, \mathrm{CT}\right)$, $\mathrm{symk} := \mathrm{KDF}\left(1^\lambda, \mathrm{dk}, \mathrm{K}\right)$, and output $\mathrm{Dec}'\left(1^\lambda, \mathrm{symk}, \mathrm{C}\right)$.

Proof. (Correctness) Let λ, $\mathrm{des} \in \Omega$, $(\mathrm{msk}, \mathrm{pp}_\kappa) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ which satisfy $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, $\mathrm{sk} \in \left[\mathrm{KeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}\right)\right]$, $m \in \mathcal{M}$ and $ct \in \left[\mathrm{Enc}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, m\right)\right]$ be arbitrary, but fixed.

By construction of Π_{hyb} it holds $\mathrm{pp}_\kappa = (\mathrm{pp}'_\kappa, \mathrm{dk})$, where $\left(\mathrm{msk}', \mathrm{pp}'_\kappa\right) \in \left[\mathrm{Setup}'\left(1^\lambda, \mathrm{des}\right)\right]$ as well as $\mathrm{dk} \in \left[\mathrm{Sample}\left(1^\lambda\right)\right]$. Furthermore, for the key sk it holds $\mathrm{sk} \in \left[\mathrm{KeyGen}'\left(1^\lambda, \mathrm{pp}'_\kappa, \mathrm{msk}', \mathrm{kInd}\right)\right]$. Finally, ciphertext ct is a tuple $(\mathrm{CT}, \mathrm{C})$. Element CT is computed by $(\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{Encaps}'\left(1^\lambda, \mathrm{pp}'_\kappa, \mathrm{cInd}\right)$, that is CT uniquely determines $\mathrm{K} \in \mathbb{K}_\lambda$ by correctness of Π_{KEM}. Furthermore, $\mathrm{C} = \mathrm{Enc}'\left(1^\lambda, \mathrm{symk}, m\right)$, where $\mathrm{symk} = \mathrm{KDF}\left(1^\lambda, \mathrm{dk}, \mathrm{K}\right)$.

Decryption algorithm Dec on input $\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{sk}, ct\right)$ computes $\mathrm{Decaps}'\left(1^\lambda, \mathrm{pp}'_\kappa, \mathrm{sk}, \mathrm{CT}\right)$. Hence, by correctness of Π_{KEM} (cf. Definition 2.3) it holds

$$\mathrm{Decaps}'\left(1^\lambda, \mathrm{pp}'_\kappa, \mathrm{sk}, \mathrm{CT}\right) = \mathrm{K} \ .$$

Next, Dec evaluates function $\mathrm{KDF}\left(1^\lambda, \mathrm{dk}, \mathrm{K}\right)$ and hence, receives symk from above. Finally, Dec executes the deterministic algorithm $\mathrm{Dec}'\left(1^\lambda, \mathrm{symk}, \mathrm{C}\right) = \mathrm{Dec}'\left(1^\lambda, \mathrm{symk}, \mathrm{Enc}'\left(1^\lambda, \mathrm{symk}, m\right)\right)$ and, by correctness of Π_{DEM}, reconstructs and outputs m. $\square$

C.5 CCA-Security Definition for Predicate Encryption Schemes

Next we define the CCA-security experiment for predicate encryption schemes. We use the notation of hybrid construction, i.e. the ciphertext is split into two parts. The definition is very similar to the definition of experiment $\text{aP-KEM}^{\text{aCCA}}_{\Pi_{\text{KEM}},\mathcal{A}}(\lambda, \text{des})$. Index i denotes the number of a covered key generation query and kInd_i denotes the key index used in the query with number i. W.l.o.g. we assume that $\mathcal{A}$ uses index i in the oracle queries only after the i'th query to the covered key generation oracle. In the security proof we will slightly change this experiment. Those parts of the experiment, which will be changed later, are framed and numbered.

$\mathbf{aPE}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$:

Setup : Challenger $\mathcal{C}$ generates $(\text{msk}, \text{pp}_\kappa) \leftarrow \text{Setup}\left(1^\lambda, \text{des}\right)$ and starts $\mathcal{A}\left(1^\lambda, \text{pp}_\kappa\right)$.

Phase I : $\mathcal{A}$ has access to the following oracles:

CoveredKeyGen (kInd_i) for $\text{kInd}_i \in \mathbb{X}_\kappa$: $\mathcal{C}$ generates and stores $\text{sk}_i \leftarrow \text{KeyGen}(\text{msk}, \text{kInd}_i)$.

Open (i) : $\mathcal{C}$ returns sk_i. We call the corresponding key index kInd_i a corrupted key index.

Decrypt $((\text{CT}, \text{C}), i)$: If $\text{CT} \in \mathbb{C}_{\text{cInd}}$ for $\text{cInd} \in \mathbb{Y}_\kappa$, and $\text{C} \in \{0,1\}^*$, then the challenger $\mathcal{C}$ returns $\text{Dec}(\text{sk}_i, (\text{CT}, \text{C}))$.

Challenge : $\mathcal{A}$ submits $m_0, m_1 \in \mathcal{M}$ of the same length and a target ciphertext index $\text{cInd}^* \in \mathbb{Y}_\kappa$, under the restriction that for every corrupted key index kInd_i it holds $\text{R}_\kappa(\text{kInd}_i, \text{cInd}^*) = 0$. Challenger flips a bit $b \leftarrow \{0,1\}$ and returns $\langle 11 \rangle$ $\boxed{ct^* = (\text{CT}^*, \text{C}^*) \leftarrow \text{Enc}(\text{cInd}^*, m_b)}$.

Phase II : $\mathcal{A}$ has access to the following oracles:

CoveredKeyGen (kInd_i) : As before.

Open (i) : As before, under the restriction that it must hold $\text{R}_\kappa(\text{kInd}_i, \text{cInd}^*) = 0$.

Decrypt $((\text{CT}, \text{C}), i)$: If $\text{CT} \in \mathbb{C}_{\text{cInd}}$ for $\text{cInd} \in \mathbb{Y}_\kappa$, $\text{C} \in \{0,1\}^*$, and $(\text{CT}, \text{C}) \neq ct^*$, $\mathcal{C}$ returns $\langle 12 \rangle$ $\boxed{\text{Dec}(\text{sk}_i, (\text{C}, \text{CT}))}$.

Guess : $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$. If one of the restrictions is violated, the output of the experiment is 0. The output of the experiment is 1 iff $b' = b$.

The advantage of $\mathcal{A}$ in security experiment $\text{aPE}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$ is defined as

$$\text{Adv-aPE}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda, \text{des}) := \Pr\left[\text{aPE}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda, \text{des}) = 1\right] - 1/2 \ .$$

Definition C.6. *A predicate encryption* Π *for predicate family* $\mathcal{R}_{\Omega,\Sigma}$ *is called* ***adaptively (or fully) secure against adaptively chosen-ciphertext attacks*** *if for every* $\text{des} \in \Omega$ *and every ppt (in* λ*) adversary* $\mathcal{A}$ *the function* $\text{Adv-aPE}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$ *is negligible in* λ.

C.6 Security of Hybrid Construction

In this subsection we will show that the hybrid construction Π_{hyb} from Subsection C.4 is secure.

Theorem C.1. *Suppose* Π_{KEM} *is a fully CCA-secure KEM,* Π_{DEM} *is a CCA-secure DEM, and* $\mathcal{KDF}$ *is a secure family of key derivation functions. Then, hybrid scheme* Π_{PE} *is a fully CCA-secure predicate encryption scheme. In particular, for every* $\text{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exist ppt algorithms* $\mathcal{B}_1$*,* $\mathcal{B}_2$*, and* $\mathcal{B}_3$ *with essentially the same running time as* $\mathcal{A}$ *such that for sufficiently large* λ *it holds*

$$\begin{aligned}\text{Adv-aPE}^{\text{aCCA}}_{\Pi,\mathcal{A}}(\lambda, \text{des}) \leq{}& 2 \cdot \text{Adv-aP-KEM}^{\text{aCCA}}_{\Pi,\mathcal{B}_1}(\lambda, \text{des}) \\ &+ \text{Adv-KDF}_{\mathcal{KDF},\mathcal{B}_2}(\lambda) + \text{Adv-DEM}^{\text{CCA}}_{\Pi,\mathcal{B}_3}(\lambda) \ .\end{aligned}$$

Proof. We define a sequence of probability experiments in Fig. 2.

Fig. 2. Proof Structure

$$\mathrm{G}_0 \xrightarrow[\approx]{} \mathrm{G}_1 \xrightarrow[\text{KEM}]{} \mathrm{G}_2 \xrightarrow[\text{KDF}]{} \mathrm{G}_3\ (\text{DEM})$$

G_0 is the CCA experiment $\text{aPE}^{\text{CCA}}_{\Pi,\mathcal{A}}(\lambda, \text{des})$ and the other games arise from it through simple modifications summarized in Table 2. We explain the modifications in detail in the proof. The indistinguishability

Table 2. The probability experiments from security proof of hybrid construction.

G_1:	Modify $^{\langle 12\rangle}$	**Decrypt** $((\mathrm{CT}^*, \mathrm{C}), i) := \mathrm{Dec}(\mathrm{symk}^*, \mathrm{C})$ if $\mathrm{R}(\mathrm{kInd}_i, \mathrm{cInd}^*) = 1$
G_2:	Modify K^* in $^{\langle 11\rangle}$	$\mathrm{K}^* \leftarrow \mathbb{K}_\lambda$
G_3:	Modify symk^* in $^{\langle 11\rangle}$	$\mathrm{symk}^* \leftarrow \{0,1\}^{l(\lambda)}$

of the games is based on the security of corresponding cryptographic primitives labeling the edges. In the last game the advantage of any adversary is negligible due to the security of DEM.

Experiment G_1 is defined as G_0 except for $^{\langle 12\rangle}$. Namely, in the second phase decryption queries on $ct = (\mathrm{CT}, \mathrm{C})$ and i which satisfies $\mathrm{CT} = \mathrm{CT}^*$, $\mathrm{C} \neq \mathrm{C}^*$, and $\mathrm{R}(\mathrm{kInd}_i, \mathrm{cInd}^*) = 1$, are answered directly with $\mathrm{Dec}(\mathrm{symk}^*, \mathrm{C})$. Thereby symk^* is the key used to compute the challenge ciphertext $ct^* = (\mathrm{CT}^*, \mathrm{C}^*)$.

The change in G_1 is conceptual, since our definition of KEM is error-free and hence, symk^* is the encapsulated key for CT^*. Hence, for every $\mathrm{des} \in \Omega$ it holds for every $\mathcal{A}$

$$\Pr[\mathcal{A} \text{ wins in } G_0] = \Pr[\mathcal{A} \text{ wins in } G_1] \quad .$$

Experiment G_2 is defined as G_1 except for the computation of K^* in $^{\langle 11\rangle}$. Namely, the challenge is generated as follows. $\mathcal{C}$ computes an encapsulation as before $(\_, \mathrm{CT}^*) \leftarrow \mathrm{Encaps}'(\mathrm{cInd}^*)$. Then, a random key $\mathrm{K}^* \leftarrow \mathbb{K}_\lambda$ is chosen. At the end the symmetric part is computed as before by $\mathrm{symk}^* := \mathrm{KDF}(\mathrm{dk}, \mathrm{K}^*)$ and $\mathrm{C}^* \leftarrow \mathrm{Enc}'(\mathrm{symk}^*, m_b)$. Due to this modification C^* and CT^* are independently generated such that CT^* does not contain any information about the symmetric key used to encrypt the actual message. Both experiments are indistinguishable due to the security property of KEM, as stated in the following lemma.

Lemma C.1. *For every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}_1$ such that for every security parameter λ and every $\mathrm{des} \in \Omega$ it holds*

$$\Pr[\mathcal{A} \textit{ wins in } G_1] - \Pr[\mathcal{A} \textit{ wins in } G_2] = 2 \cdot \text{Adv-aP-KEM}^{\text{aCCA}}_{\Pi_{\text{KEM}}, \mathcal{B}_1}(\lambda, \mathrm{des}) \quad .$$

The running time of $\mathcal{B}_1$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Given an algorithm $\mathcal{A}$, which can distinguish between G_1 and G_2, we construct an algorithm $\mathcal{B}_1$ against Π_{KEM} with essentially the same running time.

We analyze the constructed algorithm. $\mathcal{B}_1$ gets as input correctly generated public parameters and extends these by the derivation key for the hash function. This corresponds to the computations in the experiment. Next, we state, that $\mathcal{B}_1$ answers all queries of $\mathcal{A}$ correctly using the own oracles. The covered key generation queries and the opening queries are redirected to the own challenger and hence, all computations for these queries are performed correctly. By construction, $\mathcal{B}_1$ queries an irregular opening query if and only if $\mathcal{A}$ does. Next, we observe that by construction, $\mathcal{B}_1$ never queries decapsulation of CT^* in the second phase. In Phase I such a query is allowed. Hence, all queries of $\mathcal{B}_1$ are permissible if $\mathcal{A}$ does not violate the restrictions of the experiment.

Next, we consider decryption queries. Thereby, the special queries from the second phase will be analyzed separately. The decryption queries of $\mathcal{A}$ are split into the decapsulation part, realized using the own decapsulation oracle, and the decryption part, performed by $\mathcal{B}$. Since the challenger of $\mathcal{B}$ executes the decapsulation algorithm, $\mathcal{B}$ derives the corresponding symmetric key from it and hence, answers such decryption queries as defined in the experiments. The special queries in the second phase for $\mathrm{CT} = \mathrm{CT}^*$ are answered using the symmetric key symk^* from the challenge phase. This is exactly as defined in the experiments.

Finally, consider the challenge phase. If K^* and CT^* in the challenge of $\mathcal{B}_1$ are independently generated (let say the challenge bit for $\mathcal{B}_1$ is $\mu = 1$), the view of $\mathcal{A}$ is as in G_2. Otherwise, the view of $\mathcal{A}$ is as in G_1 by construction of $\mathcal{B}_1$. Hence, we get

Algorithm 1: $\mathcal{B}_1$ in experiment $\text{aP-KEM}^{\text{aCCA}}_{\Pi_{\text{KEM}},\mathcal{B}_1}(\lambda, \text{des})$

Input : $(1^\lambda, \text{pp}'_\kappa)$.
Require: $(\text{msk}, \text{pp}'_\kappa) \in [\text{Setup}'(1^\lambda, \text{des})]$.
1 **Setup**
2 Generate $\text{dk} \leftarrow \text{Sample}(1^\lambda)$, set $\text{pp}_\kappa := (\text{pp}'_\kappa, \text{dk})$, and simulate $\mathcal{A}$ with $(1^\lambda, \text{pp}_\kappa)$.
3 **Phase I**
4 **CoveredKeyGen** (kInd_i) with $\text{kInd}_i \in \mathbb{X}_\kappa$:
5 Queries are redirected to the own challenger.
6 **Open** (i):
7 Queries are redirected to the own challenger.
8 **Decrypt** $((\text{CT}, \text{C}), i)$ with $\text{CT} \in \mathbb{C}_{\text{cInd}}$ for $\text{cInd} \in \mathbb{Y}_\kappa$:
9 Query the own oracle **Decapsulate** $(\text{CT}, i) \rightarrow \text{K}$. Compute $\text{symk} := \text{KDF}(1^\lambda, \text{dk}, \text{K})$ and return $m \leftarrow \text{Dec}'(1^\lambda, \text{symk}, \text{C})$.
10 **Challenge** on input $m_0, m_1 \in \mathcal{M}$ of the same length and $\text{cInd}^* \in \mathbb{Y}_\kappa$:
11 Asks for the challenge $(\text{K}^*, \text{CT}^*)$ on cInd^*.
12 Compute and store $\text{symk}^* := \text{KDF}(1^\lambda, \text{dk}, \text{K}^*)$.
13 Flip a bit $b \leftarrow \{0,1\}$, and compute $\text{C}^* \leftarrow \text{Enc}'(1^\lambda, \text{symk}^*, m_b)$.
14 Return the challenge $ct^* := (\text{CT}^*, \text{C}^*)$.
15 **Phase II**
16 **CoveredKeyGen** (kInd_i) with $\text{kInd}_i \in \mathbb{X}_\kappa$:
17 As before.
18 **Open** (i):
19 As before.
20 **Decrypt** $((\text{CT}, \text{C}), i)$ with $\text{CT} \in \mathbb{C}_{\text{cInd}}$ for $\text{cInd} \in \mathbb{Y}_\kappa$:
21 If $\text{R}(\text{kInd}_i, \text{cInd}) = 1$, $\text{CT} = \text{CT}^*$ and $\text{C} \neq \text{C}^*$ return $m \leftarrow \text{Dec}'(1^\lambda, \text{symk}^*, \text{C})$. Other permissible queries are answered as in Phase I.
22 **Guess** on input $b' \in \{0,1\}$:
23 Output 1 if and only if $b' = b$ and $\mathcal{A}$ did not violate the restrictions.

$$\begin{aligned}
\text{Adv-aP-KEM}^{\text{aCCA}}_{\Pi,\mathcal{B}_1}(\lambda, \text{des}) &= \Pr\left[\text{aP-KEM}^{\text{aCCA}}_{\Pi,\mathcal{B}_1}(\lambda, \text{des}) = 1\right] - 1/2 \\
&= 1/2 \cdot \left(\Pr\left[\mathcal{B}_1(1^\lambda, \text{pp}'_\kappa) = 0 \mid \mu = 0\right] + \Pr\left[\mathcal{B}_1(1^\lambda, \text{pp}'_\kappa) = 1 \mid \mu = 1\right] - 1\right) \\
&= 1/2 \cdot \left(\Pr\left[\mathcal{B}_1(1^\lambda, \text{pp}'_\kappa) = 1 \mid \mu = 1\right] - \Pr\left[\mathcal{B}_1(1^\lambda, \text{pp}'_\kappa) = 1 \mid \mu = 0\right]\right) \\
&= 1/2 \cdot (|\Pr[b' = b \mid \mu = 1] - \Pr[b' = b \mid \mu = 0]|) \\
&= 1/2 \cdot (\Pr[\mathcal{A} \text{ wins in } \text{G}_2] - \Pr[\mathcal{A} \text{ wins in } \text{G}_1]) \quad .
\end{aligned}$$

This finally proves Lemma C.1. ◻

Experiment G_3 is defined as G_2 except for the computation of symk^* in $\langle 11 \rangle$, which is chosen at random: $\text{symk}^* \leftarrow \{0,1\}^{l(\lambda)}$.

Lemma C.2. *For every ppt algorithm $\mathcal{A}$ and every* $\text{des} \in \Omega$ *there exists a ppt algorithm $\mathcal{B}_2$ such that for every security parameter λ it holds*

$$\text{Adv-KDF}_{\mathcal{KDF},\mathcal{B}_2}(\lambda) = |\Pr[\mathcal{A} \textit{ wins in } \text{G}_2] - \Pr[\mathcal{A} \textit{ wins in } \text{G}_3]| \quad .$$

The running time of $\mathcal{B}_2$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Assume that there exist a ppt algorithm $\mathcal{A}$ such that

$$|\Pr[\mathcal{A} \text{ wins in } \text{G}_2] - \Pr[\mathcal{A} \text{ wins in } \text{G}_3]|$$

is not negligible. We construct an algorithm $\mathcal{B}_2$ which breaks the security property of $\mathcal{KDF}$.

$\mathcal{B}_2$ given 1^λ, dk, and symk_μ simulates the experiment using correctly generated msk and pp_κ as defined in the experiments, but uses symk_μ instead of symk^*. $\mathcal{B}_2$ outputs 1 if and only if $\mathcal{A}$ outputs the correct bit and does not violates the restrictions. By construction, if $\mu = 0$ the view of $\mathcal{A}$ is as in G_2. Otherwise, the view of $\mathcal{A}$ is as in G_3. Hence, we get

$$\begin{aligned}\text{Adv-KDF}_{\mathcal{KDF},\mathcal{B}_2}(\lambda) &= \left|\Pr\left[\mathcal{B}_2\left(1^\lambda, \mathrm{dk}, \mathrm{symk}_0\right) = 1\right] - \Pr\left[\mathcal{B}_2\left(1^\lambda, \mathrm{dk}, \mathrm{symk}_1\right) = 1\right]\right| \\ &= \left|\Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_2\right] - \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_3\right]\right| \ .\end{aligned}$$

This proves Lemma C.2. □

Now we analyze experiment G_3. We prove the following lemma.

Lemma C.3. *For every ppt algorithm $\mathcal{A}$ and every* des $\in \Omega$ *there exists a ppt algorithm $\mathcal{B}_3$ such that for every security parameter λ it holds*

$$\text{Adv-DEM}^{\mathrm{CCA}}_{\Pi,\mathcal{B}_3}(\lambda) = \Pr\left[\mathcal{A} \textit{ wins in } \mathrm{G}_3\right] - 1/2 \ .$$

The running time of $\mathcal{B}_2$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Assume that there exist a ppt algorithm $\mathcal{A}$ which has not-negligible advantage in G_3. We construct an algorithm $\mathcal{B}_3$ which breaks the CCA-security property of Π_{DEM}:

$\mathcal{B}_3$ on input 1^λ simulates $\mathcal{A}$ as follows:

Setup, Phase I: Using correctly generated msk and pp_κ, $\mathcal{B}_3$ simulates everything as defined in the experiment until the challenge phase.

Challenge: Given m_0, m_1, and cInd^* ask for the challenge on m_0, m_1 and receive C^*. Compute the encapsulation $(\_, \mathrm{CT}^*) \leftarrow \mathrm{Encaps}'\left(1^\lambda, \mathrm{pp}'_\kappa, \mathrm{cInd}^*\right)$, and output $(\mathrm{CT}^*, \mathrm{C}^*)$.

Phase II: As defined in the experiment except for the decryption query:

Decrypt $((\mathrm{CT}, \mathrm{C}), i)$ with $\mathrm{R}(\mathrm{kInd}_i, \mathrm{cInd}) = 1$, $\mathrm{CT} = \mathrm{CT}^*$ and $\mathrm{C} \neq \mathrm{C}^*$. Query the own oracle **Decrypt** (C) and given m, return it.

Guess: Output the output of $\mathcal{A}$.

$\mathcal{B}_3$ does not use the own oracle in the first Phase. Furthermore, in the second phase $\mathcal{B}_3$ never ask the decryption of C^* by construction. Hence, all queries of $\mathcal{B}_3$ are permissible. Furthermore, G_3 is defined in such a way, that symk^* from the challenge is chosen independently from CT^*. This corresponds to the key generation of the challenger of $\mathcal{B}_3$. $\mathcal{B}_3$ wins if and only if $\mathcal{A}$ wins. Hence, it holds:

$$\begin{aligned}\text{Adv-DEM}^{\mathrm{CCA}}_{\Pi,\mathcal{B}_3}(\lambda) &= \Pr\left[\mathrm{DEM}^{\mathrm{CCA}}_{\Pi,\mathcal{B}_3}(\lambda) = 1\right] - 1/2 \\ &= \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_3\right] - 1/2 \ .\end{aligned}$$

This proves Lemma C.3. □

All together we get for every ppt $\mathcal{A}$:

$$\begin{aligned}\text{Adv-aPE}^{\mathrm{CCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) &= \Pr\left[\mathrm{aPE}^{\mathrm{CCA}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) = 1\right] - 1/2 \\ &= \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_0\right] - \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_1\right] \\ &\quad + \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_1\right] - \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_2\right] \\ &\quad + \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_2\right] - \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_3\right] \\ &\quad + \Pr\left[\mathcal{A} \text{ wins in } \mathrm{G}_3\right] - 1/2 \\ &\leq 0 + 2 \cdot \text{Adv-aP-KEM}^{\mathrm{aCCA}}_{\Pi,\mathcal{B}_1}(\lambda, \mathrm{des}) \\ &\quad + \text{Adv-KDF}_{\mathcal{KDF},\mathcal{B}_2}(\lambda) + \text{Adv-DEM}^{\mathrm{CCA}}_{\Pi,\mathcal{B}_3}(\lambda) \ .\end{aligned}$$

This finally proves Theorem C.1. □

D Security Proof of the CCA-Secure Pair Encoding Framework

In this section we present a formal proof of Theorem 4.1. We start with general lemmas which will be used in the main proof, which is then presented in Subsection D.3.

D.1 On the distribution of semi-functional components

In this subsection we will prove some useful lemmas about the output distributions of semi-functional algorithms. These lemmas will be used in several lemmas of the main proof.

The first lemma states that as long as $\hat{\alpha}$ is chosen uniformly at random, the distribution of the resulting semi-functional keys is independent of the concrete generator of $\mathbb{G}_{p_2}$.

Lemma D.1. *For every* λ*, every* $\mathrm{des} \in \Omega$*, every* $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \_, \_\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$*, every* $\mathrm{kInd} \in \mathbb{X}_\kappa$*, every* $\mathrm{type} \in \{1, 2, 3\}$*, and every generator* $\tilde{g}_2 \in \mathbb{G}_{p_2}$ *it holds*

$$
\begin{aligned}
&\Pr\left[\widehat{\mathrm{sk}} : \hat{\alpha} \leftarrow \mathbb{Z}_N, \widehat{\mathrm{sk}} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, \mathrm{type}, \hat{\alpha}, g_2, \hat{\boldsymbol{h}}\right)\right] \\
&= \Pr\left[\widetilde{\mathrm{sk}} : \tilde{\alpha} \leftarrow \mathbb{Z}_N, \widetilde{\mathrm{sk}} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, \mathrm{type}, \tilde{\alpha}, \tilde{g}_2, \hat{\boldsymbol{h}}\right)\right] \quad .
\end{aligned}
$$

Proof. We will prove the lemma for all key types simultaneously. Let λ, $\mathrm{des} \in \Omega$, $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \_, \_\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, and a generator $\tilde{g}_2 \in \mathbb{G}_{p_2}$ be arbitrary, but fixed. Elements $g_2, \tilde{g}_2 \in \mathbb{G}_{p_2}$ are generators of $\mathbb{G}_{p_2}$. Hence, there exists $x \in \mathbb{Z}_{p_2}^*$ such that $\tilde{g}_2 = g_2^x$. Next, denote $\widehat{\mathrm{sk}} = (\mathrm{kInd}, \boldsymbol{K}_1)$ and $\widetilde{\mathrm{sk}} = (\mathrm{kInd}, \boldsymbol{K}_2)$. By the definition of SFKeyGen, the input values $\hat{\alpha}$ and g_2 affect only the $\mathbb{G}_{p_2}$ components of the group elements in the generated key. Furthermore, these components are independently generated from the $\mathbb{G}_{p_1}$ and from the $\mathbb{G}_{p_3}$ components. Hence, the distribution of the $\mathbb{G}_{p_1}$ components and the distribution of the $\mathbb{G}_{p_3}$ components of $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ are identical and we will consider only the distributions of the $\mathbb{G}_{p_2}$ components of $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$.

In the first probability space, the $\mathbb{G}_{p_2}$ components $\widehat{\boldsymbol{K}}$ of $\boldsymbol{K}_1$ are determined by the mutually independent random variables $\hat{\boldsymbol{r}}$ (defined by SFKeyGen) and $\hat{\alpha}$. In the second probability space, the $\mathbb{G}_{p_2}$ components $\widetilde{\boldsymbol{K}}$ of $\boldsymbol{K}_2$ are determined by the mutually independent random variables $\tilde{\boldsymbol{r}}$, (defined by SFKeyGen) and $\tilde{\alpha}$. Namely, it holds:

$$
\widehat{\boldsymbol{K}} = \begin{cases} g_2^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 1 \\ g_2^{\boldsymbol{k}\left(\hat{\alpha}, \hat{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 2 \\ g_2^{\boldsymbol{k}\left(\hat{\alpha}, \boldsymbol{0}, \boldsymbol{0}\right)} & \text{if type} = 3 \end{cases} \quad \text{and} \quad \widetilde{\boldsymbol{K}} = \begin{cases} \tilde{g}_2^{\boldsymbol{k}\left(0, \tilde{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} = g_2^{\boldsymbol{k}\left(0, x \cdot \tilde{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 1 \\ \tilde{g}_2^{\boldsymbol{k}\left(\tilde{\alpha}, \tilde{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} = g_2^{\boldsymbol{k}\left(x \cdot \tilde{\alpha}, x \cdot \tilde{\boldsymbol{r}}, \hat{\boldsymbol{h}}\right)} & \text{if type} = 2 \\ \tilde{g}_2^{\boldsymbol{k}\left(\tilde{\alpha}, \boldsymbol{0}, \boldsymbol{0}\right)} = g_2^{\boldsymbol{k}\left(x \cdot \tilde{\alpha}, \boldsymbol{0}, \boldsymbol{0}\right)} & \text{if type} = 3 \ , \end{cases}
$$

where $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$. For the keys of Type 1 and of Type 2 the values $\hat{\boldsymbol{r}}$ and $x \cdot \tilde{\boldsymbol{r}}$ are uniformly distributed over $\mathbb{Z}_{p_2}^{m_2}$ due to the choices of $\hat{\boldsymbol{r}}$ and $\tilde{\boldsymbol{r}}$, and because $x \neq 0 \pmod{p_2}$. Additionally, for the keys of Type 2 and of Type 3 the values $\hat{\alpha}$ and $x \cdot \tilde{\alpha}$ are uniformly distributed over $\mathbb{Z}_{p_2}$ due to the choices of $\hat{\alpha}$ and $\tilde{\alpha}$, and because $x \neq 0 \pmod{p_2}$. Hence, we deduce that the $\mathbb{G}_{p_2}$ components of the group elements in the keys are identically distributed in both probability experiments. As mentioned above, this implies that $\widehat{\mathrm{sk}}$ and $\widetilde{\mathrm{sk}}$ are identically distributed. This proves the lemma. $\square$

Next lemma is very similar and shows that the output distribution of the semi-functional encapsulation algorithm is independent of the concrete generator of $\mathbb{G}_{p_2}$.

Lemma D.2. *For every* λ*, every* $\mathrm{des} \in \Omega$*, every* $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$*, every* $\mathrm{cInd} \in \mathbb{Y}_\kappa$*, and every generator* $\tilde{g}_2 \in \mathbb{G}_{p_2}$ *it holds*

$$
\begin{aligned}
&\Pr\left[\widehat{\mathrm{K}}, \widehat{\mathrm{CT}} : \widehat{\mathrm{K}}, \widehat{\mathrm{CT}} \leftarrow \mathrm{SFEncaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)\right] \\
&= \Pr\left[\widetilde{\mathrm{K}}, \widetilde{\mathrm{CT}} : \widetilde{\mathrm{K}}, \widetilde{\mathrm{CT}} \leftarrow \mathrm{SFEncaps}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{cInd}, \tilde{g}_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)\right] \quad .
\end{aligned}
$$

Proof. Let λ, $\mathrm{des} \in \Omega$, $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, and a generator $\tilde{g}_2 \in \mathbb{G}_{p_2}$ be arbitrary, but fixed. Elements $g_2, \tilde{g}_2 \in \mathbb{G}_{p_2}$ are generators of $\mathbb{G}_{p_2}$. Hence, there exists $x \in \mathbb{Z}_{p_2}^*$ such that $\tilde{g}_2 = g_2^x$. Next, denote $\widehat{\mathrm{CT}} = (\mathrm{cInd}, \boldsymbol{C}_1, C_1'')$ and $\widetilde{\mathrm{CT}} = (\mathrm{cInd}, \boldsymbol{C}_2, C_2'')$. First, let us consider the distributions of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$. By the definition of SFEncaps, the input value g_2 only affects the $\mathbb{G}_{p_2}$ components of the group elements in the generated encapsulation. Furthermore, these components are generated independently from the $\mathbb{G}_{p_1}$ components and all these elements do not contain the $\mathbb{G}_{p_3}$ components. Hence, the distribution of the $\mathbb{G}_{p_1}$ components of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are identical and we will only consider the distributions of the $\mathbb{G}_{p_2}$ components of these elements.

In the first probability space, the $\mathbb{G}_{p_2}$ components $\widehat{\boldsymbol{C}}$ of $\boldsymbol{C}_1$ are determined by the mutually independent random variables $\hat{s}$ and $\hat{\boldsymbol{s}}$ (defined by SFEncaps). In the second probability space, the $\mathbb{G}_{p_2}$ components $\widetilde{\boldsymbol{C}}$ of $\boldsymbol{C}_2$ are determined by the mutually independent random variables $\tilde{s}$ and $\tilde{\boldsymbol{s}}$ (defined by SFEncaps). Namely, it holds:

$$\widehat{\boldsymbol{C}} = g_2^{\boldsymbol{c}\left(\hat{s},\hat{\boldsymbol{s}},\hat{\boldsymbol{h}}\right)} \quad \text{and} \quad \widetilde{\boldsymbol{C}} = \tilde{g}_2^{\boldsymbol{c}\left(\tilde{s},\tilde{\boldsymbol{s}},\hat{\boldsymbol{h}}\right)} = g_2^{\boldsymbol{c}\left(x\cdot\tilde{s},x\cdot\tilde{\boldsymbol{s}},\hat{\boldsymbol{h}}\right)} \ ,$$

where $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$. The values $\hat{s}, x\cdot\tilde{s} \in \mathbb{Z}_{p_2}$, and $\hat{\boldsymbol{s}}, x\cdot\tilde{\boldsymbol{s}} \in \mathbb{Z}_{p_2}^{w_2}$ are uniformly distributed due to the choice of the corresponding random values and because $x \neq 0 \pmod{p_2}$. Hence, we deduce that $\widehat{\boldsymbol{C}}$ and $\widetilde{\boldsymbol{C}}$ are identically distributed. As mentioned above, this implies that $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are identically distributed too. Next, we claim that if the random variables, which determine the $\mathbb{G}_{p_1}$ components and the $\mathbb{G}_{p_2}$ components of $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ take the same values than $C_1'' = C_2''$. Namely, $\boldsymbol{C}_1 = \boldsymbol{C}_2$ implies that the hash value t and the random value s are the same in both cases and furthermore it holds $\hat{s} = x\cdot\tilde{s} \pmod{p_2}$. Hence, by construction of SFEncaps it holds

$$C_2'' = \left(U_1^t \cdot V_1\right)^s \cdot \left(\tilde{g}_2^{\hat{u}_2\cdot t}\cdot\tilde{g}_2^{\hat{v}_2}\right)^{\tilde{s}} = \left(U_1^t \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2\cdot t}\cdot g_2^{\hat{v}_2}\right)^{x\cdot\tilde{s}} = \left(U_1^t \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2\cdot t}\cdot g_2^{\hat{v}_2}\right)^{\hat{s}} = C_1'' \ .$$

This proves the lemma. □

D.2 Supplementary Algorithms

In this subsection we will show how to simulate different elements of the scheme. The algorithms from this subsection will be used in several reductions. Partially, these algorithms were (implicitly) presented in the proof of the original framework from [3]. We separately define these algorithms in order to avoid repetitions and also in order to simplify the involved proofs.

Simulation of the Semi-Functional Public Parameters. By the definition of algorithm SFSetup, the semi-functional public parameters for the predicate R_κ consist of the normal public parameters pp_κ, a group generator $\hat{g}_2 \in \mathbb{G}_{p_2}$, a vector $\hat{\boldsymbol{h}} \in \mathbb{Z}_{p_2}^n$ and two additional elements $\hat{u}_2, \hat{v}_2 \in \mathbb{Z}_{p_2}$ ($n = \mathrm{Param}(\kappa)$). All these elements are chosen independently and remain hidden in the realization of the scheme. But for the proof, these elements are essential, since the normal keys and the normal challenge encapsulation are changed to their semi-functional counterparts. That is, the $\mathbb{G}_{p_2}$ components, which we also call semi-functional, are appended to the corresponding group elements. These components are not completely random (which is the case for the $\mathbb{G}_{p_3}$ components of the user secret keys). Rather, the $\mathbb{G}_{p_2}$ components have the same structure as the normal $\mathbb{G}_{p_1}$ components. This is indispensable in the proof, where the properly distributed semi-functional keys and a semi-functional challenge encapsulation will be generated given either a generator of $\mathbb{G}_{p_1p_2}$ or a generator of $\mathbb{G}_{p_2p_3}$ instead of a generator of $\mathbb{G}_{p_2}$.

In this subsection we actually show how to generate properly distributed semi-functional public parameters (except $g_2 \in \mathbb{G}_{p_2}$) given $\mathrm{des} \in \Omega$, the restricted group description $\mathbb{GD}_N$, $g_1 \in \mathbb{G}_{p_1}$, and $g_3 \in \mathbb{G}_{p_3}$.

Lemma D.3. *There exist a ppt algorithm* SimPP *such that for every security parameter λ and every* $\mathrm{des} \in \Omega$ *it holds*

1. SimPP *can be used to generate the normal public parameters and the master secret key:*

$$\begin{aligned} &\Pr\left[\mathrm{msk}, \mathrm{pp}_\kappa : (\mathrm{msk}, \mathrm{pp}_\kappa) \leftarrow \mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right] \\ =\ &\Pr\left[\mathrm{msk}, \mathrm{pp}_\kappa : (\mathrm{msk}, \mathrm{pp}_\kappa, \_, \_, \_) \leftarrow \mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right] \\ =\ &\Pr\left[\mathrm{msk}, \mathrm{pp}_\kappa : \begin{array}{l} \widetilde{\mathbb{GD}} \leftarrow \mathcal{G}\left(1^\lambda\right), \tilde{g}_1 \leftarrow \mathbb{G}_{p_1}, \tilde{g}_3 \leftarrow \mathbb{G}_{p_3}, \\ (\mathrm{msk}, \mathrm{pp}_\kappa, \_, \_, \_) \leftarrow \mathrm{SimPP}\left(\mathrm{des}, \widetilde{\mathbb{GD}}_{\tilde{N}}, \tilde{g}_1, \tilde{g}_3\right) \end{array}\right] \ , \end{aligned}$$

where $\widetilde{\mathbb{GD}}_{\tilde{N}}$ is the restricted group description of $\widetilde{\mathbb{GD}}$.

2. For every $\widetilde{\mathbb{GD}} \in [\mathcal{G}(1^\lambda)]$*, every* $\tilde{g}_1 \in \mathbb{G}_{p_1}$*, every* $\tilde{g}_3 \in \mathbb{G}_{p_3}$*, and every* $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v)$ *generated by* $\mathrm{SimPP}\left(\mathrm{des}, \widetilde{\mathbb{GD}}_{\tilde{N}}, \tilde{g}_1, \tilde{g}_3\right)$ *it holds*

$$\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right) \quad ,$$

where $\mathbb{GD}_N = \widetilde{\mathbb{GD}}_{\tilde{N}}$*,* $g_1 = \tilde{g}_1$*,* $g_3 = \tilde{g}_3$*,* $g_1^{\boldsymbol{h}} = g_1^{\boldsymbol{h}'}$*,* $U_1 = g_1^u$*, and* $V_1 = g_1^v$*. Furthermore, the values* $\boldsymbol{h}'$*,* u*,* v *modulo* p_2 *and modulo* p_3 *are uncorrelated with* pp_κ*.*

3. For every $(\mathrm{msk}, \mathrm{pp}_\kappa) \in [\mathrm{Setup}(1^\lambda, \mathrm{des})]$ *it holds*

$$\begin{aligned}
&\Pr\left[\hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2 \,\middle|\, \mathrm{pp}_\kappa = \widehat{\mathrm{pp}}_{\hat{\kappa}} : \left(\widehat{\mathrm{msk}}, \widehat{\mathrm{pp}}_{\hat{\kappa}}, _, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \leftarrow \mathrm{SFSetup}(1^\lambda, \mathrm{des})\right] \\
&= \Pr\left[\begin{matrix} \boldsymbol{h}' \pmod{p_2}, \\ u \pmod{p_2}, \\ v \pmod{p_2} \end{matrix} \,\middle|\, \mathrm{pp}_\kappa = \widetilde{\mathrm{pp}}_{\tilde{\kappa}} : \begin{matrix} \widetilde{\mathbb{GD}} \leftarrow \mathcal{G}(1^\lambda), \tilde{g}_1 \leftarrow \mathbb{G}_{p_1}, \tilde{g}_3 \leftarrow \mathbb{G}_{p_3}, \\ \left(\widetilde{\mathrm{msk}}, \widetilde{\mathrm{pp}}_{\tilde{\kappa}}, \boldsymbol{h}', u, v\right) \leftarrow \mathrm{SimPP}\left(\mathrm{des}, \widetilde{\mathbb{GD}}_{\tilde{N}}, \tilde{g}_1, \tilde{g}_3\right) \end{matrix}\right] \quad ,
\end{aligned}$$

where $\widetilde{\mathbb{GD}}_{\tilde{N}}$ *is the restricted group description of* $\widetilde{\mathbb{GD}}$*.*

Proof. The algorithm SimPP is as follows:

Algorithm 2: SimPP

Input : $(\mathrm{des}, \mathbb{GD}_N, g_1, g_3)$.
1. Set $\kappa := (\mathrm{des}, N)$ and compute $n := \mathrm{Param}(\kappa)$.
2. Pick $\alpha \leftarrow \mathbb{Z}_N$ and compute $Y := \mathrm{e}(g_1, g_1)^\alpha$.
3. Pick $\boldsymbol{h}' \leftarrow \mathbb{Z}_N^n$ and $u, v \leftarrow \mathbb{Z}_N$. Compute $g_1^{\boldsymbol{h}'}$, $U_1 := g_1^u$ and $V_1 := g_1^v$.
4. Choose a hash function $\mathrm{H} \leftarrow \mathcal{H}_\kappa$.
5. Define $\mathrm{msk} := \alpha$ and $\mathrm{pp}_\kappa := \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}'}, U_1, V_1, g_3, Y, \mathrm{H}\right)$.

Output : $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v)$.

SimPP is a ppt algorithm with respect to λ by construction.

Consider the first statement of the lemma. The first equation holds by the definition of the algorithm SFSetup, which uses Setup for the generation of the master secret key and the public parameters. In the last probability space, $\widetilde{\mathbb{GD}}$, $\tilde{g}_1$ and $\tilde{g}_3$ are generated identically to the generation of $\mathbb{GD}$, g_1 and g_3 in the algorithm Setup. All other elements of the public parameters and the master secret key are properly distributed by construction of SimPP.

The second statement of the lemma holds by construction of SimPP. In particular, pp_κ fix the values $\boldsymbol{h}'$, u and v modulo p_1. By the Chinese Remainder Theorem these values are uncorrelated with $\boldsymbol{h}'$, u and v modulo p_2 and modulo p_3.

The last statement of the lemma holds by the definition of SFSetup, by construction of SimPP and because of the second statement. This completes the proof. ◻

Note, that SimPP outputs not only the properly distributed semi-functional public parameters, but also the exponents which are correlated with the public parameters modulo p_1. This will be exploited in the following reductions in order to generate properly distributed semi-functional keys and semi-functional encapsulations without a generator of $\mathbb{G}_{p_2}$.

Simulation of the Semi-Functional Encapsulation. From the previous section we know that using SimPP we can generate properly distributed (semi-functional) public parameters (except for $g_2 \in \mathbb{G}_{p_2}$). In this section we show how to generate correctly distributed semi-functional encapsulation (see the definition on page 12) given a generator of $\mathbb{G}_{p_1 p_2}$. The resulting algorithm will be used in almost all following reductions for the generation of the challenge.

Lemma D.4. *There exist a ppt algorithm* SimSFChlg *such that for every security parameter* λ*, every* $\mathrm{des} \in \Omega$*, every* $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \in [\mathrm{SFSetup}(1^\lambda, \mathrm{des})]$*, every* $\mathrm{cInd} \in \mathbb{Y}_\kappa$*, every* $X_1 \in \mathbb{G}_{p_1}$*,*

$X_1 \neq 1_{\mathbb{G}}$*, and every* $X_2 \in \mathbb{G}_{p_2}$*,* $X_2 \neq 1_{\mathbb{G}}$ *it holds*

$$\begin{aligned}
&\Pr\left[\mathrm{K}, \mathrm{CT} : (\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{SFEncaps}\left(\mathrm{pp}_\kappa, \mathrm{cInd}, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right)\right] \\
&= \Pr\left[\mathrm{K}, \mathrm{CT} \,\middle|\, \begin{array}{rl} \widetilde{\mathrm{pp}}_{\tilde{\kappa}} = \mathrm{pp}_\kappa, & \widetilde{\mathbb{GD}} \leftarrow \mathcal{G}\left(1^\lambda\right), \tilde{g}_1 \leftarrow \mathbb{G}_{p_1}, \tilde{g}_3 \leftarrow \mathbb{G}_{p_3}, \\ \boldsymbol{h}' = \hat{\boldsymbol{h}} \pmod{p_2}, & \left(\widetilde{\mathrm{msk}}, \widetilde{\mathrm{pp}}_{\tilde{\kappa}}, \boldsymbol{h}', u, v\right) \leftarrow \mathrm{SimPP}\left(\mathrm{des}, \widetilde{\mathbb{GD}}_N, \tilde{g}_1, \tilde{g}_3\right), \\ u = \hat{u}_2 \pmod{p_2}, & : \\ v = \hat{v}_2 \pmod{p_2} & (\mathrm{K}, \mathrm{CT}) \leftarrow \mathrm{SimSFChlg}\left(\widetilde{\mathrm{pp}}_{\tilde{\kappa}}, \widetilde{\mathrm{msk}}, \mathrm{cInd}, \boldsymbol{h}', u, v, X_1 X_2\right) \end{array}\right] ,
\end{aligned}$$

where $\widetilde{\mathbb{GD}}_N$ *is the restriction of* $\widetilde{\mathbb{GD}}$*.*

Furthermore, the second conditional probability is only over the random choices of SimSFChlg*.*

Proof. The algorithm SimSFChlg is as follows.

Algorithm 3: SimSFChlg - simulation of semi-functional challenge from SD2 (λ)

Input : $(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}, \boldsymbol{h}', u, v, X_1 X_2)$.
Require : $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$.

1. Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd})$. Let $w_1 := |\boldsymbol{c}|$.
2. Pick $s' \leftarrow \mathbb{Z}_N$ and $\boldsymbol{s}' = \left(s'_1, \ldots, s'_{w_2}\right) \leftarrow \mathbb{Z}_N^{w_2}$. Compute $\boldsymbol{C} := (X_1 X_2)^{\boldsymbol{c}\left(s', \boldsymbol{s}', \boldsymbol{h}'\right)}$.
3. Compute $t := \mathrm{H}(\mathrm{HInput}(\mathrm{cInd}, \boldsymbol{C}, \_))$ and $C'' := (X_1 X_2)^{s' \cdot (u \cdot t + v)}$.
4. Compute $\mathrm{K} := \mathrm{e}(X_1 X_2, g_1)^{\mathrm{msk} \cdot s'}$ and set $\mathrm{CT} := (\mathrm{cInd}, \boldsymbol{C}, C'')$.

Output : $(\mathrm{K}, \mathrm{CT})$.

SimSFChlg is a ppt algorithm with respect to λ by construction. In particular, all exponents in the description of the algorithm can be computed explicitly.

Let λ, $\mathrm{des} \in \Omega$, $\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, $X_1 \in \mathbb{G}_{p_1}$, $X_1 \neq 1_{\mathbb{G}}$, and $X_2 \in \mathbb{G}_{p_2}$, $X_2 \neq 1_{\mathbb{G}}$ be arbitrary but fixed. We denote the public parameters by $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$. We can write X_1 and X_2 by $g_1^{x_1}$ and by $g_2^{x_2}$, respectively. Thereby it holds $x_1 \neq 0 \pmod{p_1}$ and $x_2 \neq 0 \pmod{p_2}$.

The first probability distribution is over the interior choices of SFEncaps. On the one hand, these are the random choices of Encaps on input pp_κ and cInd: $s \leftarrow \mathbb{Z}_{p_1}$, and $\boldsymbol{s} \leftarrow \mathbb{Z}_{p_1}^{w_2}$ for $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$. On the other hand, these are $\hat{s} \leftarrow \mathbb{Z}_{p_2}$, and $\hat{\boldsymbol{s}} \leftarrow \mathbb{Z}_{p_2}^{w_2}$ chosen by SFEncaps itself. Key K and its encapsulation $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$ are completely determined by these values and by the semi-functional public parameters. Namely, it holds

$$\begin{aligned}
\mathrm{K} &= Y^s \ , \\
\boldsymbol{C} &= g_1^{\boldsymbol{c}(s, \boldsymbol{s}, \boldsymbol{h})} \cdot g_2^{\boldsymbol{c}\left(\hat{s}, \hat{\boldsymbol{s}}, \hat{\boldsymbol{h}}\right)} \ , \\
C'' &= \left(U_1^t \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2 \cdot t} \cdot g_2^{\hat{v}_2}\right)^{\hat{s}} \ ,
\end{aligned}$$

where $t = \mathrm{H}(\mathrm{HInput}(\mathrm{cInd}, \boldsymbol{C}, \_))$.

Now, consider SimSFChlg in the context of the conditional distribution defined in the lemma. All input values of SimSFChlg except for the values $\boldsymbol{h}', u, v \pmod{p_3}$ are fixed. In particular, $\widetilde{\mathrm{pp}}_{\tilde{\kappa}} = \mathrm{pp}_\kappa$, which implies $\widetilde{\mathrm{msk}} = \mathrm{msk}$. Furthermore, pp_κ determine $\boldsymbol{h}' \pmod{p_1}$, $u \pmod{p_1}$ and $v \pmod{p_1}$ by Statement 2 of Lemma D.3, since $\mathbb{GD}_N = \widetilde{\mathbb{GD}}_N$, $g_1 = \tilde{g}_1$, $g_3 = \tilde{g}_3$, $g_1^{\boldsymbol{h}} = g_1^{\boldsymbol{h}'}$, $V_1 = g_1^v$, and $U_1 = g_1^u$. By construction of SimSFChlg the key K is as follows

$$\begin{aligned}
\mathrm{K} &= \mathrm{e}(X_1 X_2, g_1)^{\widetilde{\mathrm{msk}} \cdot s'} \\
&= \mathrm{e}\left(g_1^{x_1}, g_1\right)^{\mathrm{msk} \cdot s'} \\
&= Y^{x_1 \cdot s'} \ .
\end{aligned}$$

Hence, K is a key with $s = x_1 \cdot s' \pmod{p_1}$, which is properly distributed due to the choice of s' and because $x_1 \neq 0 \pmod{p_1}$. Furthermore, it holds

$$\begin{aligned}
\boldsymbol{C} &= (X_1 X_2)^{\boldsymbol{c}\left(s', \boldsymbol{s}', \boldsymbol{h}'\right)} & C'' &= (X_1 X_2)^{s'(u \cdot t + v)} \\
&= g_1^{x_1 \cdot \boldsymbol{c}\left(s', \boldsymbol{s}', \boldsymbol{h}'\right)} \cdot g_2^{x_2 \cdot \boldsymbol{c}\left(s', \boldsymbol{s}', \boldsymbol{h}'\right)} & &= g_1^{x_1 \cdot s' \cdot (u \cdot t + v)} \cdot g_2^{x_2 \cdot s' \cdot (u \cdot t + v)} \\
&= g_1^{\boldsymbol{c}\left(x_1 \cdot s', x_1 \cdot \boldsymbol{s}', \boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{c}\left(x_2 \cdot s', x_2 \cdot \boldsymbol{s}', \hat{\boldsymbol{h}}\right)} \ , & &= \left(U_1^t \cdot V_1\right)^{x_1 \cdot s'} \cdot \left(g_2^{\hat{u} \cdot t + \hat{v}}\right)^{x_2 \cdot s'} \ ,
\end{aligned}$$

where $t = \mathrm{H}(\mathrm{HInput}(\mathrm{cInd}, \boldsymbol{C}, \_))$. In particular, we used $\boldsymbol{h}' = \hat{\boldsymbol{h}} \pmod{p_2}$, $u = \hat{u}_2 \pmod{p_2}$, and $v = \hat{v}_2 \pmod{p_2}$ in the last equations. Hence, $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$ is a semi-functional encapsulation of K with $\boldsymbol{s} = x_1 \cdot \boldsymbol{s}' \pmod{p_1}$, $\hat{s} = x_2 \cdot s' \pmod{p_2}$, and $\hat{\boldsymbol{s}} = x_2 \cdot \boldsymbol{s}' \pmod{p_2}$. Since $x_1 \neq 0 \pmod{p_1}$, $x_2 \neq 0 \pmod{p_2}$, all these elements are properly distributed due to the choice of s' and $\boldsymbol{s}'$, and due to the Chinese Remainder Theorem. This completes the proof. ☐

Simulation of the Semi-Functional Keys of Type 3. In this subsection, analogously to the previous subsection, we show how to generate correctly distributed semi-functional secret keys of Type 3 (see the definition on page 12) given a generator of $\mathbb{G}_{p_2 p_3}$.

Lemma D.5. *There exist a ppt algorithm* SimSFKeyT3 *such that for every security parameter λ, every* $\mathrm{des} \in \Omega$, *every* $(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \_, \_, \_) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, *every* $\mathrm{kInd} \in \mathbb{X}_\kappa$, *every* $Y_2 \in \mathbb{G}_{p_2}$, $Y_2 \neq 1_{\mathbb{G}}$, $Y_3 \in \mathbb{G}_{p_3}$ *it holds*

$$
\begin{aligned}
&\Pr\left[\mathrm{sk} : \hat{\alpha} \leftarrow \mathbb{Z}_N, \mathrm{sk} \leftarrow \mathrm{SFKeyGen}\left(1^\lambda, \mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, 3, \hat{\alpha}, g_2, \_\right)\right] \\
=& \Pr\left[\mathrm{sk} : \alpha' \leftarrow \mathbb{Z}_N, \mathrm{sk} \leftarrow \mathrm{SimSFKeyT3}\left(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, Y_2 Y_3, \alpha'\right)\right] \\
=& \Pr\left[\mathrm{sk} \;\middle|\; \widetilde{\mathrm{pp}}_{\tilde{\kappa}} = \mathrm{pp}_\kappa : \begin{array}{c} \widetilde{\mathbb{GD}} \leftarrow \mathcal{G}\left(1^\lambda\right), \tilde{g}_1 \leftarrow \mathbb{G}_{p_1}, \tilde{g}_3 \leftarrow \mathbb{G}_{p_3}, \\ \left(\widetilde{\mathrm{msk}}, \widetilde{\mathrm{pp}}_{\tilde{\kappa}}, \_, \_, \_\right) \leftarrow \mathrm{SimPP}\left(\mathrm{des}, \widetilde{\mathbb{GD}}_{\tilde{N}}, \tilde{g}_1, \tilde{g}_3\right), \\ \alpha' \leftarrow \mathbb{Z}_{\tilde{N}}, \mathrm{sk} \leftarrow \mathrm{SimSFKeyT3}\left(\widetilde{\mathrm{pp}}_{\tilde{\kappa}}, \widetilde{\mathrm{msk}}, \mathrm{kInd}, Y_2 Y_3, \alpha'\right) \end{array}\right],
\end{aligned}
$$

where $\widetilde{\mathbb{GD}}_{\tilde{N}}$ is the restriction of $\widetilde{\mathbb{GD}}$.

Furthermore, SimSFKeyT3 *sets the $\mathbb{G}_{p_2}$ component of $\boldsymbol{K}$ in* $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$ *to* $Y_2^{\boldsymbol{k}\left(\alpha', \boldsymbol{0}, \boldsymbol{0}\right)}$, *where* $(\boldsymbol{k}, \_) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$.

Proof. The algorithm SimSFKeyT3 is as follows.

Algorithm 4: SimSFKeyT3 - simulation of semi-functional keys of Type 3

Input : $(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}, Y_2 Y_3, \alpha')$.
Require : $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$.
1 Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\kappa, \mathrm{kInd})$. Let $m_1 := |\boldsymbol{k}|$.
2 Pick $\boldsymbol{r}' \leftarrow \mathbb{Z}_N^{m_2}$, $\boldsymbol{R}_3' \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute

$$\boldsymbol{K} := g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}\right)} \cdot (Y_2 Y_3)^{\boldsymbol{k}\left(\alpha', \boldsymbol{0}, \boldsymbol{0}\right)} \cdot \boldsymbol{R}_3' .$$

Output : $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$.

SimSFKeyT3 is a ppt algorithm with respect to λ by construction. Random elements from $\mathbb{G}_{p_3}$ can be sampled using $g_3 \in \mathrm{pp}_\kappa$. The element $g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}\right)}$ can be computed given $\boldsymbol{k}$, msk, $g_1^{\boldsymbol{h}} \in \mathrm{pp}_\kappa$, and $\boldsymbol{r}'$ as shown in Lemma F.1. The elements from $\boldsymbol{k}(\alpha', \boldsymbol{0}, \boldsymbol{0})$ can be computed explicitly given $\boldsymbol{k}$ and α'.

We will only prove the first equation, since the second equation is then implied by Statement 1 of Lemma D.3.

Let λ, $\mathrm{des} \in \Omega$, $(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \_, \_, \_) \in \left[\mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right)\right]$, $\mathrm{kInd} \in \mathbb{X}_\kappa$, $Y_2 \in \mathbb{G}_{p_2}$, $Y_2 \neq 1_{\mathbb{G}}$, and $Y_3 \in \mathbb{G}_{p_3}$ be arbitrary but fixed. Let $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, and $m_1 = |\boldsymbol{k}|$. Furthermore, since g_2, Y_2 are both generators of $\mathbb{G}_{p_2}$, there exists $x \in \mathbb{Z}_{p_2}^*$ such that $Y_2 = g_2^x$.

The first probability space is determined by $\hat{\alpha}$ and by the random variables $\boldsymbol{r} \leftarrow \mathbb{Z}_{p_1}^{m_2}$, and $\boldsymbol{R}_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$ defined by SFKeyGen (or rather KeyGen as a sub algorithm of SFKeyGen). Vector $\boldsymbol{r}$ is uniformly distributed over $\mathbb{Z}_{p_1}^{m_2}$ whereas vector $\boldsymbol{R}_3$ is uniformly distributed over $\mathbb{G}_{p_3}^{m_1}$. The output of SFKeyGen is a secret key $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$ such that:

$$\boldsymbol{K} = g_1^{\boldsymbol{k}(\mathrm{msk}, \boldsymbol{r}, \boldsymbol{h})} \cdot g_2^{\boldsymbol{k}(\hat{\alpha}, \boldsymbol{0}, \boldsymbol{0})} \cdot \boldsymbol{R}_3 .$$

Note, that the $\mathbb{G}_{p_2}$ components of $\boldsymbol{K}$ are fixed by the input values.

Now, consider SimSFKeyT3 in the context of the second probability distribution. By construction of SimSFKeyT3, it outputs $\mathrm{sk} = (\mathrm{kInd}, \boldsymbol{K})$, where $\boldsymbol{K} \in \mathbb{G}^{m_1}$ and it holds

$$\begin{aligned}\boldsymbol{K} &= g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r}',\boldsymbol{h}\right)} \cdot (Y_2 Y_3)^{\boldsymbol{k}\left(\alpha',\boldsymbol{0},\boldsymbol{0}\right)} \cdot \boldsymbol{R}_3' \\ &= g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r}',\boldsymbol{h}\right)} \cdot Y_2^{\boldsymbol{k}\left(\alpha',\boldsymbol{0},\boldsymbol{0}\right)} \cdot Y_3^{\boldsymbol{k}\left(\alpha',\boldsymbol{0},\boldsymbol{0}\right)} \cdot \boldsymbol{R}_3' \\ &= g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r}',\boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{k}\left(x\cdot\alpha',\boldsymbol{0},\boldsymbol{0}\right)} \cdot Y_3^{\boldsymbol{k}\left(\alpha',\boldsymbol{0},\boldsymbol{0}\right)} \cdot \boldsymbol{R}_3' \ .\end{aligned}$$

The $\mathbb{G}_{p_1}$ components of the group elements in $\boldsymbol{K}$ are computed as defined in the (semi-functional) key generation algorithm. The $\mathbb{G}_{p_2}$ components are properly distributed, since $\hat{\alpha}$ and $x \cdot \alpha'$ are identically distributed, due to the choice of $\hat{\alpha}$ and α', and since $x \neq 0 \pmod{p_2}$. Finally, the $\mathbb{G}_{p_3}$ components of $\boldsymbol{K}$ are properly distributed due to the choice of the group elements in $\boldsymbol{R}_3'$.

Furthermore, SimSFKeyT3 sets the $\mathbb{G}_{p_2}$ component of $\boldsymbol{K}$ to $Y_2^{\boldsymbol{k}\left(\alpha',\boldsymbol{0},\boldsymbol{0}\right)}$ as shown above in an intermediate step. This completes the proof. □

Difference-Lemma. The following general lemma will be used in almost all reduction steps of the main proof. See [20] for the proof of this lemma.

Lemma D.6. *(Difference Lemma) Let E_1, E_2 and F be events defined in a probability space, and suppose that $E_1 \wedge \neg F \Leftrightarrow E_2 \wedge \neg F$. Then $|\Pr[E_1] - \Pr[E_2]| \leq \Pr[F]$.*

D.3 Proof of the Main Theorem

In this section we provide the formal proof of our main theorem. As explained in Section 4, some of the reductions from the original CPA-secure framework of [3] require only few and simple modifications. For the sake of completeness we will present the complete proof and explain which parts of the proof are new.

Remark D.1. Formally, we have to show that the statement of our main theorem holds for every $\mathrm{des} \in \Omega$. However, we will not present different reduction algorithms for different description parameters des. Rather, the reduction algorithms in the proof will get des as an additional input.

From $\mathbf{G}_{\mathbf{Real}}$ to $\mathbf{G}_{\mathbf{resH}}$. The first game $\mathrm{G}_{\mathrm{Real}}$ in the proof sequence of probability experiments (see Fig. 1) is the CCA-security experiment $\text{aP-KEM}_{\Pi,\mathcal{A}}^{\text{aCCA}}(\lambda, \mathrm{des})$ from page 8. The restricted hash game $\mathrm{G}_{\mathrm{resH}}$ is defined as $\mathrm{G}_{\mathrm{Real}}$ except for the Guess phase, where the output $\langle 10 \rangle$ is modified. Recall that $\mathrm{HInput}(\cdot)$ is defined on page 11 as a part of the encapsulation algorithm. It takes as input an encapsulation and computes the corresponding input for the hash function. The last group element C'' of the encapsulation does not affect the hash input.

> Changes in $\mathrm{G}_{\mathrm{resH}}$ compared to $\mathrm{G}_{\mathrm{Real}}$:
> Exchange $\langle 10 \rangle$ for:
> 1. The output is 0, if $\mathcal{A}$ queried the decapsulation of $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ for some $\mathrm{cInd} \in \mathbb{Y}_\kappa$ such that
>
> $$\mathrm{HInput}(\mathrm{CT}) \neq \mathrm{HInput}(\mathrm{CT}^*) \qquad \text{and} \qquad t = t^* \pmod{N} \ ,$$
>
> where CT^* is the challenge encapsulation, t and t^* are the hash values of CT and CT^* respectively.
> 2. Otherwise, the output is as defined in $\text{aP-KEM}_{\Pi,\mathcal{A}}^{\text{aCCA}}(\lambda, \mathrm{des})$.

We call by HashAbort the event that a query, as defined in Step 1 above, occurs. The probability for this event is negligible due to the collision-resistance of $\mathcal{H}$ as stated in the following lemma.

Lemma D.7. *For every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ and every $\mathrm{des} \in \Omega$ it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{Real}}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resH}}}(\lambda, \mathrm{des})\right| \leq \mathrm{Adv}_{\mathcal{H},\mathcal{B}}^{\mathrm{CR}}(\lambda, \mathrm{des}) \ .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$, that can distinguish between $\mathrm{G}_{\mathrm{Real}}$ and $\mathrm{G}_{\mathrm{resH}}$, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks the security property of the collision-resistant hash function family $\mathcal{H}$.

Let λ and $\mathrm{des} \in \Omega$ be arbitrary, but fixed. Both probability experiments are identical as long as the event HashAbort does not occur. Hence, by Lemma D.6 it holds

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{Real}}}(\lambda,\mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resH}}}(\lambda,\mathrm{des})\right| \leq \Pr\left[\mathrm{HashAbort}\right] \quad .$$

But if this event occurs, we get a collision (x_1, x_2) for H:

$$x_1 = \mathrm{HInput}(\mathrm{CT}) \neq \mathrm{HInput}(\mathrm{CT}^*) = x_2 \quad ,$$
$$\mathrm{H}(x_1) = t = t^* = \mathrm{H}(x_2) \quad ,$$

which violates the security property of $\mathcal{H}$. More formally, we can construct a ppt algorithm $\mathcal{B}$ against $\mathcal{H}$ as follows. $\mathcal{B}$ on input $\left(1^\lambda, \mathbb{GD}, \mathrm{des}, s\right)$ (as defined in experiment $\mathrm{CR}_{\mathcal{H},\mathcal{A}}(\lambda,\mathrm{des})$ on page 19) simulates $\mathcal{A}$ using $\mathrm{H} := \mathrm{H}_{\lambda,\mathbb{GD}_N,\mathrm{des},s}$ and the public parameters generated using $\mathbb{GD}$ as defined in the experiment. If HashAbort occurs, $\mathcal{B}$ outputs the collision (x_1, x_2) from above for H and wins its experiment. We deduce $\mathrm{Adv}_{\mathcal{H},\mathcal{B}}^{\mathrm{CR}}(\lambda,\mathrm{des}) = \Pr[\mathrm{HashAbort}]$. This completes the proof. □

From $\mathbf{G}_{\mathbf{resH}}$ to $\mathbf{G}_{\mathbf{resQ}}$. The game with restricted queries $\mathrm{G}_{\mathrm{resQ}}$ is defined as $\mathrm{G}_{\mathrm{resH}}$ except for the Guess phase, where we again modify the output $\langle 10 \rangle$. We keep the modification from $\mathrm{G}_{\mathrm{resQ}}$ and add two additional checks:

Changes in $\mathrm{G}_{\mathrm{resQ}}$ compared to $\mathrm{G}_{\mathrm{resH}}$:
Exchange $\langle 10 \rangle$ for:
1. The same as Step 1 in $\mathrm{G}_{\mathrm{resH}}$.
2. The output is 0, if $\mathcal{A}$ queried the covered key generation oracle in Phase I or in Phase II on key index kInd with
$$\mathrm{Factor}(\kappa, \mathrm{kInd}, \mathrm{cInd}^*) = F \neq \bot \quad ,$$
where Factor is the algorithm from the domain-transferability property of $\mathcal{R}$ (see Definition 2.2).
3. The output is 0, if $\mathcal{A}$ queried the decapsulation oracle on $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ for some $\mathrm{cInd} \in \mathbb{Y}_\kappa$ such that for the corresponding hash value t it holds
$$t \neq t^* \pmod N \qquad \text{and} \qquad \gcd(t - t^*, N) \neq 1 \quad ,$$
where t^* is the hash value for the challenge CT^*.
4. Otherwise, the output is as defined in $\mathrm{aP\text{-}KEM}_{\Pi,\mathcal{A}}^{\mathrm{aCCA}}(\lambda,\mathrm{des})$ (the same as Step 2 in $\mathrm{G}_{\mathrm{resH}}$).

We call by FactorAbort the event, that the output of game $\mathrm{G}_{\mathrm{resQ}}$ is defined to be 0 by Step 2 or Step 3 from above. The probability for this event is negligible, since in both cases we can compute a non-trivial factor of N, which violates Assumption SD2 by Lemma 2.1 as stated in the following lemma.

Lemma D.8. *For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resH}}}(\lambda,\mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resQ}}}(\lambda,\mathrm{des})\right| \leq \mathrm{Adv}_{\mathcal{B}}^{\mathrm{SD2}}(\lambda) \quad .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$, that can distinguish between $\mathrm{G}_{\mathrm{resH}}$ and $\mathrm{G}_{\mathrm{resQ}}$, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2. $\mathcal{B}$ is given $\mathrm{des} \in \Omega$ in addition as explained in Remark D.1.

Let λ and $\mathrm{des} \in \Omega$ be arbitrary, but fixed. Experiments $\mathrm{G}_{\mathrm{resH}}$ and $\mathrm{G}_{\mathrm{resQ}}$ are identical as long as the event FactorAbort does not occur. Hence, by Lemma D.6 it holds

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resH}}}(\lambda,\mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resQ}}}(\lambda,\mathrm{des})\right| \leq \Pr\left[\mathrm{FactorAbort}\right] \quad .$$

Next we will analyze the probability for this event. We construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and wins in Experiment SD2 if the event FactorAbort occurs. $\mathcal{B}$ is given $\mathrm{des} \in \Omega$ in addition.

Algorithm 5: $\mathcal{B}$ against Assumption SD2

Input : (D, Z, des).

Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Compute $(\mathrm{msk}, \mathrm{pp}_\kappa, \_, \_, \_) \leftarrow \mathrm{SimPP}(\mathrm{des}, \mathbb{GD}_N, g_1, g_3)$ and simulate $\mathcal{A}$ as defined in the experiment $\mathrm{G_{resH}}$ until its output.

Perform Step 1 from the Guess Phase. Output a guess $\nu \leftarrow \{0,1\}$ if the result of the experiment is defined to be 0 in this step.

(Instead of Step 2) For every kInd_i used as input for the covered key generation oracle in Phase I or in Phase II compute $F_i := \mathrm{Factor}(\kappa, \mathrm{kInd}_i, \mathrm{cInd}^*)$.

if *there exists* F_i *such that* $F_i \neq \perp$ **then**

  Break the own challenge as shown in the proof of Lemma 2.1 using (D, Z, F_i).

(Instead of Step 3) For every decapsulation query on $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ compute the corresponding hash value t and check if $t \neq t^* \pmod N$ and $\gcd(t - t^*, N) \neq 1$, where t^* is the hash value for the challenge CT^*.

if t *with required property is found* **then**

  Compute a factor $F = \gcd(t - t^*, N)$ of N and break the own challenge as shown in the proof of Lemma 2.1, using (D, Z, F).

Output a guess $\nu \leftarrow \{0,1\}$.

In the first step, $\mathcal{B}$ generates properly distributed public parameters and the corresponding master secret key by Lemma D.3. Hence, $\mathcal{B}$ can simulate the adversary as defined in the experiment. Note, that the challenge Z of $\mathcal{B}$ is not used in the simulation of $\mathcal{A}$.

If the event FactorAbort does not occur, $\mathcal{B}$ outputs a guess and hence, it outputs 1 with probability $1/2$ independently of the value Z. If the event FactorAbort occurs, $\mathcal{B}$ computes a non-trivial factor of N and breaks Experiment SD2 with success probability 1 by Lemma 2.1.

Formally, for every $\mathrm{des} \in \Omega$ and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{B}'}(\lambda) &= \left|\Pr[\mathcal{B}'(D, Z_0) = 1] - \Pr[\mathcal{B}'(D, Z_1) = 1]\right| \\
&= \Pr[\mathrm{FactorAbort}] \cdot \left|\Pr[\mathcal{B}_{\mathcal{A}}(D, Z_0, \mathrm{des}) = 1 \mid \mathrm{FactorAbort}]\right. \\
&\qquad\qquad\qquad\qquad\left. - \Pr[\mathcal{B}_{\mathcal{A}}(D, Z_1, \mathrm{des}) = 1 \mid \mathrm{FactorAbort}]\right| \\
&= \Pr[\mathrm{FactorAbort}] \cdot \left|\Pr[\mathcal{B}''(D, Z_0, F) = 1] - \Pr[\mathcal{B}''(D, Z_1, F) = 1]\right| \\
&= \Pr[\mathrm{FactorAbort}] \quad ,
\end{aligned}$$

where $\mathcal{B}''$ is the algorithm from Lemma 2.1. This proves the lemma. □

Supplementary corollaries. One can efficiently check if the event HashAbort or the event FactorAbort occurs, as shown in the definition of $\mathrm{G_{resH}}$ and in the definition of $\mathrm{G_{resQ}}$. In the following experiments, the output will be 0 if one of these events occurs. Equivalently, we can assume that these events never happen. We obtain the following corollaries.

Corollary D.1. *Suppose that events* HashAbort *and* FactorAbort *do not occur. Then, for every* $p_i \mid N$ *and every encapsulation* CT*, used by* $\mathcal{A}$ *as input for the decapsulation oracle, it holds*

$$\mathrm{HInput}(\mathrm{CT}) \neq \mathrm{HInput}(\mathrm{CT}^*) \qquad \textit{implies} \qquad t \neq t^* \pmod{p_i} \ ,$$

where $t = \mathrm{H}(\mathrm{HInput}(\mathrm{CT}))$ *and* $t^* = \mathrm{H}(\mathrm{HInput}(\mathrm{CT}^*))$.

Proof. By the definition of H it holds $t, t^* \in \mathbb{Z}_N$. If the event HashAbort does not occur, it holds $t \neq t^* \pmod N$ for every CT which satisfies $\mathrm{HInput}(\mathrm{CT}) \neq \mathrm{HInput}(\mathrm{CT}^*)$. W.l.o.g. let $0 < t - t^* < N$ and $p_i \mid N$ be arbitrary but fixed. Assume that $t = t^* \pmod{p_i}$. Then we deduce that $\gcd(t - t^*, N) \geq p_i > 1$, which is the FactorAbort event (Step 3). Hence, it holds $t \neq t^* \pmod{p_i}$ for every CT which satisfies $\mathrm{HInput}(\mathrm{CT}) \neq \mathrm{HInput}(\mathrm{CT}^*)$. □

Corollary D.2. *Suppose that event* FactorAbort *does not occur. Then, for every* kInd*, used by $\mathcal{A}$ in covered key generation queries, it holds*

$$\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}^*) = 0 \qquad \textit{implies} \qquad \mathrm{R}_{p_2}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd}^*)) = 0 \ ,$$

where f_1 and f_2 are the projection maps from the domain-transferability property of $\mathcal{R}$.

Proof. The implication is guaranteed by Step 2 in the Guess phase from the definition of $\mathrm{G}_{\mathrm{resQ}}$, since otherwise algorithm Factor outputs a non-trivial factor F of N. □

Together with the properties of selective master key hiding and co-selective master key hiding of the underlying pair encoding schemes, Corollary D.2 is crucial for the analysis of the reduction steps leading from $\mathrm{G}_{k,1}$ to $\mathrm{G}_{k,2}$ and from G_{q_1+1} to G_{q_1+2}. In turn, Corollary D.1 is crucial for our last additional reduction where we prove that G_{q_1+3} and G'_{q_1+3} are indistinguishable.

From $\mathbf{G}_{\mathbf{resQ}}$ to $\mathbf{G}'_{\mathbf{0}}$. The experiment G'_0 is as $\mathrm{G}_{\mathrm{resQ}}$, but the challenge encapsulation is semi-functional:

Changes in G'_0 compared to $\mathrm{G}_{\mathrm{resQ}}$:
Exchange $\langle 1 \rangle$ for:

$$\left(\mathrm{msk}, \mathrm{pp}_\kappa, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \leftarrow \mathrm{SFSetup}\left(1^\lambda, \mathrm{des}\right) \ .$$

Exchange $\langle 5 \rangle$ for:

$$(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SFEncaps}\left(\mathrm{cInd}^*, g_2, \hat{\boldsymbol{h}}, \hat{u}_2, \hat{v}_2\right) \ .$$

The following lemma corresponds to Lemma 28 from [3]. We use our supplementary algorithms from Subsection D.2, which simplifies the description of the reduction algorithm and the proof. Furthermore, we extended the algorithm by the computation of our additional elements.

Lemma D.9. *(cf. Lemma 28 in [3]) For every* des $\in \Omega$ *and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_{\mathrm{resQ}}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}'_0}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}^{\mathrm{SD1}}_{\mathcal{B}}(\lambda) \ .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$, that can distinguish between $\mathrm{G}_{\mathrm{resQ}}$ and G'_0, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD1 with the same success probability. Let des $\in \Omega$ be arbitrary but fixed. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD1 as explained in Remark D.1 and is as follows.
$\mathcal{B}$ is a ppt algorithm with respect to λ by construction. In particular, all exponents in the description of the algorithm can be computed explicitly.

Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$. By construction of $\mathcal{B}$ and by Statement 1 of Lemma D.3 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and the master secret msk are distributed as defined in the experiments. Note, that $\mathbb{GD}_N$, g_1 and g_3 are distributed as required by Lemma D.3 due to the definition of experiment SD1. Furthermore, by Statement 2 of Lemma D.3 it holds $g_1^{\boldsymbol{h}'} = g_1^{\boldsymbol{h}}$, $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{\boldsymbol{h}} = \boldsymbol{h}' \pmod{p_2}$, $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma D.3.

All secret keys generated in Phase I and in Phase II are normal in both experiments and hence, by construction of $\mathcal{B}$ are correctly generated using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$. Let g_2 be an arbitrary but fixed generator of $\mathbb{G}_{p_2}$. By the definition of probability experiment SD1 it holds $Z = g_1^{z_1} g_2^{z_2}$, where z_1 is uniformly distributed in $\mathbb{Z}^*_{p_1}$, and z_2 is either uniformly distributed in $\mathbb{Z}^*_{p_2}$ (if $Z = Z_1$) or $z_2 = 0 \pmod{p_2}$ (if $Z = Z_0$).

Next, consider the challenge phase and the generated challenge. It is important to notice, that by Lemma D.2, we can consider the distribution of the encapsulation for any fixed generator of $\mathbb{G}_{p_2}$. By

Algorithm 6: $\mathcal{B}$ against Assumption SD1

Input : (D, Z, des).

Require: $D = (\mathbb{GD}_N, g_1, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup
- Compute $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v) \leftarrow \mathrm{SimPP}(\mathrm{des}, \mathbb{GD}_N, g_1, g_3)$ and simulate $\mathcal{A}$ on input $\left(1^\lambda, \mathrm{pp}_\kappa\right)$.

Phase I
- Simulate this phase as defined in the experiments using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$ to generate the keys.

Challenge (given $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$ from $\mathcal{A}$)
- Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$.
- Pick $s' \leftarrow \mathbb{Z}_N$, $\boldsymbol{s}' \leftarrow \mathbb{Z}_N^{w_2}$ and compute
$$\boldsymbol{C}^* := Z^{\boldsymbol{c}\left(s', \boldsymbol{s}', \boldsymbol{h}'\right)} .$$
- Compute $t^* := \mathrm{H}(\mathrm{HInput}(\mathrm{cInd}^*, \boldsymbol{C}^*, \_))$ and
$$C''^* := Z^{s' \cdot \left(u \cdot t^* + v\right)} .$$
- Set $\mathrm{CT}^* := (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*)$ and $\mathrm{K}_0 := \mathrm{e}(Z, g_1)^{\mathrm{msk} \cdot s'}$.
- Pick $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, flip a coin $b \leftarrow \{0,1\}$, set $\mathrm{K}^* := \mathrm{K}_b$, and return the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.

Phase II
- Simulate this phase as defined in the experiments using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$ to generate the keys.

Guess
- Simulate this phase as defined in the experiment.

construction of $\mathcal{B}$ it holds

$$\begin{aligned}
\mathrm{K}_0 &= \mathrm{e}(Z, g_1)^{\mathrm{msk} \cdot s'} & \boldsymbol{C}^* &= Z^{\boldsymbol{c}\left(s', \boldsymbol{s}', \boldsymbol{h}'\right)} & C''^* &= Z^{s' \cdot (u \cdot t^* + v)} \\
&= \mathrm{e}\left(g_1^{z_1} g_2^{z_2}, g_1\right)^{\mathrm{msk} \cdot s'} & &= g_1^{z_1 \cdot \boldsymbol{c}\left(s', \boldsymbol{s}', \boldsymbol{h}'\right)} \cdot g_2^{z_2 \cdot \boldsymbol{c}\left(s', \boldsymbol{s}', \boldsymbol{h}'\right)} & &= \left(g_1^{z_1} g_2^{z_2}\right)^{s' \cdot (u \cdot t^* + v)} \\
&= Y^{z_1 \cdot s'} , & &= g_1^{\boldsymbol{c}\left(z_1 \cdot s', z_1 \cdot \boldsymbol{s}', \boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{c}\left(z_2 \cdot s', z_2 \cdot \boldsymbol{s}', \hat{\boldsymbol{h}}\right)} , & &= \left(U_1^{t^*} V_1\right)^{z_1 \cdot s'} \cdot \left(g_2^{\hat{u}_2 \cdot t^*} g_2^{\hat{v}_2}\right)^{z_2 \cdot s'} .
\end{aligned}$$

We claim that $\mathrm{CT}^* = (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*)$ is a properly distributed encapsulation of K_0, which is either normal (if $Z = Z_0$) or semi-functional, as defined on page 12 (if $Z = Z_1$). Namely, $\mathcal{B}$ implicitly sets the random values of the normal ($\mathbb{G}_{p_1}$) components as $s := z_1 \cdot s' \pmod{p_1}$, and $\boldsymbol{s} := z_1 \cdot \boldsymbol{s}' \pmod{p_1}$. The random values of semi-functional ($\mathbb{G}_{p_2}$) components are set as $\hat{s} := z_2 \cdot s' \pmod{p_2}$, $\hat{\boldsymbol{s}} := z_2 \cdot \boldsymbol{s}' \pmod{p_2}$. These values are properly distributed due to the choice of s' and $\boldsymbol{s}'$. Thereby, we use the fact that $z_1 \neq 0 \pmod{p_1}$ in both cases and $z_2 \neq 0 \pmod{p_2}$ in the case of $Z = Z_1$. Furthermore, the value s' and all values in $\boldsymbol{s}'$ modulo p_1 and modulo p_2 are uncorrelated by the Chinese Remainder Theorem.

We deduce that $\mathcal{B}$ perfectly simulates experiment $\mathrm{G}_{\mathrm{resQ}}$ if $Z = Z_0$ and experiment G_0 if $Z = Z_1$. Furthermore, the output of $\mathcal{B}$ is 1 if and only if $\mathcal{A}$ wins in the corresponding experiment. Hence, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_\mathcal{A}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD1}}(\lambda) &= \left|\Pr[\mathcal{B}'(D, Z_0) = 1] - \Pr[\mathcal{B}'(D, Z_1) = 1]\right| \\
&= \left|\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resQ}}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_0'}(\lambda, \mathrm{des})\right)\right| \\
&= \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{resQ}}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_0'}(\lambda, \mathrm{des})\right| .
\end{aligned}$$

This proves the lemma. □

From G_0' to $\mathrm{G}_{0,3}$. The main modification in $\mathrm{G}_{0,3}$ is that the decapsulation queries are answered using separately generated normal keys which we denote by sk_i'. Hence, the keys, generated in the covered key generation queries and denoted by sk_i, will be used only in the opening oracle. Consequently, we do not have to generate these keys in the covered key generation queries anymore, instead they are generated in the opening oracle, when the keys are given to the adversary. This last change is only conceptual at this point, but is crucial for the following reductions and the resulting security guaranties.

> Changes in $\mathrm{G}_{0,3}$ compared to G_0':
> **CoveredKeyGen** (kInd_i) - Do not generate keys in $^{\langle 2\rangle}$ and in $^{\langle 7\rangle}$, just store (i, kInd_i).
> **Open** (i) - Instead of $^{\langle 3\rangle}$ and $^{\langle 8\rangle}$, generate $^{\langle 13\rangle}$ $\boxed{\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)}$ if sk_i was not generated yet. Return sk_i.
> **Decapsulate** (CT, i) instead of $^{\langle 4\rangle}$ and $^{\langle 9\rangle}$:
>
> – For the first decapsulation query with index i generate and store $\mathrm{sk}_i' \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$.
> – Return $\mathrm{Decaps}\left(\mathrm{sk}_i', \mathrm{CT}\right)$.

The following lemma is new in the original sequence of probability experiments (see Fig. 1). We show that because of the consistency checks and especially because of the verifiability property of the underlying pair encoding scheme both experiments are unconditionally indistinguishable.

Lemma D.10. *For every security parameter λ, every $\mathrm{des} \in \Omega$ and every algorithm $\mathcal{A}$ it holds*

$$\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_0'}(\lambda, \mathrm{des}) = \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_0}(\lambda, \mathrm{des}) \quad .$$

Proof. All generated keys are normal in both experiments by definition. By construction, the view of $\mathcal{A}$ in both experiments can only differ if there is a decapsulation query on CT and $i \in \mathbb{N}$ such that the probability distributions defined by $\mathrm{Decaps}(\mathrm{sk}_i, \mathrm{CT})$ and $\mathrm{Decaps}\left(\mathrm{sk}_i', \mathrm{CT}\right)$ are not equal. This can not happen due to Lemma 3.1, which proves the lemma. □

Remark D.2. As mentioned above, all reduction steps between $\mathrm{G}_{0,3}$ and G_{q_1+3} are similar to the original CPA-secure construction of [3]. In all reduction steps between these two experiments the reduction algorithm knows the master secret key. Hence, all normal keys used to answer decapsulation queries can be generated using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$. Furthermore, as mentioned before, we have to show that the additional element C''^* for the challenge encapsulation can be generated. For those steps, which are based on the subgroup decision assumptions, this is already covered by Lemma D.4 and by the algorithm SimSFChlg. The steps based on the security properties of the underlying pair encoding schemes require further explanations (Lemma D.12 and Lemma D.15). For the sake of completeness we present all reductions using our supplementary algorithms.

From $\mathbf{G}_{k-1,3}$ to $\mathbf{G}_{k,1}$ for $k \in [q_1]$. Experiment $\mathrm{G}_{k-1,3}$ is defined as experiment $\mathrm{G}_{0,3}$, but the first $k-1$ keys, corrupted in Phase I, are semi-functional of Type 3. Hence, $\mathrm{G}_{0,3}$ is a special case of $\mathrm{G}_{k-1,3}$ for $k = 1$. The following experiments include an index $j \in \mathbb{N}$, which denotes the current number of corrupted keys in Phase I.

> Experiment $\mathrm{G}_{k-1,3}$ for $k \in [q_1]$ as generalization of $\mathrm{G}_{0,3}$:
>
> – Set $j := 0$ in the Setup phase.
>
> **Open** $(i)^a$: Exchange $^{\langle 13\rangle}$ in Phase I (defined in $\mathrm{G}_{0,3}$ on page 36) for:
>
> – Set $j := j + 1$;
> – If $j < k$, choose $\hat{\alpha}_j \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 3, \hat{\alpha}_j, g_2, \hat{\boldsymbol{h}}\right)$.
> – If $j \geq k$, return $\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$.
>
> [a] W.l.o.g. assume that $\mathcal{A}$ never asks for the same index. Otherwise, just store the keys.

$\mathrm{G}_{k,1}$ is as $\mathrm{G}_{k-1,3}$, but the k's key corrupted in the first phase is semi-functional of Type 1:

> Changes in $\mathrm{G}_{k,1}$ compared to $\mathrm{G}_{k-1,3}$:
> **Open** (i): modify $^{\langle 13\rangle}$ in Phase I by
>
> – If $j = k$, return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 1, \_, g_2, \hat{\boldsymbol{h}}\right)$.

Algorithm 7: $\mathcal{B}$ against Assumption SD2

Input : (D, Z, des).

Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup

  Compute $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v) \leftarrow \mathrm{SimPP}(\mathbb{GD}_N, g_1, g_3, \mathrm{des})$.

  Set $j := 0$.

Phase I

  CoveredKeyGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:

    Store (i, kInd_i).

  Open (i) with $i \in \mathbb{N}$:

    Set $j := j + 1$.

    case $j < k$ **do**

      Pick $\alpha'_j \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SimSFKeyT3}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i, Y_2Y_3, \alpha_j)$.

    case $j = k$ **do**

      Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\mathrm{kInd}_i)$. Let $m_1 := |\boldsymbol{k}|$.

      Pick $\boldsymbol{r}', \hat{\boldsymbol{r}}' \leftarrow \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}'_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute

$$\boldsymbol{K} := g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}\right)} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3 \ .$$

      Return $\mathrm{sk}_i := (\mathrm{kInd}_i, \boldsymbol{K})$.

    case $j > k$ **do**

      Return $\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$.

  Decapsulate (CT, i) with $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$, $i \in \mathbb{N}$:

    As defined in the experiment using normal secret key $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$, generated once.

Challenge (given $\mathrm{cInd}^* \in \mathbb{Y}_\kappa$ from $\mathcal{A}$)

  Generate $(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SimSFChlg}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}^*, \boldsymbol{h}', u, v, X_1X_2)$

  Pick $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, flip a coin $b \leftarrow \{0,1\}$, set $\mathrm{K}^* := \mathrm{K}_b$, and return the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.

Phase II

  Simulates this phase as defined in the experiment using msk.

Guess

  Simulate this phase as defined in the experiment

Lemma D.11. *(cf. Lemma 29 in [3]) Suppose $q_1 \in \mathbb{N}$ is the upper bound for the number of corrupted keys in Phase I. Let $k \in [q_1]$ be arbitrary. For every $\mathrm{des} \in \Omega$ and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k-1,3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}_{\mathcal{B}}^{\mathrm{SD2}}(\lambda) \ .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Let $k \in [q_1]$ be arbitrary, but fixed. Given a ppt adversary $\mathcal{A}$, that can distinguish between $\mathrm{G}_{k-1,3}$ and $\mathrm{G}_{k,1}$, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 with the same success probability. Let $\mathrm{des} \in \Omega$ be arbitrary. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD2 as explained in Remark D.1 and is as follows.
$\mathcal{B}$ is a ppt algorithm by construction. In particular, random elements from $\mathbb{G}_{p_3}$ can be chosen using g_3, the elements from $g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}\right)}$ can be computed as shown in Lemma F.1, and $\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)$ can be computed explicitly.

Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$. By construction of $\mathcal{B}$ and by Statement 1 of Lemma D.3 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and the master secret msk are distributed as defined in the experiments. Note, that $\mathbb{GD}_N$, g_1 and g_3 are distributed as required by Lemma D.3 due to the definition of experiment SD2. Furthermore, by Statement 2 of Lemma D.3 it holds $g_1^{\boldsymbol{h}'} = g_1^{\boldsymbol{h}}$, $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{\boldsymbol{h}} = \boldsymbol{h}' \pmod{p_2}$, $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma D.3.

Let g_2 be an arbitrary but fixed generator of $\mathbb{G}_{p_2}$. Then, by the definition of probability experiment SD2 it holds $Z = g_1^{z_1} g_2^{z_2} g_3^{z_3}$, where $z_1 \in \mathbb{Z}_{p_1}^*$, $z_3 \in \mathbb{Z}_{p_3}^*$ are uniformly distributed, and z_2 is either uniformly distributed in $\mathbb{Z}_{p_2}^*$ (if $Z = Z_1$) or $z_2 = 0 \pmod{p_2}$ (if $Z = Z_0$). Furthermore, $X_1 = g_1^{x_1}$, $X_2 = g_2^{x_2}$, $Y_2 = g_2^{y_2}$ and $Y_3 = g_3^{y_3}$, where $x_1 \in \mathbb{Z}_{p_1}^*$, $x_2, y_2 \in \mathbb{Z}_{p_2}^*$ and $y_3 \in \mathbb{Z}_{p_3}^*$ are uniformly distributed and independent.

The semi-functional challenge and all semi-functional keys of Type 3 are generated as required in the experiment by Lemma D.4, and by Lemma D.5 respectively. The normal keys are correctly generated using $\mathrm{KeyGen}(\mathrm{msk}, \cdot)$.

Consider the corrupted key $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$ generated for $j = k$. By construction of the algorithm $\mathcal{B}$ it holds:

$$\begin{aligned}
\boldsymbol{K} &= g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}\right)} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}\right)} \cdot \left(g_1^{z_1} g_2^{z_2} g_3^{z_3}\right)^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}\right) + z_1 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot g_2^{z_2 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}' + z_1 \cdot \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{k}\left(0, z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right)} \cdot g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \quad .
\end{aligned}$$

We claim, that sk_i is either a properly distributed normal secret key (if $Z = Z_0$) or a properly distributed semi-functional secret key of Type 1 (if $Z = Z_1$). Namely, $\mathcal{B}$ implicitly sets the random values of the normal ($\mathbb{G}_{p_1}$) components as $\boldsymbol{r} = z_1 \cdot \hat{\boldsymbol{r}}' + \boldsymbol{r}' \pmod{p_1}$, which are properly distributed due to the choice of $\boldsymbol{r}'$. The random values of the semi-functional ($\mathbb{G}_{p_2}$) components are set as $\hat{\boldsymbol{r}} = z_2 \cdot \hat{\boldsymbol{r}}'$, which are properly distributed (for Type 1 keys) due to the choice of $\hat{\boldsymbol{r}}'$ if $Z = Z_1$, and which disappear if $Z = Z_0$, since $z_2 = 0 \pmod{p_2}$. Finally, the $\mathbb{G}_{p_3}$ components are set as $\boldsymbol{R}_3 = g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot \boldsymbol{R}_3'$ and are properly distributed by the choice of $\boldsymbol{R}_3'$.

Hence, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD2}}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\
&= \left|\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k-1,3}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des})\right)\right| \\
&= \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k-1,3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des})\right| \quad .
\end{aligned}$$

The second equation holds since $\mathcal{B}$ perfectly simulates $\mathrm{G}_{k-1,3}$ and $\mathrm{G}_{k,1}$ if $Z = Z_0$ and $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. ◻

From $\mathrm{G}_{k,1}$ to $\mathrm{G}_{k,2}$ for $k \in [q_1]$. $\mathrm{G}_{k,2}$ is as $\mathrm{G}_{k,1}$, but the k's key is semi-functional of Type 2:

> Changes in $\mathrm{G}_{k,2}$ compared to $\mathrm{G}_{k,1}$:
> **Open** (i): modify $\langle 13 \rangle$ in Phase I (defined in $\mathrm{G}_{0,3}$) for the key with $j = k$
>
> – If $j = k$, choose $\hat{\alpha}_k \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 2, \hat{\alpha}_k, g_2, \hat{\boldsymbol{h}}\right)$.

Lemma D.12. *(cf. Lemma 30 in [3]) Suppose $q_1 \in \mathbb{N}$ is the upper bound for the number of corrupted keys in Phase I. Let $k \in [q_1]$ be arbitrary. For every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ and every* $\mathrm{des} \in \Omega$ *it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,1}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{k,2}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}_{\mathrm{P},\mathcal{B}}^{\mathrm{CMH}}(\lambda, \mathrm{des}) \quad .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Let $k \in [q_1]$ be arbitrary, but fixed. Given a ppt adversary $\mathcal{A}$, that can distinguish between $\mathrm{G}_{k,1}$ and $\mathrm{G}_{k,2}$, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks the co-selective master-key hiding security property of the underlining pair encoding scheme with the same advantage. $\mathcal{B}$ on input $(\mathrm{des}, \mathbb{GD}_N, g_1, g_2, g_3)$, as defined in $\mathrm{Exp}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}^{\mathrm{CMH}}(\lambda, \mathrm{des})$, is as follows.

$\mathcal{B}$ is a ppt algorithm with respect to λ by construction. It uses different supplementary ppt algorithms, the own oracles, and performs besides only simple computation. Next we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$.

Algorithm 8: $\mathcal{B}$ against co-selective master-key hiding security property of P

Input : $(\mathrm{des}, \mathbb{GD}_N, g_1, g_2, g_3)$.

Setup
- Compute the public parameters and the master secret key $(\mathrm{msk}, \mathrm{pp}_\kappa, \_, u, v) \leftarrow \mathrm{SimPP}\left(1^\lambda, \mathbb{GD}_N, g_1, g_3, \mathrm{des}\right)$. Set $j := 0$.

Phase I
- **CoveredKeyGen** (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
  - Store (i, kInd_i).
- **Open** (i)**:**
  - Set $j := j + 1$.
  - **case** $j < k$ **do**
    - Pick $\alpha_j \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}(\mathrm{msk}, \mathrm{kInd}_i, 3, \alpha_j, g_2, \_)$.
  - **case** $j = k$ **do**
    - Generate a normal key $(\mathrm{kInd}_i, \boldsymbol{K}) \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$.
    - Query the own oracle:
    $$\widehat{\boldsymbol{K}} := \mathcal{O}^1_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd}_i) \quad .$$
    - Output $\mathrm{sk}_i = \left(\mathrm{kInd}_i, \boldsymbol{K} \cdot \widehat{\boldsymbol{K}}\right)$.
  - **case** $j > k$ **do**
    - Output $\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$.
- **Decapsulate** (CT, i)**:**
  - As defined in the experiment using $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$, generated once.

Challenge (given cInd^* from $\mathcal{A}$)
- Compute $(\mathrm{K}_0, (\mathrm{cInd}^*, \boldsymbol{C}, \_)) \leftarrow \mathrm{Encaps}(\mathrm{cInd}^*)$.
- Query the own oracle:
$$\widehat{\boldsymbol{C}} := \mathcal{O}^2_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*) \quad ,$$
  set $\boldsymbol{C}^* := \boldsymbol{C} \cdot \widehat{\boldsymbol{C}}$.
- Compute $t^* := \mathrm{H}(\mathrm{HInput}(\mathrm{cInd}^*, \boldsymbol{C}^*, \_))$ and
$$C''^* := (C_1^*)^{u \cdot t^* + v} \quad .$$
- Choose $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, pick $b \leftarrow \{0,1\}$, set $\mathrm{K}^* := \mathrm{K}_b$ and return $(\mathrm{K}^*, \mathrm{CT}^*_{\mathrm{cInd}^*} = (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*))$.

Phase II
- Simulates this phase as defined in the experiment using msk.

Guess
- Simulate this phase as defined in the experiment

Let security parameter λ and $\mathrm{des} \in \Omega$ be arbitrary, but fixed. By the definition of the Experiment $\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$, $\mathbb{GD}_N$ is the restricted group description of $\mathbb{GD}$ generated by $\mathcal{G}\left(1^\lambda\right)$. Furthermore, the generators $g_i \in \mathbb{G}_{p_i}$ are chosen uniformly at random. Hence, by construction of $\mathcal{B}$ and by Statement 1 of Lemma D.3 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and the master secret msk are distributed as defined in the experiments. Furthermore, by Statement 2 of Lemma D.3 it holds $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma D.3. Furthermore, $\mathcal{B}$ implicitly sets the input generator g_2 as the generator of $\mathbb{G}_{p_2}$. This generator is properly distributed as mentioned above. Vector $\hat{\boldsymbol{h}} \pmod{p_2}$ of the semi-functional public parameters will be defined below.

It is important to notice that all oracle queries made by $\mathcal{B}$ are permissible if all corruption queries of $\mathcal{A}$ are permissible, since $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}^*) = 0$ implies $\mathrm{R}_{p_2}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd}^*)) = 0$ by Corollary D.2. The normal keys and the semi-functional keys of Type 3 are generated using msk and g_2 as defined in the experiments.

Next, we claim that the challenge encapsulation is a properly distributed semi-functional encapsulation. Furthermore, the k's corrupted key is either a properly distributed semi-functional key of Type 1 (if $\nu = 0$) or a properly distributed semi-functional key of Type 2 (if $\nu = 1$). Namely, by the definition of the Experiment $\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}(\lambda, \mathrm{des})$, the challenger choose $\tilde{\alpha} \leftarrow \mathbb{Z}_N$ and $\tilde{\boldsymbol{h}} \leftarrow \mathbb{Z}_N^n$, where $n = \mathrm{Param}(\kappa)$.

Then, $\mathcal{B}$ receives

$$\widehat{\boldsymbol{K}} = \mathcal{O}^1_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd}_i) = \begin{cases} g_2^{\boldsymbol{k}(0,\tilde{\boldsymbol{r}},\tilde{\boldsymbol{h}})} & \text{if } \nu = 0 \\ g_2^{\boldsymbol{k}(\tilde{\alpha},\tilde{\boldsymbol{r}},\tilde{\boldsymbol{h}})} & \text{if } \nu = 1 \end{cases} ,$$

where $\tilde{\boldsymbol{r}} \in \mathbb{Z}_{p_2}^{m_2}$ is chosen uniformly at random, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd}_i)$. Hence, $\widehat{\boldsymbol{K}}$ is a properly distributed semi-functional part of either Type 1 key (if $\nu = 0$) or Type 2 key (if $\nu = 1$) with random values $\hat{\boldsymbol{h}} = \tilde{\boldsymbol{h}} \pmod{p_2}$, $\hat{\boldsymbol{r}} = \tilde{\boldsymbol{r}} \pmod{p_2}$, and $\hat{\alpha} = \tilde{\alpha} \pmod{p_2}$. Furthermore, $\mathcal{B}$ receives

$$\widehat{\boldsymbol{C}} = \mathcal{O}^2_{\mathrm{CMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*) = g_2^{\boldsymbol{c}(\tilde{s},\tilde{\boldsymbol{s}},\hat{\boldsymbol{h}})} ,$$

where $\tilde{s} \in \mathbb{Z}_N$ and $\tilde{\boldsymbol{s}} \in \mathbb{Z}_N^{w_2}$ are chosen uniformly at random, $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$. Hence, $\widehat{\boldsymbol{C}}$ is a properly distributed semi-functional part of an encapsulation with random values $\hat{s} = \tilde{s} \pmod{p_2}$, $\hat{\boldsymbol{s}} = \tilde{\boldsymbol{s}} \pmod{p_2}$, and $\hat{\boldsymbol{h}} = \tilde{\boldsymbol{h}} \pmod{p_2}$.

Finally, we show that the last group element in the challenge encapsulation is correctly generated:

$$\begin{aligned} C''^* &= (C_1)^{u \cdot t^* + v} \cdot \left(\widehat{C}_1\right)^{u \cdot t^* + v} \\ &= (g_1^s)^{u \cdot t^* + v} \cdot (g_2^{\hat{s}})^{u \cdot t^* + v} \\ &= \left(U_1^{t^*} \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2 \cdot t^* + \hat{v}_2}\right)^{\hat{s}} , \end{aligned}$$

where s is the random element fixed by C_1, which is chosen as the first element of $\boldsymbol{C}$ in Line 19 and $\hat{s}$ is fixed by $\widehat{C}_1$ as defined above. In the second equation we used the normality of P. Hence, C''^* is exactly as defined in SFEncaps.

We deduce that for every $\mathcal{A}$, every security parameter λ and every $\mathrm{des} \in \Omega$ it holds

$$\begin{aligned} \mathrm{Adv}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{B}}(\lambda, \mathrm{des}) &= \left|\mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},0,\mathcal{B}}(\lambda, \mathrm{des}) - \mathrm{Exp}^{\mathrm{CMH}}_{\mathrm{P},\mathcal{G},1,\mathcal{B}}(\lambda, \mathrm{des})\right| \\ &= \left|\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{k,1}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{k,2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right)\right| \\ &= \left|\mathrm{Adv}^{\mathrm{G}_{k,1}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{k,2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| . \end{aligned}$$

The second equation holds since $\mathcal{B}$ correctly simulates $\mathrm{G}_{k,1}$ and $\mathrm{G}_{k,2}$ if $\nu = 0$ and if $\nu = 1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding game. This proves the lemma. □

From $\mathbf{G}_{k,2}$ to $\mathbf{G}_{k,3}$ for $k \in [q_1]$. $\mathrm{G}_{k,3}$ is as $\mathrm{G}_{k,2}$, but the k's corrupted key is semi-functional of Type 3.

Changes in $\mathrm{G}_{k,3}$ compared to $\mathrm{G}_{k,2}$: **Open** (i): modify $\langle 13 \rangle$ in Phase I (defined in $\mathrm{G}_{0,3}$) for the key with $j = k$ – If $j = k$, choose $\hat{\alpha}_k \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_k \leftarrow \mathrm{SFKeyGen}(\mathrm{msk}, \mathrm{kInd}_i, 3, \hat{\alpha}_k, g_2, \_)$.

Lemma D.13. *(cf. Lemma 31 in [3]) Suppose $q_1 \in \mathbb{N}$ is the upper bound for the number of corrupted keys in Phase I. Let $k \in [q_1]$ be arbitrary. For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ it holds*

$$\left|\mathrm{Adv}^{\mathrm{G}_{k,2}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{k,3}}_{\Pi,\mathcal{A}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}^{\mathrm{SD2}}_{\mathcal{A}}(\lambda) .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$, that can distinguish between $\mathrm{G}_{k,2}$ and $\mathrm{G}_{k,3}$, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 with the same advantage. Let $\mathrm{des} \in \Omega$ be arbitrary. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD2 as explained in Remark D.1.

$\mathcal{B}$ is almost the same as Algorithm 7. We only change the simulation of corrupted key number k:

Algorithm 9: $\mathcal{B}$ against Assumption SD2 as modification of Algorithm 7

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\text{des} \in \Omega$.

...
Phase I
  ...
  Open (i) with $i \in \mathbb{N}$:
    ...
    case $j = k$ **do**
      Compute $(\boldsymbol{k}, m_2) := \text{Enc1}(\text{kInd}_k)$. Let $m_1 := |\boldsymbol{k}|$.
      Pick $\boldsymbol{r}', \hat{\boldsymbol{r}}', \hat{\alpha}' \leftarrow \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}_3' \leftarrow \mathbb{G}_{p_3}^{m_1}$. Compute

$$\boldsymbol{K} := g_1^{\boldsymbol{k}\left(\text{msk}, \boldsymbol{r}', \boldsymbol{h}\right)} \cdot (Y_2Y_3)^{\boldsymbol{k}\left(\hat{\alpha}', \mathbf{0}, \mathbf{0}\right)} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \ .$$

      Return $\text{sk}_i = (\text{kInd}_i, \boldsymbol{K})$.
...

$\mathcal{B}$ is a ppt algorithm by construction. Next we will analyze the view of $\mathcal{A}$ and the success probability of algorithm $\mathcal{B}$.

Let all elements be as defined in the proof of Lemma D.11. In particular, $Z = g_1^{z_1} g_2^{z_2} g_3^{z_3}$, $Y_2 = g_2^{y_2}$, $Y_3 = g_3^{y_3}$, $\boldsymbol{h} = \boldsymbol{h}' \pmod{p_1}$ and $\hat{\boldsymbol{h}} = \boldsymbol{h}' \pmod{p_2}$. Here, we only analyze the distribution of the secret key $\text{sk}_i = (\text{kInd}_i, \boldsymbol{K})$ generated for $j = k$. By construction of $\mathcal{B}$ it holds

$$\begin{aligned}
\boldsymbol{K} &= g_1^{\boldsymbol{k}\left(\text{msk}, \boldsymbol{r}', \boldsymbol{h}\right)} \cdot (Y_2Y_3)^{\boldsymbol{k}\left(\hat{\alpha}', \mathbf{0}, \mathbf{0}\right)} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\text{msk}, \boldsymbol{r}', \boldsymbol{h}\right)} \cdot (g_2^{y_2} g_3^{y_3})^{\boldsymbol{k}\left(\hat{\alpha}', \mathbf{0}, \mathbf{0}\right)} \cdot (g_1^{z_1} g_2^{z_2} g_3^{z_3})^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\text{msk}, \boldsymbol{r}', \boldsymbol{h}\right) + z_1 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot g_2^{y_2 \cdot \boldsymbol{k}\left(\hat{\alpha}', \mathbf{0}, \mathbf{0}\right) + z_2 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right)} \cdot g_3^{y_3 \cdot \boldsymbol{k}\left(\hat{\alpha}', \mathbf{0}, \mathbf{0}\right) + z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\text{msk}, \boldsymbol{r}' + z_1 \cdot \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right)} \cdot g_3^{\boldsymbol{k}\left(y_3 \cdot \hat{\alpha}', z_3 \cdot \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot \boldsymbol{R}_3' \ .
\end{aligned}$$

We claim, that this key is either a properly distributed semi-functional key of Type 2 (if $Z = Z_1$) or a properly distributed semi-functional key of Type 3 (if $Z = Z_0$). Namely, $\mathcal{B}$ implicitly sets the random values of the $\mathbb{G}_{p_1}$ components as $\boldsymbol{r} = z_1 \cdot \hat{\boldsymbol{r}}' + \boldsymbol{r}' \pmod{p_1}$, which are properly distributed due to the choice of $\boldsymbol{r}'$. The random values of $\mathbb{G}_{p_2}$ components are set as $\hat{\alpha} = y_2 \cdot \hat{\alpha}' \pmod{p_2}$ and $\hat{\boldsymbol{r}} = z_2 \cdot \hat{\boldsymbol{r}}' \pmod{p_2}$. If $Z = Z_0$, it holds $z_2 = 0 \pmod{p_2}$ and thus

$$\boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}}\right) = \boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', \mathbf{0}, \hat{\boldsymbol{h}}\right) = \boldsymbol{k}\left(y_2 \cdot \hat{\alpha}', \mathbf{0}, \mathbf{0}\right) \pmod{p_2} \ .$$

Hence, if $Z = Z_0$ the $\mathbb{G}_{p_2}$ components are properly distributed as defined for Type 2 keys due to the choice of $\hat{\alpha}'$ (since $y_2 \neq 0 \pmod{p_2}$). If $Z = Z_1$, then $\hat{\alpha}$ and $\hat{\boldsymbol{r}}$ are properly distributed, as defined for Type 3 keys, due to the choice of $\hat{\alpha}' \pmod{p_2}$ and $\hat{\boldsymbol{r}}' \pmod{p_2}$ (since $y_2, z_2 \neq 0 \pmod{p_2}$), respectively. Finally, the $\mathbb{G}_{p_3}$ components are set as $\boldsymbol{R}_3 = g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}\right)} \cdot \boldsymbol{R}_3'$ and are properly distributed by the choice of $\boldsymbol{R}_3'$.

We deduce that for every $\text{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \text{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\text{Adv}_{\mathcal{B}'}^{\text{SD2}}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\
&= \left|\frac{1}{2} + \text{Adv}_{\Pi,\mathcal{A}}^{\text{G}_{k,2}}(\lambda, \text{des}) - \left(\frac{1}{2} + \text{Adv}_{\Pi,\mathcal{A}}^{\text{G}_{k,3}}(\lambda, \text{des})\right)\right| \\
&= \left|\text{Adv}_{\Pi,\mathcal{A}}^{\text{G}_{k,2}}(\lambda, \text{des}) - \text{Adv}_{\Pi,\mathcal{A}}^{\text{G}_{k,3}}(\lambda, \text{des})\right| \ .
\end{aligned}$$

The second equation holds since $\mathcal{B}$ perfectly simulates $\text{G}_{k,2}$ and $\text{G}_{k,3}$ if $Z = Z_0$ and if $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. □

From $\mathbf{G}_{q_1,3}$ to $\mathbf{G}_{q_1+1}$. Experiment $\mathrm{G}_{q_1,3}$ is a special case of $\mathrm{G}_{k,3}$ for $k = q_1$. In $\mathrm{G}_{q_1,3}$ all corrupted keys in Phase I are semi-functional of Type 3 and in Phase II all corrupted keys are normal. We simplify the description of the experiment as follows.

> $\mathrm{G}_{q_1,3}$:
> **Open** (i) in Phase I:
>
> – If sk_i is not generated yet, choose $\hat{\alpha}_j \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 3, \hat{\alpha}_j, g_2, \hat{\boldsymbol{h}}\right)$.
>
> **Open** (i) in Phase II:
>
> – If sk_i is not generated yet, return $\mathrm{sk}_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$.

G_{q_1+1} is as $\mathrm{G}_{q_1,3}$, but the corrupted keys in Phase II are semi-functional of Type 1:

> Changes in G_{q_1+1} compared to $\mathrm{G}_{q_1,3}$:
> **Open** (i) in Phase II:
>
> – If sk_i is not generated yet, return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 1, \_, g_2, \hat{\boldsymbol{h}}\right)$.

Lemma D.14. *(cf. Lemma 32 in [3]) For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1,3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+1}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}_{\mathcal{B}}^{\mathrm{SD2}}(\lambda) \quad .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$, that can distinguish between $\mathrm{G}_{q_1,3}$ and G_{q_1+1}, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 with the same advantage. Let $\mathrm{des} \in \Omega$ be arbitrary. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD2 as explained in Remark D.1. The algorithm $\mathcal{B}$ is again similar to the Algorithm 7, but there is no distinction of cases in the opening oracle. Hence, we presented the complete algorithm. $\mathcal{B}$ is a ppt algorithm with respect to 1^λ by construction. In particular, random elements from $\mathbb{G}_{p_3}$ can be chosen using g_3, the elements from $g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r}',\boldsymbol{h}\right)}$ can be computed as shown in Lemma F.1, and $\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)$ can be computed explicitly.

Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$. By construction of $\mathcal{B}$ and by Statement 1 of Lemma D.3 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and the master secret msk are distributed as defined in the experiments. Note, that $\mathbb{GD}_N$, g_1 and g_3 are distributed as required by Lemma D.3 due to the definition of experiment SD2. Furthermore, by Statement 2 of Lemma D.3 it holds $g_1^{\boldsymbol{h}'} = g_1^{\boldsymbol{h}}$, $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{\boldsymbol{h}} = \boldsymbol{h}' \pmod{p_2}$, $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma D.3.

Let g_2 be an arbitrary but fixed generator of $\mathbb{G}_{p_2}$. Then, by the definition of probability experiment SD2 it holds $Z = g_1^{z_1} g_2^{z_2} g_3^{z_3}$, where $z_1 \in \mathbb{Z}_{p_1}^*$, $z_3 \in \mathbb{Z}_{p_3}^*$ are uniformly distributed, and z_2 is either uniformly distributed in $\mathbb{Z}_{p_2}^*$ (if $Z = Z_1$) or $z_2 = 0 \pmod{p_2}$ (if $Z = Z_0$). Furthermore, $X_1 = g_1^{x_1}$, $X_2 = g_2^{x_2}$, $Y_2 = g_2^{y_2}$ and $Y_3 = g_3^{y_3}$, where $x_1 \in \mathbb{Z}_{p_1}^*$, $x_2, y_2 \in \mathbb{Z}_{p_2}^*$ and $y_3 \in \mathbb{Z}_{p_3}^*$ are uniformly distributed and mutually independent.

The challenge and all semi-functional keys of Type 3 in Phase I are generated as required in the experiments by Lemma D.4, and by Lemma D.5 respectively.

Consider the corrupted keys in Phase II. By construction of the algorithm $\mathcal{B}$, for every i and every generated secret key $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$ it holds:

$$\begin{aligned}
\boldsymbol{K} &= g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r}',\boldsymbol{h}\right)} \cdot Z^{\boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r}',\boldsymbol{h}\right)} \cdot \left(g_1^{z_1} g_2^{z_2} g_3^{z_3}\right)^{\boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r}',\boldsymbol{h}\right) + z_1 \cdot \boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h}'\right)} g_2^{z_2 \cdot \boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h}'\right)} \cdot g_3^{z_3 \cdot \boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \\
&= g_1^{\boldsymbol{k}\left(\mathrm{msk},\boldsymbol{r}' + z_1 \cdot \hat{\boldsymbol{r}}',\boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{k}\left(0, z_2 \cdot \hat{\boldsymbol{r}}',\hat{\boldsymbol{h}}\right)} \cdot g_3^{z_3 \cdot \boldsymbol{k}\left(0,\hat{\boldsymbol{r}}',\boldsymbol{h}'\right)} \cdot \boldsymbol{R}_3' \quad .
\end{aligned}$$

Algorithm 10: $\mathcal{B}$ against Assumption SD2

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup
- Compute the public parameters and the master secret key $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v) \leftarrow \mathrm{SimPP}\left(1^\lambda, \mathbb{GD}_N, g_1, g_3, \mathrm{des}\right)$. Set $j := 0$.

Phase I
- **CoveredKeyGen** (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
  - Store (i, kInd_i).
- **Open** (i)**:**
  - Set $j := j + 1$.
  - Pick $\alpha'_j \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk}_i \leftarrow \mathrm{SimSFKeyT3}\left(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i, Y_2Y_3, \alpha'_j\right)$.
- **Decapsulate** (CT, i)**:**
  - As defined in the experiment using $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.

Challenge (given cInd^* from $\mathcal{A}$)
- Generate $(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SimSFChlg}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}^*, \boldsymbol{h}', u, v, X_1X_2)$.
- Pick $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, flip a coin $b \leftarrow \{0,1\}$, set $\mathrm{K}^* := \mathrm{K}_b$, and return the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.

Phase II
- **CoveredKeyGen** (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
  - Store (i, kInd_i).
- **Open** (i)**:**
  - Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\mathrm{kInd}_i)$. Let $m_1 = |\boldsymbol{k}|$. Pick $\boldsymbol{r}', \hat{\boldsymbol{r}}' \leftarrow \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}'_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute
  $$\boldsymbol{K} := g_1^{\boldsymbol{k}\left(\mathrm{msk}, \boldsymbol{r}', \boldsymbol{h}\right)} \cdot Z^{\boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3 \ .$$
  - Return $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$.
- **Decapsulate** (CT, i)**:**
  - As defined in the experiment using $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.

Guess
- Simulate this phase as defined in the experiment.

We claim, that for every i, the corresponding secret key sk is either a properly distributed normal secret key (if $Z = Z_0$) or a properly distributed semi-functional secret key of Type 1 (if $Z = Z_1$). Namely, $\mathcal{B}$ implicitly sets the random values of the normal components (in $\mathbb{G}_{p_1}$) as $\boldsymbol{r} = z_1 \cdot \hat{\boldsymbol{r}}' + \boldsymbol{r}' \pmod{p_1}$, which are properly distributed due to the choice of $\boldsymbol{r}'$. The random values of the semi-functional components (in $\mathbb{G}_{p_2}$) are set as $\hat{\boldsymbol{r}} = z_2 \cdot \hat{\boldsymbol{r}}'$, which are properly distributed (for Type 1 keys) due to the choice of $\hat{\boldsymbol{r}}'$ if $Z = Z_1$, and which disappear if $Z = Z_0$ and hence, $z_2 = 0 \pmod{p_2}$. Finally, the $\mathbb{G}_{p_3}$ components are set as $\boldsymbol{R}_3 = g_3^{z_3 \cdot \boldsymbol{k}\left(0, \hat{\boldsymbol{r}}', \boldsymbol{h}'\right)} \cdot \boldsymbol{R}'_3$ and are properly distributed by the choice of $\boldsymbol{R}'_3$.

Hence, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_\mathcal{A}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD2}}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\
&= \left|\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1,3}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+1}}(\lambda, \mathrm{des})\right)\right| \\
&= \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1,3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+1}}(\lambda, \mathrm{des})\right| .
\end{aligned}$$

The second equation holds since $\mathcal{B}$ perfectly simulates $\mathrm{G}_{q_1,3}$ and G_{q_1+1} if $Z = Z_0$ and $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. □

From $\mathbf{G}_{q_1+1}$ to $\mathbf{G}_{q_1+2}$. G_{q_1+2} is as G_{q_1+1}, but the key in Phase II are semi-functional of Type 2.

Changes in G_{q_1+2} compared to G_{q_1+1}:

– Choose $\hat{\alpha} \leftarrow \mathbb{Z}_N$ at the beginning of Phase II.

Open (i) in Phase II

– Return $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}\left(\mathrm{msk}, \mathrm{kInd}_i, 2, \hat{\alpha}, g_2, \hat{\boldsymbol{h}}\right)$.

Lemma D.15. *(cf. Lemma 33 in [3]) For every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}$ such that for every security parameter λ and every $\mathrm{des} \in \Omega$ it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+1}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+2}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}_{\mathrm{P},\mathcal{B}}^{\mathrm{SMH}}(\lambda, \mathrm{des}) \quad .$$

The running time of $\mathcal{B}$ is essentially the same as the running time of $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$, that can distinguish between G_{q_1+1} and G_{q_1+2}, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks the selective master-key hiding security property of the underlining pair encoding scheme with the same advantage. $\mathcal{B}$ on input $(\mathbb{GD}_N, g_1, g_2, g_3, \mathrm{des})$ as defined in $\mathrm{Exp}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}^{\mathrm{SMH}}(\lambda, \mathrm{des})$ is as follows.

Note, that $\mathcal{B}$ is similar to Algorithm 8, but the oracle is used in Phase II to generate all corrupted keys. $\mathcal{B}$ is a ppt algorithm with respect to λ by construction. It uses different supplementary ppt algorithms and performs besides only simple computation. Next we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$.

Let security parameter λ and $\mathrm{des} \in \Omega$ be arbitrary, but fixed. By the definition of the Experiment $\mathrm{Exp}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}^{\mathrm{SMH}}(\lambda, \mathrm{des})$, $\mathbb{GD}_N$ is the restricted group description of $\mathbb{GD}$ generated by $\mathcal{G}\left(1^\lambda\right)$. Furthermore, the generators $g_i \in \mathbb{G}_{p_i}$ are chosen uniformly at random. Hence, by construction of $\mathcal{B}$ and by Statement 1 of Lemma D.3 the public parameters $\mathrm{pp}_\kappa = \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)$ and the master secret msk are distributed as defined in the experiments. Furthermore, by Statement 2 of Lemma D.3 it holds $U_1 = g_1^u$ and $V_1 = g_1^v$. $\mathcal{B}$ implicitly sets the semi-functional elements as $\hat{u}_2 = u \pmod{p_2}$, and $\hat{v}_2 = v \pmod{p_2}$. These elements are properly distributed by Statement 3 of Lemma D.3. Furthermore, $\mathcal{B}$ implicitly sets the input generator g_2 as the generator of $\mathbb{G}_{p_2}$. This generator is properly distributed as mentioned above. Vector $\hat{\boldsymbol{h}} \pmod{p_2}$ of the semi-functional public parameters will be defined below.

It is important to notice that all oracle queries made by $\mathcal{B}$ are permissible if all corruption queries of $\mathcal{A}$ are permissible, since $\mathrm{R}_N(\mathrm{kInd}, \mathrm{cInd}^*) = 0$ implies $\mathrm{R}_{p_2}(f_1(\mathrm{kInd}), f_2(\mathrm{cInd}^*)) = 0$ by Corollary D.2. The normal keys and the semi-functional keys of Type 3 are generated using msk and g_2 as defined in the experiments.

By the definition of the Experiment $\mathrm{Exp}_{\mathrm{P},\mathcal{G},\nu,\mathcal{A}}^{\mathrm{SMH}}(\lambda, \mathrm{des})$, the challenger choose $\hat{\alpha} \leftarrow \mathbb{Z}_N$ and $\hat{\boldsymbol{h}} \leftarrow \mathbb{Z}_N^n$, where $n = \mathrm{Param}(\kappa)$. Then, $\mathcal{B}$ receives

$$\widehat{\boldsymbol{K}} = \mathcal{O}^2_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{kInd}_i) = \begin{cases} g_2^{\boldsymbol{k}\left(0,\tilde{\boldsymbol{r}},\hat{\boldsymbol{h}}\right)} & \text{if } \nu = 0 \\ g_2^{\boldsymbol{k}\left(\hat{\alpha},\tilde{\boldsymbol{r}},\hat{\boldsymbol{h}}\right)} & \text{if } \nu = 1 \end{cases} ,$$

where $\hat{r} \in \mathbb{Z}_{p_2}^{m_2}$ is chosen uniformly at random for every key. Hence, all corrupted keys in Phase II are either properly distributed semi-functional keys of Type 1 (if $\nu = 0$) or properly distributed semi-functional keys of Type 2 (if $\nu = 1$) by construction. If $\nu = 1$, the element $\hat{\alpha}$ is the same for all these keys, as defined in the experiment G_{q_1+2}.

Furthermore, $\mathcal{B}$ receives

$$\widehat{\boldsymbol{C}} = \mathcal{O}^1_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}(\mathrm{cInd}^*) = g_2^{\boldsymbol{c}\left(\tilde{s},\tilde{\boldsymbol{s}},\hat{\boldsymbol{h}}\right)} \quad ,$$

where $\tilde{s} \in \mathbb{Z}_{p_2}$ and $\tilde{\boldsymbol{s}} \in \mathbb{Z}_{p_2}^{w_2}$ are chosen uniformly at random, $\hat{\boldsymbol{h}}$ is as above. Hence, $\widehat{\boldsymbol{C}}$ are properly distributed semi-functional components with $\hat{s} = \tilde{s} \pmod{p_2}$ and $\hat{\boldsymbol{s}} = \tilde{\boldsymbol{s}} \pmod{p_2}$.

Finally, we show that the last group element in the challenge encapsulation is correctly generated:

$$\begin{aligned} C''^* &= (C_1)^{u\cdot t^*+v} \cdot \left(\widehat{C}_1\right)^{u\cdot t^*+v} \\ &= \left(g_1^s\right)^{u\cdot t^*+v} \cdot \left(g_2^{\hat{s}}\right)^{u\cdot t^*+v} \\ &= \left(U_1^{t^*} \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}_2\cdot t^*+\hat{v}_2}\right)^{\hat{s}} \quad , \end{aligned}$$

Algorithm 11: $\mathcal{B}$ against selective master-key hiding security property

Input : $(\mathbb{GD}_N, g_1, g_2, g_3, \mathrm{des})$.

1 **Setup**

2 Compute the public parameters and the master secret key $(\mathrm{msk}, \mathrm{pp}_\kappa, \_, u, v) := \mathrm{SimPP}\left(1^\lambda, \mathbb{GD}_N, g_1, g_3, \mathrm{des}\right)$. Set $j := 0$.

3 **Phase I**

4 **CoveredKeyGen** (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:

5 Store (i, kInd_i).

6 **Open** (i)**:**

7 Set $j := j + 1$.

8 Pick $\alpha_j \leftarrow \mathbb{Z}_N$ and output $\mathrm{sk} \leftarrow \mathrm{SFKeyGen}\,(\mathrm{msk}, \mathrm{kInd}_i, 3, \alpha_j, g_2, \_)$.

9 **Decapsulate** (CT, i)**:**

10 As defined in the experiment using $\mathrm{sk}_i' \leftarrow \mathrm{KeyGen}\,(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.

11 **Challenge** (given cInd^* from $\mathcal{A}$)

12 Compute $(\mathrm{K}_0, (\mathrm{cInd}^*, \boldsymbol{C}, \_)) \leftarrow \mathrm{Encaps}\,(\mathrm{cInd}^*)$.

13 Query the own oracle

$$\widehat{\boldsymbol{C}} := \mathcal{O}^1_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\,(\mathrm{cInd}^*) \quad ,$$

Set $\boldsymbol{C}^* := \boldsymbol{C} \cdot \widehat{\boldsymbol{C}}$.

14 Compute $t^* := \mathrm{H}\,(\mathrm{HInput}\,(\mathrm{cInd}^*, \boldsymbol{C}^*, \_))$ and

$$C''^* := (C_1^*)^{u \cdot t^* + v} \quad .$$

15 Choose $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, pick $b \leftarrow \{0,1\}$, set $\mathrm{K}^* := \mathrm{K}_b$ and return $(\mathrm{K}^*, (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*))$.

16 **Phase II**

17 **CoveredKeyGen** (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:

18 Store (i, kInd_i).

19 **Open** (i)**:**

20 Query the oracle

$$\widehat{\boldsymbol{K}} := \mathcal{O}^2_{\mathrm{SMH},\nu,\hat{\alpha},\hat{\boldsymbol{h}}}\,(\mathrm{kInd}_i) \quad .$$

21 Compute a normal key $(\mathrm{kInd}_i, \boldsymbol{K}) \leftarrow \mathrm{KeyGen}\,(\mathrm{msk}, \mathrm{kInd}_i)$ and return

$$\mathrm{sk} := \left(\mathrm{kInd}_i, \boldsymbol{K} \cdot \widehat{\boldsymbol{K}}\right).$$

22 **Decapsulate** (CT, i)**:**

23 As defined in the experiment using $\mathrm{sk}_i' \leftarrow \mathrm{KeyGen}\,(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.

24 **Guess**

25 Simulate this phase as defined in the experiment

where s is the random element fixed by C_1, which is chosen as the first element of $\boldsymbol{C}$ in Line 12 and $\hat{s}$ is fixed by $\widehat{C}_1$ as defined above. In the second equation we used the normality of P. Hence, C''^* is exactly as defined in SFEncaps.

We deduce that for every $\mathcal{A}$ there exist a ppt algorithm $\mathcal{B}$ such that for every security parameter λ and every $\mathrm{des} \in \Omega$ it holds

$$\begin{aligned}
\mathrm{Adv}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{B}}\,(\lambda, \mathrm{des}) &= \left|\mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},0,\mathcal{B}}\,(\lambda, \mathrm{des}) - \mathrm{Exp}^{\mathrm{SMH}}_{\mathrm{P},\mathcal{G},1,\mathcal{B}}\,(\lambda, \mathrm{des})\right| \\
&= \left|\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1+1}}_{\Pi,\mathcal{A}}\,(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}^{\mathrm{G}_{q_1+2}}_{\Pi,\mathcal{A}}\,(\lambda, \mathrm{des})\right)\right| \\
&= \left|\mathrm{Adv}^{\mathrm{G}_{q_1+1}}_{\Pi,\mathcal{A}}\,(\lambda, \mathrm{des}) - \mathrm{Adv}^{\mathrm{G}_{q_1+2}}_{\Pi,\mathcal{A}}\,(\lambda, \mathrm{des})\right|.
\end{aligned}$$

The second equation holds since $\mathcal{B}$ correctly simulates G_{q_1+1} and G_{q_1+2} if $\nu = 0$ and if $\nu = 1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding game. This proves the lemma. ☐

From $\mathbf{G}_{q_1+2}$ to $\mathbf{G}_{q_1+3}$. G_{q_1+3} is as G_{q_1+2}, but the key in Phase II are semi-functional of Type 3.

Changes in G_{q_1+3} compared to G_{q_1+2}:
Open (i) in Phase II:

– $\mathrm{sk}_i \leftarrow \mathrm{SFKeyGen}(\mathrm{msk}, \mathrm{kInd}_i, 3, \hat{\alpha}, g_2, \_)$. (Where $\hat{\alpha}$ as defined in G_{q_1+1})

Lemma D.16. *(cf. Lemma 34 in [3]) For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+2}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+3}}(\lambda, \mathrm{des})\right| = \mathrm{Adv}_{\mathcal{A}}^{\mathrm{SD2}}(\lambda) \quad .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$, that can distinguish between G_{q_1+2} and G_{q_1+3}, we construct a ppt algorithm $\mathcal{B}$ which uses $\mathcal{A}$ and breaks Assumption SD2 with the same advantage. Let $\mathrm{des} \in \Omega$ be arbitrary. $\mathcal{B}$ is given des in addition to its input (D, Z) from experiment SD2 as explained in Remark D.1. Note that $\mathcal{B}$ is almost the same as Algorithm 10. Hence, next we present only the simulation of corrupted keys in Phase II:

Algorithm 12: $\mathcal{B}$ against Assumption SD2 as modification of Algorithm 10

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.
Setup
  ...
...
Phase II
  Pick $\hat{\alpha}' \leftarrow \mathbb{Z}_N$.
  ...
  Open (i)**:**
    Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\mathrm{kInd}_i)$. Let $m_1 := |\boldsymbol{k}|$.
    Pick $\boldsymbol{r}', \hat{\boldsymbol{r}}', \leftarrow \mathbb{Z}_N^{m_2}$ and $\boldsymbol{R}_3' \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute

$$\boldsymbol{K} := g_1^{\boldsymbol{k}(\alpha, \boldsymbol{r}', \boldsymbol{h})} \cdot (Y_2Y_3)^{\boldsymbol{k}(\hat{\alpha}', 0, 0)} \cdot Z^{\boldsymbol{k}(0, \hat{\boldsymbol{r}}', \boldsymbol{h})} \cdot \boldsymbol{R}_3'$$

    Return $(\mathrm{kInd}_i, \boldsymbol{K})$.
  ...
...

According to the analysis of Algorithm 10 we have to consider only the corrupted keys in Phase 2. Note, that these keys are generated as the semi-functional keys of Type 3 in Phase I in Algorithm 9 except for the choice of $\hat{\alpha}'$, which is the same for all keys in this phase.

Let $Y_2 = g_2^{y_2}$ and $Z = g_1^{z_1} g_2^{z_2}$, where g_2 is an arbitrary but fixed generator of $\mathbb{G}_{p_2}$, $y_2 \in \mathbb{Z}_{p_2}^*$, $z_1 \in \mathbb{Z}_{p_1}^*$, and either $z_2 = 0 \pmod{p_2}$ if $Z = Z_0$ or $z_2 \in \mathbb{Z}_{p_2}^*$ if $Z = Z_1$, as defined in SD2. As already shown in the analysis of Algorithm 9, the semi-functional $(\mathbb{G}_{p_2})$ components of the keys are set to:

$$g_2^{\boldsymbol{k}(y_2 \cdot \hat{\alpha}', z_2 \cdot \hat{\boldsymbol{r}}', \hat{\boldsymbol{h}})} \quad .$$

The elements in $\hat{\boldsymbol{r}}'$ are chosen uniformly at random for every key, whereas $\hat{\alpha}'$ is chosen once in the Setup Phase. Hence, the corresponding key is either a properly distributed semi-functional key of Type 2 (if $Z = Z_1$) or a properly distributed semi-functional key of Type 3 (if $Z = Z_0$) as already explained in the analysis of Algorithm 9.

Hence, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD2}}(\lambda) &= \left|\Pr[\mathcal{B}'(D, Z_0) = 1] - \Pr[\mathcal{B}'(D, Z_1) = 1]\right| \\
&= \left|\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+3}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+2}}(\lambda, \mathrm{des})\right)\right| \\
&= \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+2}}(\lambda, \mathrm{des})\right| \quad .
\end{aligned}$$

The second equation holds since $\mathcal{B}$ perfectly simulates G_{q_1+3} and G_{q_1+2} if $Z = Z_0$ and $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. ☐

From $\mathbf{G}_{q_1+3}$ to $\mathbf{G}'_{q_1+3}$. In experiment G_{q_1+3} all keys generated in the opening oracle are semi-functional of Type 3 and all keys used in the decapsulation queries are normal. Experiment G'_{q_1+3} is defined as G_{q_1+3} except for decapsulation queries, which are answered without user secret keys. Namely, an additional generator X_2 of $\mathbb{G}_{p_2}$ is chosen uniformly at random in the setup phase and the decapsulation queries on $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$, which passes the consistency checks, are answered with $\mathrm{e}\left(g_1^{\mathrm{msk}} \cdot X_2, C_1\right)$, where C_1 is the first element of $\boldsymbol{C}$.

Changes in G'_{q_1+3} compared to G_{q_1+3}:

– Pick $X_2 \leftarrow \mathbb{G}_{p_2}$ in the Setup phase.

Exchange $\langle 4 \rangle$ and $\langle 9 \rangle$ for:

– **Decapsulate** (CT, i):
  - Perform the (implicit) syntactic checks. Return $\perp$ if these are not satisfied. Otherwise $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'')$ for $\mathrm{cInd} \in \mathbb{Y}_\kappa$. Return $\perp$ if $\mathrm{R}(\mathrm{kInd}_i, \mathrm{cInd}) = 0$.
  - Return $\perp$ if the consistency checks in Decaps are not satisfied for CT.
  - Return $\mathrm{e}\left(g_1^{\mathrm{msk}} \cdot X_2, C_1\right)$, where $C_1 \in \boldsymbol{C}$.

Lemma D.17. *For every* $\mathrm{des} \in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}'_{q_1+3}}(\lambda, \mathrm{des})\right| \leq \mathrm{Adv}_{\mathcal{B}}^{\mathrm{SD2}}(\lambda) + \frac{q_{\mathrm{dec1}}}{p_1} + \frac{2}{p_2} \quad ,$$

where q_{dec1} *is the number of decapsulation queries in Phase I. The running time of* $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Experiments G_{q_1+3} and G'_{q_1+3} differ only in the realization of the decapsulation oracles. First, let us consider experiment G_{q_1+3}. Suppose that $\mathcal{A}$ queries the decapsulation oracle on (CT, i) with $\mathrm{CT} = (\mathrm{cInd}, \boldsymbol{C}, C'') \in \mathbb{C}_{\mathrm{cInd}}$ and $i \in \mathbb{N}$, and suppose that CT passes the consistency checks. Then, due to the checks in (4), the elements in $\boldsymbol{C}$ do not contain $\mathbb{G}_{p_3}$ components. By the definition of experiment G_{q_1+3}, the decapsulation queries are answered using normal keys. Let $\mathrm{sk}'_i = (\mathrm{kInd}_i, \boldsymbol{K})$, $\boldsymbol{K} = g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot \boldsymbol{R}_3$ be the corresponding normal secret key generated to answer the decapsulation query. Recall that the group elements of normal keys do not contain $\mathbb{G}_{p_2}$ components. Hence, during the decapsulation of CT using sk'_i the $\mathbb{G}_{p_2}$ components of $\boldsymbol{C}$ and the $\mathbb{G}_{p_3}$ components of $\boldsymbol{K}$ disappear. Namely, it holds

$$\mathrm{K} = \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot \boldsymbol{R}_3\right)^{\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})}\right)^{\boldsymbol{E}}, \boldsymbol{C}\right) = \mathrm{e}\left(g_1, C_1\right)^{\mathrm{msk}} \quad ,$$

where the last equation holds since CT pass the check in (5) and due to the soundness of algorithm Vrfy.

In turn, by the definition of experiment G'_{q_1+3}, the decapsulation oracle on CT as above returns

$$\mathrm{K} = \mathrm{e}\left(g_1^{\mathrm{msk}} \cdot X_2, C_1\right) = \mathrm{e}\left(g_1, C_1\right)^{\mathrm{msk}} \cdot \mathrm{e}\left(X_2, C_1\right) \quad .$$

We deduce that the view of $\mathcal{A}$ and hence, its success probability in the experiments G'_{q_1+3} and G_{q_1+3} can differ if and only if $\mathcal{A}$ queries the decapsulation oracle on $(\mathrm{CT}, i) \in \mathbb{C}_{\mathrm{cInd}} \times \mathbb{N}$ such that CT pass the consistency checks and $C_1 \in \mathrm{CT}$ contains a $\mathbb{G}_{p_2}$ component. At the same time the challenger should not output 0 in the Guess Phase because of an abort event (recall, that in this case the output of both experiments is 0). Hence, we call by CipherAbort the event that a decapsulation query on (CT, i) with property from above (C_1 contains $\mathbb{G}_{p_2}$ component) exist and the events HashAbort and FactorAbort do not occur. By Lemma D.6 it holds

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}'_{q_1+3}}(\lambda, \mathrm{des})\right| \leq \Pr\left[\mathrm{CipherAbort}\right] \quad .$$

Algorithm 13: $\mathcal{B}$ against Assumption SD2

Input : (D, Z, des).
Require: $D = (\mathbb{GD}_N, g_1, X_1X_2, Y_2Y_3, g_3)$, $Z \in \mathbb{G}$, $\mathrm{des} \in \Omega$.

Setup
  Compute $(\mathrm{msk}, \mathrm{pp}_\kappa, \boldsymbol{h}', u, v) \leftarrow \mathrm{SimPP}(\mathbb{GD}_N, g_1, g_3, \mathrm{des})$. Pick $\hat{\alpha}' \leftarrow \mathbb{Z}_N$.
Phase I
  CoveredKeyGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
    Store (i, kInd_i).
  Open (i):
    Pick $\hat{\alpha} \leftarrow \mathbb{Z}_N$ and return $\mathrm{sk} \leftarrow \mathrm{SimSFKeyT3}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i, Y_2Y_3, \hat{\alpha})$.
  Decapsulate (CT, i):
    As defined in the experiment using normal secret key $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.
Challenge (given cInd^* from $\mathcal{A}$)
  Generate $(\mathrm{K}_0, \mathrm{CT}^*) \leftarrow \mathrm{SimSFChlg}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{cInd}^*, \boldsymbol{h}', u, v, X_1X_2)$.
  Pick $\mathrm{K}_1 \leftarrow \mathbb{G}_\mathrm{T}$, flip a coin $b \leftarrow \{0,1\}$, set $\mathrm{K}^* := \mathrm{K}_b$, and return the challenge $(\mathrm{K}^*, \mathrm{CT}^*)$.
Phase II
  CoveredKeyGen (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
    As before.
  Open (i):
    Return $\mathrm{sk} \leftarrow \mathrm{SimSFKeyT3}(\mathrm{pp}_\kappa, \mathrm{msk}, \mathrm{kInd}_i, Y_2Y_3, \hat{\alpha}')$.
  Decapsulate (CT, i):
    As defined in the experiment using normal secret key $\mathrm{sk}'_i \leftarrow \mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd}_i)$ generated once.
Guess
  Ignore the output of $\mathcal{A}$. Choose $\nu' \leftarrow \{0,1\}$.
  Perform the original checks of bad events and output ν' if one of these events occurs.
  foreach *decapsulation query on* (CT, i) *with* $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ *and* $\mathrm{R}(\mathrm{kInd}_i, \mathrm{cInd}) = 1$*, where* CT *pass the consistency checks (let t be the corresponding hash value)* **do**
    if $t = t^* \pmod N$ **then**
      if *the query is made in Phase I* **then** output ν';
      else
        Look for an index $l \in [w_1 + 1]$ (index $w_1 + 1$ corresponds to the element C'') such that $C_l \neq C^*_l$. Then, compute $G_1 := {}^{C_l}/_{C^*_l}$.
        if $\mathrm{e}(G_1, Z) \neq 1_{\mathbb{G}_\mathrm{T}}$ **then** output 1;
        else output 0;
      end
    else
      Compute $G_2 := C'' \cdot C_1^{-(u \cdot t + v)}$.
      if $G_2 \neq 1_\mathbb{G}$ **then**
        if $\mathrm{e}(G_2, Z) \neq 1_{\mathbb{G}_\mathrm{T}}$ **then** output 1;
        else output 0;
      end
    end
  end
  Output ν'.

In order to analyze CipherAbort event, we construct an algorithm $\mathcal{B}$ against Experiment SD2, which uses $\mathcal{A}$ as a subroutine. $\mathcal{B}$ simulates game G_{q_1+3} for $\mathcal{A}$ and if the event CipherAbort occurs, $\mathcal{B}$ breaks Experiment SD2 which violates Assumption SD2. The main observation is that if the event CipherAbort occurs and $t \neq t^* \pmod N$, then with overwhelming probability (over the random choices of $\hat{u}$ and $\hat{v} \pmod{p_2}$) we can use the elements C_1 and C'' in order to compute a generator of $\mathbb{G}_{p_2}$. Furthermore, if the event CipherAbort occurs and $t = t^* \pmod N$, we will found an index l such that C_l and C^*_l differ only in the $\mathbb{G}_{p_2}$ components and hence, we again compute a generator of $\mathbb{G}_{p_2}$. Using a generator of $\mathbb{G}_{p_2}$ we can break the own challenge.

Let des $\in \Omega$ be arbitrary, but fixed. $\mathcal{B}$ is given des $\in \Omega$ in addition to its input (D, Z) from experiment SD2 as explained in Remark D.1.

$\mathcal{B}$ is a ppt algorithm with respect to λ by construction. It uses different supplementary ppt algorithms and performs besides only simple computation. Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$. $\mathcal{B}$ simulates $\mathcal{A}$ until her output (which is ignored) using supplementary algorithms. We are not interested in the output of the experiment, but in the success probability of $\mathcal{B}$ related to the probability that the event CipherAbort occurs. Hence, consider the computation of $\mathcal{B}$ in the Guess Phase.

At first, let us consider the computation of $\mathcal{B}$ in the Guess Phase independently of the event CipherAbort. Recall from page 11 that for $\mathrm{CT} \in \mathbb{C}_{\mathrm{cInd}}$ the input of the hash function is $\mathrm{HInput}(\mathrm{CT}) = (\mathrm{cInd}, \mathrm{e}(g_1, C_1), \ldots, \mathrm{e}(g_1, C_{w_1}))$. Let (CT, i) be an encapsulation, considered in the foreach loop. Due to the checks in Line 22 we can assume that event HashAbort does not occur (cf. Corollary D.1). Hence, in Line 25, $t = t^* \pmod N$ implies $\mathrm{HInput}(\mathrm{CT}) = \mathrm{HInput}(\mathrm{CT}^*)$. If an encapsulation with these properties occurs in Phase I, $\mathcal{B}$ aborts and outputs a guess (event Abort). But in Phase I, $\mathcal{A}$ gets no information about C_1^*, and therefore it gets no information about its $\mathbb{G}_{p_1}$ component $g_1^{s^*}$. In turn, $s^* \pmod{p_1}$ is uniformly distributed over $\mathbb{Z}_{p_1}$ and fixes the second element of $\mathrm{HInput}(\mathrm{CT}^*)$. Hence, the overall probability that $\mathcal{B}$ aborts and outputs a guess in Line 25 is at most q_{dec1}/p_1, where q_{dec1} is the number of decapsulation queries in Phase I:

$$\Pr[\mathrm{Abort}] \leq q_{\mathrm{dec1}}/p_1 \quad .$$

Next we consider $\mathcal{B}$'s computation in Line 27. In this case a query with $t = t^* \pmod N$ is made in the second phase and consequently it holds $\mathrm{HInput}(\mathrm{CT}) = \mathrm{HInput}(\mathrm{CT}^*)$. But in Phase II the adversary is not allowed to query the decapsulation of CT^*, that is $\mathrm{CT} \neq \mathrm{CT}^*$. We deduce that $\mathrm{cInd} = \mathrm{cInd}^*$ and the $\mathbb{G}_{p_1}$ components of all corresponding elements in $\boldsymbol{C}$ and $\boldsymbol{C}^*$ are equal. Together with the consistency check from (3) this implies that the $\mathbb{G}_{p_1}$ components of C'' and C''^* are equal, too. Furthermore, by the consistency checks in (4), the group elements of CT do not contain $\mathbb{G}_{p_3}$ components. The group elements of CT^* do not contain $\mathbb{G}_{p_3}$ components by construction. Hence, there is an index $l \in [w_1 + 1]$ such that $C_l \neq C_l^*$, the $\mathbb{G}_{p_1}$ components of both elements are equal and the $\mathbb{G}_{p_3}$ components are not present. We deduce that if $t = t^* \pmod N$ and the query is made in Phase II, $\mathcal{B}$ computes a generator $G_1 \in \mathbb{G}_{p_2}$ and can solve the own challenge with success probability 1.

Next, we consider the computation of $\mathcal{B}$ in Line 32. In this case $t \neq t^* \pmod N$, which implies

$$\mathrm{HInput}(\mathrm{CT}) \neq \mathrm{HInput}(\mathrm{CT}^*) \quad .$$

Hence, applying Corollary D.1 it holds $t \neq t^* \pmod{p_2}$. Let g_2 be an arbitrary, but fixed generator of $\mathbb{G}_{p_2}$. By the consistency checks in (4), the group elements of CT do not contain $\mathbb{G}_{p_3}$ components. Hence, we can denote $C_1 = g_1^s \cdot g_2^{\kappa_1}$, where $s \in \mathbb{Z}_{p_1}$ and $\kappa_1 \in \mathbb{Z}_{p_2}$. Furthermore, by the consistency checks in (3) it holds $C'' = g_1^{s\cdot(u\cdot t+v)} \cdot g_2^{\kappa''}$, where $\kappa'' \in \mathbb{Z}_{p_2}$. We deduce that $\mathcal{B}$ computes

$$G_2 = \frac{C''}{C_1^{u\cdot t+v}} = g_2^{\kappa'' - \kappa_1\cdot(u\cdot t+v)} \in \mathbb{G}_{p_2} \quad .$$

Hence, if $G_2 \neq 1_{\mathbb{G}}$ (which is additionally checked in the algorithm), $\mathcal{B}$ again solves the own challenge with success probability 1.

In summary, if $\mathcal{B}$ does not output the guess bit ν' (chosen uniformly and independently of any other random variables), and rather outputs 0 or 1 directly, the output is correct. In particular, the probability that $\mathcal{B}$ outputs 1 if $Z = Z_1$ is at least $1/2$, whereas the probability that $\mathcal{B}$ outputs 1 if $Z = Z_0$ is at most $1/2$. Hence, for every des $\in \Omega$ and every ppt algorithm $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_{\mathcal{A}}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}\mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD2}}(\lambda) &= |\Pr[\mathcal{B}'(D, Z_0) = 1] - \Pr[\mathcal{B}'(D, Z_1) = 1]| \\ &= \Pr[\mathcal{B}'(D, Z_1) = 1] - \Pr[\mathcal{B}'(D, Z_0) = 1] \quad .\end{aligned}$$

For our analysis, we can even neglect the advantage of $\mathcal{B}'$ in the case that the event CipherAbort does not occur. It holds

$$\Pr\left[\mathcal{B}'(D, Z_1) = 1 \;\middle|\; \overline{\mathrm{CipherAbort}}\right] - \Pr\left[\mathcal{B}'(D, Z_0) = 1 \;\middle|\; \overline{\mathrm{CipherAbort}}\right] \geq 0 \ ,$$

since also for this conditional probability distribution it holds $\Pr\left[\mathcal{B}'(D, Z_1) = 1 \,\middle|\, \overline{\text{CipherAbort}}\right] \geq {}^1\!/_2$ and $\Pr\left[\mathcal{B}'(D, Z_0) = 1 \,\middle|\, \overline{\text{CipherAbort}}\right] \leq {}^1\!/_2$. Hence, we continue the analysis:

$$\begin{aligned}
\text{Adv}_{\mathcal{B}'}^{\text{SD2}}(\lambda) &= \Pr[\mathcal{B}'(D, Z_1) = 1] - \Pr[\mathcal{B}'(D, Z_0) = 1] \\
&= \Pr[\text{CipherAbort}] \cdot (\Pr[\mathcal{B}'(D, Z_1) = 1 \mid \text{CipherAbort}] \\
&\qquad\qquad - \Pr[\mathcal{B}'(D, Z_0) = 1 \mid \text{CipherAbort}]) \\
&\quad + \Pr\left[\overline{\text{CipherAbort}}\right] \cdot \left(\Pr\left[\mathcal{B}'(D, Z_1) = 1 \,\middle|\, \overline{\text{CipherAbort}}\right]\right. \\
&\qquad\qquad \left.- \Pr\left[\mathcal{B}'(D, Z_0) = 1 \,\middle|\, \overline{\text{CipherAbort}}\right]\right) \\
&\geq \Pr[\text{CipherAbort}] \cdot (\Pr[\mathcal{B}'(D, Z_1) = 1 \mid \text{CipherAbort}] \\
&\qquad\qquad - \Pr[\mathcal{B}'(D, Z_0) = 1 \mid \text{CipherAbort}]) \quad .
\end{aligned}$$

Next, we consider the advantage of $\mathcal{B}$ under the condition, that the event CipherAbort occurs and the event Abort defined above does not occur. We claim, that under these conditions $\mathcal{B}$ either outputs a correct bit using G_1 or $\mathcal{B}$ founds $G_2 \in \mathbb{G}_{p_2}$ except for a negligible probability and solve the own challenge. Namely, if a query with $t = t^* \pmod N$ is made in the second phase $\mathcal{B}$ will found G_1 and solve the own challenge as already explained above. The second part of our claim is a bit more complex.

By construction of $\mathcal{B}$, the adversary $\mathcal{A}$ gets information about $\hat{u}, \hat{v} \in \mathbb{Z}_{p_2}$ only from the value $\hat{u} \cdot t^* + \hat{v} \pmod{p_2}$, which is included in the exponent of C''^* in the semi-functional challenge encapsulation. Furthermore, $t \neq t^* \pmod N$ implies $t \neq t^* \pmod N$ as explained above. But if $t \neq t^* \pmod{p_2}$, the values $\hat{u} \cdot t^* + \hat{v} \pmod{p_2}$ and $\hat{u} \cdot t + \hat{v} \pmod{p_2}$ are independent. Hence, $\mathcal{A}$ get no information about $\hat{u} \cdot t + \hat{v} \pmod{p_2}$. But, if event CipherAbort occurs, there is an encapsulation CT such that $C_1 = g_1^s \cdot g_2^{\kappa_1}$ for some $s \in \mathbb{Z}_{p_1}$ and $\kappa_1 \in \mathbb{Z}_{p_2}^*$. For this encapsulation CT, by the analysis from above, the corresponding elements C'' and G_2 are equal to $g_1^{s \cdot (u \cdot t + v)} \cdot g_2^{\kappa''}$ and $g_2^{\kappa'' - \kappa_1 \cdot (u \cdot t + v)}$ respectively. Since $\kappa_1 \neq 0 \pmod{p_2}$ we deduce that the probability for $\kappa'' - \kappa_1 \cdot (u \cdot t + v) = 0 \pmod{p_2}$ is negligible (namely ${}^1\!/_{p_2}$) over the random choice of $\hat{u}, \hat{v} \pmod{p_2}$. Hence, except for negligible probability ${}^1\!/_{p_2}$, $\mathcal{B}$ computes a generator $G_2 \in \mathbb{G}_{p_2}$ and outputs a correct bit in Line 27.

We deduce that it holds

$$\begin{aligned}
\Pr\left[\mathcal{B}'(D, Z_1) = 1 \,\middle|\, \text{CipherAbort} \wedge \overline{\text{Abort}}\right] &\geq 1 - {}^1\!/_{p_2} \\
\Pr\left[\mathcal{B}'(D, Z_0) = 1 \,\middle|\, \text{CipherAbort} \wedge \overline{\text{Abort}}\right] &\leq {}^1\!/_{p_2} \quad .
\end{aligned}$$

Now, we can continue the analysis from above

$$\begin{aligned}
\text{Adv}_{\mathcal{B}'}^{\text{SD2}}(\lambda) &\geq \Pr[\text{CipherAbort}] \cdot (\Pr[\mathcal{B}'(D, Z_1) = 1 \mid \text{CipherAbort}] \\
&\qquad\qquad - \Pr[\mathcal{B}'(D, Z_0) = 1 \mid \text{CipherAbort}]) \\
&\overset{(\star)}{=} \Pr[\text{CipherAbort}] \cdot \Pr\left[\overline{\text{Abort}} \,\middle|\, \text{CipherAbort}\right] \cdot \\
&\quad \left(\Pr\left[\mathcal{B}'(D, Z_1) = 1 \,\middle|\, \text{CipherAbort} \wedge \overline{\text{Abort}}\right]\right. \\
&\quad\quad \left.- \Pr\left[\mathcal{B}'(D, Z_0) = 1 \,\middle|\, \text{CipherAbort} \wedge \overline{\text{Abort}}\right]\right) \\
&\geq \Pr[\text{CipherAbort}] \cdot (1 - \Pr[\text{Abort} \mid \text{CipherAbort}]) \cdot (1 - {}^2\!/_{p_2}) \\
&\geq \Pr[\text{CipherAbort}] - {}^2\!/_{p_2} - \Pr[\text{Abort} \wedge \text{CipherAbort}] \\
&\geq \Pr[\text{CipherAbort}] - {}^2\!/_{p_2} - {}^{q_{\text{dec1}}}\!/_{p_1} \quad .
\end{aligned}$$

Equation $(\star)$ holds since conditionally on the event Abort, $\mathcal{B}'$ outputs 1 with probability ${}^1\!/_2$ independently of Z and hence,

$$\Pr[\mathcal{B}'(D, Z_1) = 1 \mid \text{CipherAbort} \wedge \text{Abort}] - \Pr[\mathcal{B}'(D, Z_0) = 1 \mid \text{CipherAbort} \wedge \text{Abort}] = 0 \quad .$$

In the following step we used the previous estimations. Finally, in the last two inequalities we used only simple estimations and $\Pr[\text{Abort}] \leq {}^{q_{\text{dec1}}}\!/_{p_1}$, explained above. This proves the lemma. □

From $\mathbf{G'_{q_1+3}}$ to $\mathbf{G_{Final}}$. G_{Final} is as G'_{q_1+3}, but the key K^* is chosen uniformly at random independetly of b.

Changes in $\mathrm{G}_{\mathrm{Final}}$ compared to G'_{q_1+3}:
Exchange $\langle 6 \rangle$ for

– Set $\mathrm{K}^* \leftarrow \mathbb{G}_{\mathrm{T}}$.

Lemma D.18. *For every* des $\in \Omega$ *and every ppt algorithm* $\mathcal{A}$ *there exists a ppt algorithm* $\mathcal{B}$ *such that for every security parameter* λ *it holds*

$$\left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}'_{q_1+3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{\mathrm{Final}}}(\lambda, \mathrm{des})\right| \leq \mathrm{Adv}_{\mathcal{B}}^{\mathrm{SD3}}(\lambda) + \frac{2}{p_2} \enspace .$$

The running time of $\mathcal{B}$ *is essentially the same as the running time of* $\mathcal{A}$.

Proof. Given a ppt adversary $\mathcal{A}$, that can distinguish between both games, we construct a ppt algorithm $\mathcal{B}$ which breaks Assumption SD3 with the same advantage. Algorithm $\mathcal{B}$ is essentially the same as in the original reduction of Attrapadung (see Lemma 35 in [3]). We additionally have to show how to answer the decapsulation queries. $\mathcal{B}$ against SD3 gets an input that includes $g_1^{\alpha} X_2$ and implicitly sets $\mathrm{msk} := \alpha$. Furthermore, $\mathcal{A}$ gets no information about X_2. Hence, we can use $g_1^{\alpha} X_2$ in order to answer the decapsulation queries as defined in both games. Let des $\in \Omega$ be arbitrary but fixed. $\mathcal{B}$ is given des $\in \Omega$ in addition to its input (D, Z) from experiment SD3 as explained in Remark D.1.

$\mathcal{B}$ is a ppt algorithm with respect to λ by construction. In particular, random elements from $\mathbb{G}_{p_3}$ can be chosen using generator $g_3 \in \mathbb{G}_{p_3}$, and all exponents can be computed explicitly. Next, we analyze the view of $\mathcal{A}$ and the success probability of $\mathcal{B}$.

By construction of $\mathcal{B}$ it holds $Y = \mathrm{e}(g_1, g_1)^{\alpha}$. Hence, the master secret key msk is implicitly set to α, which is properly distributed by the definition of Experiment SD3. The public parameters are correctly generated by construction. The semi-functional generator of $\mathbb{G}_{p_2}$ is set to g_2. Also this generator is properly distributed by the definition of Experiment SD3. Furthermore, the semi-functional elements $\hat{\boldsymbol{h}}$, $\hat{u}$ and $\hat{v}$ will be implicitly set to $\boldsymbol{h}' \pmod{p_2}$, $u \pmod{p_2}$, and $v \pmod{p_2}$ respectively. These elements are properly distributed by construction. In particular, these values are independent of the corresponding values modulo p_1 by the Chinese Remainder Theorem. Furthermore, we recall that by the definition of Experiment SD3, there exist $x_2, y_2 \in \mathbb{Z}_{p_2}^*$ such that $X_2 = g_2^{x_2}$ and $Y_2 = g_2^{y_2}$.

Now, consider keys generated in Phase I. By construction of $\mathcal{B}$ it holds:

$$\begin{aligned}\boldsymbol{K} &= (g_1^{\alpha} X_2)^{\boldsymbol{k}(1,\boldsymbol{0},\boldsymbol{0})} \cdot g_1^{\boldsymbol{k}\left(0,\boldsymbol{r},\boldsymbol{h}'\right)} \cdot g_2^{\boldsymbol{k}\left(\hat{\alpha}'_j,\boldsymbol{0},\boldsymbol{0}\right)} \cdot \boldsymbol{R}_3 \\ &= g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{0},\boldsymbol{0})+\boldsymbol{k}\left(0,\boldsymbol{r},\boldsymbol{h}'\right)} \cdot g_2^{\boldsymbol{k}(x_2,\boldsymbol{0},\boldsymbol{0})+\boldsymbol{k}\left(\hat{\alpha}'_j,\boldsymbol{0},\boldsymbol{0}\right)} \cdot \boldsymbol{R}_3 \\ &= g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot g_2^{\boldsymbol{k}\left(x_2+\hat{\alpha}'_j,\boldsymbol{0},\boldsymbol{0}\right)} \cdot \boldsymbol{R}_3 \enspace .\end{aligned}$$

This is a correctly generated semi-functional key of Type 3. The normal components are properly distributed due to the choice of $\boldsymbol{r}$ and $\boldsymbol{R}_3$. The semi-functional component $\hat{\alpha}$ is set to $\hat{\alpha}_j = x_2 + \hat{\alpha}'_j \pmod{p_2}$, which is correctly distributed due to the choice of $\hat{\alpha}'_j$. In Phase II the keys are generated in the same way but with $\hat{\alpha} = x_2 + \hat{\alpha}'$ for all keys, which is properly distributed due to the choice of $\hat{\alpha}' \pmod{p_2}$. It is important to notice that the values $\hat{\alpha}_j$ and $\hat{\alpha}$ are independent of the value x_2.

Next, consider the challenge encapsulation. By construction of $\mathcal{B}$ it holds:

$$\begin{aligned}\boldsymbol{C}^* &= (g_1^s Y_2)^{\boldsymbol{c}\left(1,\boldsymbol{s}',\boldsymbol{h}'\right)} & C''^* &= (g_1^s Y_2)^{u\cdot t^*+v} \\ &= g_1^{s\cdot\boldsymbol{c}\left(1,\boldsymbol{s}',\boldsymbol{h}'\right)} \cdot g_2^{y_2\cdot\boldsymbol{c}\left(1,\boldsymbol{s}',\hat{\boldsymbol{h}}\right)} & &= g_1^{s\cdot(u\cdot t^*+v)} \cdot g_2^{y_2\cdot(u\cdot t^*+v)} \\ &= g_1^{\boldsymbol{c}\left(s,s\cdot\boldsymbol{s}',\boldsymbol{h}\right)} \cdot g_2^{\boldsymbol{c}\left(y_2,y_2\cdot\boldsymbol{s}',\hat{\boldsymbol{h}}\right)}, & &= \left(U_1^{t^*} \cdot V_1\right)^s \cdot \left(g_2^{\hat{u}\cdot t^*+\hat{v}}\right)^{y_2} \enspace .\end{aligned}$$

We claim that CT^* is a properly distributed semi-functional encapsulation except for negligible probability. The random values s and $\boldsymbol{s}$ of the normal components are set to $s \pmod{p_1}$ and $s \cdot \boldsymbol{s}'$ respectively. The value s is properly distributed due to the choice of $s \pmod{p_1}$. Vector $\boldsymbol{s}$ is properly distributed due to the choice of $\boldsymbol{s}' \pmod{p_1}$ as long as $s \neq 0 \pmod{p_2}$. The opposite happens with negligible probability $1/p_2$. The random values of the semi-functional components are set to $\hat{s} = y_2$ and $\hat{\boldsymbol{s}} = y_2 \cdot \boldsymbol{s}'$. The value $\hat{s}$ is properly distributed due to the choice of $y_2 \pmod{p_2}$. The value $\hat{\boldsymbol{s}}$ is properly distributed due to the

Algorithm 14: $\mathcal{B}$ against Assumption SD3

Input : (D, Z, des).

Require: $D = (\mathbb{GD}_N, g_1, g_1^\alpha X_2, g^s Y_2, g_2, g_3)$, $Z \in \mathbb{G}_\mathrm{T}$, $\mathrm{des} \in \Omega$.

Setup
- Set $\kappa := (\mathrm{des}, N)$ and compute $n := \mathrm{Param}(\kappa)$. Set $Y := \mathrm{e}(g_1, g_1^\alpha X_2)$.
- Pick $\boldsymbol{h}' \leftarrow \mathbb{Z}_N^n$ and $u, v \leftarrow \mathbb{Z}_N$. Compute $g_1^{\boldsymbol{h}'}$, $U_1 := g_1^u$ and $V_1 := g_1^v$.
- Choose a hash function $\mathrm{H} \leftarrow \mathcal{H}_\kappa$.
- Define $\mathrm{pp}_\kappa := \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}'}, U_1, V_1, g_3, Y, \mathrm{H}\right)$. Set $j := 0$
- Simulate $\mathcal{A}$ on input pp_κ.

Phase I
- **CoveredKeyGen** (kInd_i) with $\mathrm{kInd}_i \in \mathbb{X}_\kappa$:
  - Store (i, kInd_i).
- **Open** (i):
  - Set $j := j + 1$
  - Compute $(\boldsymbol{k}, m_2) := \mathrm{Enc1}(\mathrm{kInd})$. Let $m_1 := |\boldsymbol{k}|$.
  - Pick $\hat{\alpha}'_j \leftarrow \mathbb{Z}_N$, $\boldsymbol{r} \leftarrow \mathbb{Z}_N^{m_2}$, $\boldsymbol{R}_3 \leftarrow \mathbb{G}_{p_3}^{m_1}$ and compute
  $$\boldsymbol{K} := (g_1^\alpha X_2)^{\boldsymbol{k}(1,\boldsymbol{0},\boldsymbol{0})} \cdot g_1^{\boldsymbol{k}\left(0,\boldsymbol{r},\boldsymbol{h}'\right)} \cdot g_2^{\boldsymbol{k}\left(\hat{\alpha}'_j,\boldsymbol{0},\boldsymbol{0}\right)} \cdot \boldsymbol{R}_3 \ .$$
  - Return $\mathrm{sk}_i = (\mathrm{kInd}_i, \boldsymbol{K})$
- **Decapsulate** (CT, i):
  - **if** *all restrictions are satisfied and* CT *pass the consistency checks* **then** return $\mathrm{K} = \mathrm{e}(g_1^\alpha X_2, C_1)$;

Challenge (given cInd^* from $\mathcal{A}$)
- Compute $(\boldsymbol{c}, w_2) := \mathrm{Enc2}(\kappa, \mathrm{cInd}^*)$. Let $|\boldsymbol{c}| = w_1$.
- Pick $\boldsymbol{s}' \leftarrow \mathbb{Z}_N^{w_1}$ and compute
$$\boldsymbol{C}^* := (g_1^s Y_2)^{\boldsymbol{c}\left(1,\boldsymbol{s}',\boldsymbol{h}'\right)}$$
- Compute the hash value $t^* = \mathrm{HInput}(\mathrm{cInd}^*, \boldsymbol{C}^*, \_)$ and
$$C''^* := (g_1^s Y_2)^{u \cdot t^* + v}$$
- Return $\mathrm{K}^* := Z$ and $\mathrm{CT}^* := (\mathrm{cInd}^*, \boldsymbol{C}^*, C''^*)$

Phase II
- As Phase I, but use $\hat{\alpha}' \leftarrow \mathbb{Z}_N$ instead of $\hat{\alpha}'_j$ for all keys.

Guess
- As defined in the experiment.

choice of $\boldsymbol{s}'$, since $y_2 \neq 0 \pmod{p_2}$. Hence, CT^* is a semi-functional encapsulation of key $\mathrm{K} = Y^s = Z_1$ except for negligible probability $1/p_2$.

If $Z = Z_0$ the key K is chosen uniformly and independently at random from $\mathbb{G}_\mathrm{T}$ as required in $\mathrm{G}_\mathrm{Final}$. The simulation of CT^* is properly distributed except for negligible probability $1/p_2$.

The decapsulation queries are answered as defined in the experiment using X_2, properly distributed by the definition of Experiment SD3. In particular, all generated keys are independent of X_2.

In summary, for every $\mathrm{des} \in \Omega$ and every $\mathcal{A}$ there exists a ppt algorithm $\mathcal{B}' = \mathcal{B}_\mathcal{A}(\cdot, \cdot, \mathrm{des})$ such that for every security parameter λ it holds

$$\begin{aligned}
\mathrm{Adv}_{\mathcal{B}'}^{\mathrm{SD2}}(\lambda) &= \left|\Pr\left[\mathcal{B}'(D, Z_0) = 1\right] - \Pr\left[\mathcal{B}'(D, Z_1) = 1\right]\right| \\
&\geq \left|\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_\mathrm{Final}}(\lambda, \mathrm{des}) - \left(\frac{1}{2} + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+3}}(\lambda, \mathrm{des})\right)\right| - \frac{2}{p_2} \ . \\
&= \left|\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_\mathrm{Final}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}_{q_1+3}}(\lambda, \mathrm{des})\right| - \frac{2}{p_2} \ .
\end{aligned}$$

The second equation holds since $\mathcal{B}$ almost perfectly simulates $\mathrm{G}_\mathrm{Final}$ and G_{q_1+3} if $Z = Z_0$ and $Z = Z_1$ respectively. Furthermore, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ wins the corresponding experiment. This proves the lemma. □

$\mathbf{G_{Final}}$ **and the final analysis.** In this last game the adversary gets no information about the challenge bit and hence, its advantage is equal to zero.

Lemma D.19. *For every ppt algorithm $\mathcal{A}$ and every* $\mathrm{des} \in \Omega$ *there exists a ppt algorithm $\mathcal{B}_3$ such that for every security parameter λ it holds*

$$\mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G_{Final}}}(\lambda, \mathrm{des}) = 0 \ .$$

Summing up all factors from Lemma D.7 to Lemma D.19 we get

$$\begin{aligned}
\text{Adv-aP-KEM}_{\Pi,\mathcal{A}}^{\mathrm{aCCA}}(\lambda, \mathrm{des}) &= \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G_{Real}}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G_{resH}}}(\lambda, \mathrm{des}) \\
&\quad + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G_{resH}}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G_{resQ}}}(\lambda, \mathrm{des}) \\
&\quad + \dots \\
&\quad + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G}'_{q_1+3}}(\lambda, \mathrm{des}) - \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G_{Final}}}(\lambda, \mathrm{des}) \\
&\quad + \mathrm{Adv}_{\Pi,\mathcal{A}}^{\mathrm{G_{Final}}}(\lambda, \mathrm{des}) \\
&\leq \mathrm{Adv}_{\mathcal{H},\mathcal{B}_1}^{\mathrm{CR}}(\lambda, \mathrm{des}) + \mathrm{Adv}_{\mathcal{B}_2}^{\mathrm{SD1}}(\lambda) + \mathrm{Adv}_{\mathcal{B}_4}^{\mathrm{SD3}}(\lambda) \\
&\quad + (2q_1 + 4) \cdot \mathrm{Adv}_{\mathcal{B}_3}^{\mathrm{SD2}}(\lambda) + \mathrm{Adv}_{\mathrm{P},\mathcal{B}_6}^{\mathrm{SMH}}(\lambda, \mathrm{des}) \\
&\quad + q_1 \cdot \mathrm{Adv}_{\mathrm{P},\mathcal{B}_5}^{\mathrm{CMH}}(\lambda, \mathrm{des}) + {}^{q_{\mathrm{dec1}}}/_{p_1} + {}^{4}/_{p_2} \ .
\end{aligned}$$

This finally proves Theorem 4.1.

E Verifiability for Regular Pair Encoding Schemes

In this section we prove that regular pair encoding schemes are verifiable. We first present the formal definition of the regular pair encoding schemes from [2].

In this section we denote the encapsulation variable X_s by X_{s_0}. Furthermore, we denote the coefficients of polynomials $k_\tau \in \boldsymbol{k}$ in the key encodings $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, $m_1 = |\boldsymbol{k}|$ as follows

$$\forall_{\tau \in [m_1]} : k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i \in [m_2]} \left(a_{\tau,i} \cdot \mathrm{X}_{r_i} + \sum_{j \in [n]} \left(a_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{r_i} \right) \right) \ .$$

The coefficients of polynomials $c_\tau \in \boldsymbol{c}$ in the ciphertext encodings $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$ are denoted by

$$\forall_{\tau \in [w_1]} : c_\tau = \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} \left(b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right) \right) \ .$$

Definition E.1. *A pair encoding scheme* $\mathrm{P} = (\mathrm{Param}, \mathrm{Enc1}, \mathrm{Enc2}, \mathrm{Pair})$ *for domain-transferable predicate family $\mathcal{R}_{\Omega,\Sigma}$ is called* ***regular*** *if the following properties hold for every predicate index* $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$, *every* $\mathrm{kInd} \in \mathbb{X}_\kappa$ *and* $\mathrm{cInd} \in \mathbb{Y}_\kappa$ *which satisfy* $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$. *Suppose* $((c_1, \dots, c_{w_1}), w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$ *and* $((k_1, \dots, k_{m_1}), m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$.

1. *(Normality) There is an integer $\hat{\tau} \in [w_1]$ such that it holds*

$$c_{\hat{\tau}} = \mathrm{X}_s \ .$$

   W.l.o.g. we assume that $\hat{\tau} = 1$.
2. *Let* $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$ *be arbitrary. Then, for every $\tau \in [m_1]$ and every $\tau' \in [w_1]$ it holds*

$$\exists_{i \in [m_2]} \exists_{j \in [n]} \exists_{i' \in [w_2]_0} \exists_{j' \in [n]} : (a_{\tau,i,j} \neq 0 \wedge b_{\tau',i',j'} \neq 0) \qquad \textit{implies} \qquad e_{\tau,\tau'} = 0 \ .$$

3. *For every $i \in [m_2]$ such that there is no $\hat{\tau} \in [m_1]$ with $k_{\hat{\tau}} = \mathrm{X}_{r_i}$ it holds*

$$\forall_{\tau \in [m_1]} \forall_{j \in [n]} : a_{\tau,i,j} = 0 \ .$$

4. For every $i \in [w_2]$ such that there is no $\hat{\tau} \in [w_1]$ with $c_{\hat{\tau}} = \mathrm{X}_{s_i}$ it holds

$$\forall_{\tau \in [w_1]} \forall_{j \in [n]} : b_{\tau,i,j} = 0 \ .$$

Before proving Theorem 4.2 we present the following lemma, which leads to simple verification algorithms for the most known pair encoding schemes.

Lemma E.1. *Suppose $\mathcal{R}_{\Omega,\Sigma}$ is a domain-transferable predicate family,* P *is a pair encoding scheme for $\mathcal{R}_{\Omega,\Sigma}$, and $\mathcal{G}$ is an appropriate group generator. If for every $\kappa \in \Omega \times \Sigma$, every* cInd $\in \mathbb{Y}_\kappa$ *with $(\boldsymbol{c}, w_2) = \mathrm{Enc1}(\mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$, and every $i \in [w_2]_0$ there exists an index $\tau_i \in [w_1]$ such that $c_{\tau_i} = \mathrm{X}_{s_i}$, then* P *is verifiable with respect to $\mathcal{G}$ according to Definition 3.3.*

Proof. Let $\mathcal{R}_{\Omega,\Sigma}$ be an arbitrary but fixed domain-transferable predicate family and P be an arbitrary but fixed pair encoding scheme for $\mathcal{R}_{\Omega,\Sigma}$, which satisfies the property from the lemma. P is a normal encoding by definition, since among others, there exists $\tau_0 \in [w_1]$ such that $c_{\tau_0} = \mathrm{X}_{s_0}$. In order to prove the verifiability property of P, we construct an appropriate verification algorithm Vrfy for P. We assume w.l.o.g. that ciphertext encodings of P do not contain any polynomials multiple times, since these are redundant. Otherwise the corresponding group element in $\boldsymbol{C}$ must be checked for equality.

By Definition 3.3, Vrfy is given $\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, \mathrm{kInd}, \mathrm{cInd}, \boldsymbol{E}, \boldsymbol{C}\right)$ as input. The elements are as follows: λ is a security parameter, des $\in \Omega$, $\mathbb{GD} \in \left[\mathcal{G}\left(1^\lambda\right)\right]$ and $\mathbb{GD}_N$ is the corresponding restricted group description, $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$, $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, where $n = \mathrm{Param}(\kappa)$, kInd $\in \mathbb{X}_\kappa$ and cInd $\in \mathbb{Y}_\kappa$ with $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$, $\boldsymbol{C} \in \mathbb{G}^{w_1}$, where $(\boldsymbol{c}, w_2) = \mathrm{Enc1}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$.

We make use of the observation from Remark 3.1 and construct an algorithm Vrfy which checks if there exist $s_0 \in \mathbb{Z}_{p_1}$ and $\boldsymbol{s} \in \mathbb{Z}_{p_1}^{w_2}$ such that the $\mathbb{G}_{p_1}$ components of $\boldsymbol{C}$ are equal to $g_1^{\boldsymbol{c}(s_0,\boldsymbol{s},\boldsymbol{h})}$. By the definition of pair encoding schemes, for every $\tau \in [w_1]$ the polynomial $c_\tau \in \boldsymbol{c}$ has the form

$$c_\tau = \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} \left(b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \mathrm{X}_{s_i} \right) \right) \in \mathbb{Z}_N \left[\mathrm{X}_{s_0}, \mathbf{X}_{\boldsymbol{s}}, \mathbf{X}_{\boldsymbol{h}}\right] \ . \tag{6}$$

Due to the property of P from the lemma we can assume w.l.o.g. that $c_1 = \mathrm{X}_{s_0}, c_2 = \mathrm{X}_{s_1}, \ldots, c_{w_2+1} = \mathrm{X}_{s_{w_2}}$. Hence, the $\mathbb{G}_{p_1}$ components of the corresponding elements $C_1, \ldots, C_{w_2+1} \in \boldsymbol{C}$ particularly determine elements s_0 and $\boldsymbol{s} = (s_1, \ldots, s_{w_2})$ modulo p_1. Namely, for every $i \in [w_2]_0$ there exists unique $s_i \in \mathbb{Z}_{p_1}$ such that the $\mathbb{G}_{p_1}$ component of C_{i+1} is equal to $g_1^{s_i} = g_1^{c_{i+1}(s_0,\boldsymbol{s},\boldsymbol{h})}$. Correctness of remaining elements $C_\tau \in \boldsymbol{C}$ for $w_2 + 1 < \tau \leq w_1$ can be checked as follows:

$$\mathrm{e}\left(C_\tau, g_1\right) \stackrel{?}{=} \prod_{i \in [w_2]_0} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j \in [n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}} \right) \ . \tag{7}$$

The checks are constructed directly from (6). We deduce that for every $\tau \in [w_1] \setminus [w_2 + 1]$ the check of C_τ is satisfied if and only if the $\mathbb{G}_{p_1}$ components of C_τ is equal to $g_1^{c_\tau(s_0,\boldsymbol{s},\boldsymbol{h})}$, since

$$\begin{aligned}
&\prod_{i \in [w_2]_0} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j \in [n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}} \right) \\
&= \prod_{i \in [w_2]_0} \mathrm{e}\left(g_1^{s_i}, g_1^{b_{\tau,i} + \sum_{j \in [n]} (b_{\tau,i,j} \cdot h_j)} \right) \\
&= \mathrm{e}\left(g_1, g_1\right)^{\sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot s_i + \sum_{j \in [n]} (b_{\tau,i,j} \cdot s_i \cdot h_j) \right)} \\
&= \mathrm{e}\left(g_1, g_1\right)^{c_\tau(s_0,\boldsymbol{s},\boldsymbol{h})} \ .
\end{aligned}$$

Hence, Vrfy performs the checks in (7) for every $\tau \in [w_1] \setminus [w_2 + 1]$ and outputs 1, if and only if these checks are satisfied. The lemma follows by Remark 3.1. $\square$

The algorithms constructed directly from the proof of Lemma E.1 can be optimized for concrete schemes in different ways. The proof shows only the main idea. On the one hand, one should look for reducing the number of pairing computations in (7) using usual techniques. Note furthermore, that for the concrete schemes the number of coefficients unequal zero is small. On the other hand, the constructed algorithm does not use kInd and $\boldsymbol{E}$ given as input. In several predicate encryption schemes (e.g. attribute-based schemes) only few elements form $\boldsymbol{C}$ may be relevant for the decapsulation, that is some columns of $\boldsymbol{E} \in \mathbb{Z}_N^{m_1 \times w_1}$ contain only zeros. In this case, the verification algorithm does not have to check the corresponding elements from $\boldsymbol{C} \in \mathbb{G}^{w_1}$ in order to ensure the soundness property. This results in more efficient verification algorithms. Formally, we have to perform the check in 7 for $k \in [w_1] \setminus [w_2+1]$ if and only if the column k in $\boldsymbol{E}$ is unequal $\mathbf{0}$. This still ensures the soundness property, since the values in $\boldsymbol{C}$ which are not checked do not affect the result of $\mathrm{e}\left(g_1^{\boldsymbol{k}(\alpha,\boldsymbol{r},\boldsymbol{h})\cdot\boldsymbol{E}}, \boldsymbol{C}\right)$.

Now we extend this result to regular pair encoding schemes.

Proof. (Proof of Theorem 4.2) Let $\mathcal{R}_{\Omega,\Sigma}$ be an arbitrary but fixed domain-transferable predicate family and P be an arbitrary regular pair encoding scheme for $\mathcal{R}_{\Omega,\Sigma}$. P is a normal encoding by definition, due to the first property of regular pair encoding schemes. In order to prove the verifiability property of P, we construct an appropriate verification algorithm Vrfy for P.

By Definition 3.3, Vrfy is given $\left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, \mathrm{kInd}, \mathrm{cInd}, \boldsymbol{E}, \boldsymbol{C}\right)$ as input. The elements are as follows: λ is a security parameter, $\mathrm{des} \in \Omega$, $\mathbb{GD} \in \left[\mathcal{G}\left(1^\lambda\right)\right]$ and $\mathbb{GD}_N$ is the corresponding restricted group description, $\kappa = (\mathrm{des}, N) \in \Omega \times \Sigma$, $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, where $n = \mathrm{Param}(\kappa)$, $\mathrm{kInd} \in \mathbb{X}_\kappa$ and $\mathrm{cInd} \in \mathbb{Y}_\kappa$ with $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$, $\boldsymbol{E} \in [\mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})]$, $\boldsymbol{C} \in \mathbb{G}^{w_1}$, where $(\boldsymbol{c}, w_2) = \mathrm{Enc1}(\kappa, \mathrm{cInd})$, $w_1 = |\boldsymbol{c}|$.

We make use of the observation from Remark 3.1 and construct an algorithm Vrfy which checks if there exist $s_0 \in \mathbb{Z}_{p_1}$ and $\boldsymbol{s} \in \mathbb{Z}_{p_1}^{w_2}$ such that the $\mathbb{G}_{p_1}$ components of $\boldsymbol{C}$ are equal to $g_1^{\boldsymbol{c}(s_0,\boldsymbol{s},\boldsymbol{h})}$.

Let $I \subseteq [w_2]_0$ be an index set such that $i \in I$ if and only if there exists $\tau \in [w_1]$ with $c_\tau = \mathrm{X}_{s_i}$. Furthermore, define $I' := [w_2]_0 \setminus I$. The case $I = [w_2]_0$ is covered by Lemma E.1. Hence, we only consider the case $I \subset [w_2]_0$. Note that $c_1 = \mathrm{X}_{s_0}$ by the first property of regular encodings and hence, $0 \in I$. Let $l := |I| - 1$. Assume w.l.o.g. that $I = [l]_0$ which implies $I' = \{l+1, \ldots, w_2\}$. Furthermore, assume w.l.o.g. that for every $i \in I$ it holds $c_{i+1} = \mathrm{X}_{s_i}$. Hence, the $\mathbb{G}_{p_1}$ components of the elements $C_1, \ldots, C_{l+1} \in \boldsymbol{C}$ particularly determine elements $s_0, s_1, \ldots, s_l \pmod{p_1}$. Namely, for every $i \in I$ there exists unique $s_i \in \mathbb{Z}_{p_1}$ such that the $\mathbb{G}_{p_1}$ component of C_{i+1} is equal to $g_1^{s_i} = g_1^{c_{i+1}(s_0,\boldsymbol{s},\boldsymbol{h})}$, where only the first l elements of $\boldsymbol{s} = (s_1, \ldots, s_l, \_, \ldots, \_)$ are relevant.

We have to check if there exist $s_{l+1}, \ldots, s_{w_2} \in \mathbb{Z}_{p_1}$ such that the $\mathbb{G}_{p_1}$ components of remaining group elements $C_{l+2}, \ldots, C_{w_1} \in \boldsymbol{C}$ are consistent with all values $s_0, \ldots, s_{w_2}$.

Due to the fourth property of regular pair encodings, it holds for every $\tau \in [w_1]$:

$$c_\tau = \sum_{i \in [w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i}\right) + \sum_{i \in I} \sum_{j \in [n]} \left(b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \mathrm{X}_{s_i}\right) \quad .$$

We slightly reorder the summands with respect to I and I' and get the following form:

$$c_\tau = \sum_{i \in I'} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i}\right) + \sum_{i \in I} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j \in [n]} \left(b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \mathrm{X}_{s_i}\right)\right) \quad . \tag{8}$$

For those $\tau \in [w_1] \setminus [l+1]$, where the first summand in (8) of polynomial c_τ, is equal zero, we can perform the check for C_τ similar to the checks in (7) in the proof of Lemma E.1:

$$\mathrm{e}\left(C_\tau, g_1\right) \stackrel{?}{=} \prod_{i \in I} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j \in [n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}}\right) \quad . \tag{9}$$

The checks are satisfied if and only if the $\mathbb{G}_{p_1}$ components of C_τ for every τ as above are equal to $g_1^{c_\tau(s_0,\boldsymbol{s},\boldsymbol{h})}$, where only the first l elements of $\boldsymbol{s} = (s_1, \ldots, s_l, \_, \ldots, \_)$ are relevant. Namely, it holds

$$\begin{aligned}
&\prod_{i\in I} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j\in[n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}}\right) \\
&= \prod_{i\in I} \mathrm{e}\left(g_1^{s_i}, g_1^{b_{\tau,i}+\sum_{j\in[n]}(b_{\tau,i,j}\cdot h_j)}\right) \\
&= \mathrm{e}\left(g_1, g_1\right)^{\sum_{i\in I}\left(b_{\tau,i}\cdot s_i+\sum_{j\in[n]}(b_{\tau,i,j}\cdot s_i\cdot h_j)\right)} \quad .
\end{aligned}$$

Vrfy outputs 0, if one of these checks is not satisfied.

Let us assume w.l.o.g that t elements from $\boldsymbol{C}$, namely $C_{l+2}, \ldots, C_{l+t+1}$, can be checked using this kind of checks. It remains to check the elements $C_{l+t+2}, \ldots, C_{w_1} \in \boldsymbol{C}$. By construction, for all $\tau \in [w_1]\backslash[l+t+1]$ the first summand of c_τ in (8) is unequal zero. We first eliminate from every C_τ group elements which corresponds to the correctly evaluated second summand from (8). Namely, Vrfy computes for every $\tau \in [w_1] \setminus [l + t + 1]$:

$$Y_\tau := \mathrm{e}\left(C_\tau, g_1\right) \cdot \left(\prod_{i\in I} \mathrm{e}\left(C_{i+1}, g_1^{b_{\tau,i}} \cdot \prod_{j\in[n]} \left(g_1^{h_j}\right)^{b_{\tau,i,j}}\right)\right)^{-1} \quad . \tag{10}$$

Hence, it holds

$$Y_\tau = \mathrm{e}\left(C_k \cdot g_1{}^{-\sum_{i\in I}\left(b_{\tau,i}\cdot s_i+\sum_{j\in[n]}(b_{\tau,i,j}\cdot s_i\cdot h_j)\right)}, g_1\right) \quad .$$

Note that all Y_τ's are element of the order p_1 subgroup of $\mathbb{G}_\mathrm{T}$ and $\mathrm{e}\left(g_1, g_1\right)$ is a generator of this subgroup. Hence, for every $\tau \in [w_1] \setminus [l + t + 1]$ there exists unique $y_\tau \in \mathbb{Z}_{p_1}$ such that $Y_\tau = \mathrm{e}\left(g_1, g_1\right)^{y_\tau}$.

Next, we have to check if there exist $s_{l+1}, \ldots, s_{w_2} \in \mathbb{Z}_{p_1}$ such that for every $\tau \in [w_1] \setminus [l + t + 1]$ it holds

$$Y_\tau = \mathrm{e}\left(g_1, g_1\right)^{\sum_{i\in I'}(b_{\tau,i}\cdot s_i)} \quad ,$$

or rather

$$y_\tau = \sum_{i\in I'} (b_{\tau,i} \cdot s_i) \pmod{p_1} \quad . \tag{11}$$

This can be verified as follows.

Let us consider (11) more abstractly. Recall that $I' = \{l + 1, \ldots, w_2\}$. Let $\varsigma := w_1 - (l + t + 1)$ and $\varrho := w_2 - l$. These are the number of group elements $Y_\tau \in \mathbb{G}_\mathrm{T}$ which have to be checked and the size of I' respectively. Let $\boldsymbol{M} = (m_{i,j})_{i\in[\varsigma],j\in[\varrho]} \in \mathbb{Z}_N^{\varsigma\times\varrho}$ be the matrix of coefficients $b_{\tau,i}$ corresponding to (11). Hence, we have to check if there exist $\boldsymbol{s} \in \mathbb{Z}_{p_1}^{\varrho}$ such that

$$\boldsymbol{y} = \boldsymbol{M} \cdot \boldsymbol{s} \pmod{p_1} \quad ,$$

where $\boldsymbol{y} = (y_\tau)_{\tau\in[w_1]\setminus[l+t+1]} \in \mathbb{Z}_{p_1}^{\varsigma}$ and $\boldsymbol{s} = (s_i)_{i\in I'} \in \mathbb{Z}_{p_1}^{\varrho}$ are vectors of the corresponding elements y_τ and s_i.

Using the Gaussian elimination algorithm (over $\mathbb{Z}_N$) on $\boldsymbol{M}$, Vrfy derives an invertible matrix $\boldsymbol{T} \in \mathbb{Z}_N^{\varsigma\times\varsigma}$ such that $\boldsymbol{T} \cdot \boldsymbol{M}$ is in reduced row echelon form. Under the assumption, that the factorization of N is a difficult task, we can ignore the case that a zero divisor is hit during the computation of the algorithm. Furthermore, since $\boldsymbol{T}$ is invertible over $\mathbb{Z}_N$, it will be also invertible over $\mathbb{Z}_{p_1}$ and it holds

$$\begin{aligned}
&\exists_{\boldsymbol{s}\in\mathbb{Z}_{p_1}^{\varrho}} : \boldsymbol{M} \cdot \boldsymbol{s} = \boldsymbol{y} \pmod{p_1} \\
\Leftrightarrow\ &\exists_{\boldsymbol{s}\in\mathbb{Z}_{p_1}^{\varrho}} : \boldsymbol{T} \cdot \boldsymbol{M} \cdot \boldsymbol{s} = \boldsymbol{T} \cdot \boldsymbol{y} \pmod{p_1} \\
\Leftrightarrow\ &\exists_{\boldsymbol{s}\in\mathbb{Z}_{p_1}^{\varrho}} : \mathrm{e}\left(g_1, g_1\right)^{\boldsymbol{T}\cdot\boldsymbol{M}\cdot\boldsymbol{s}} = \mathrm{e}\left(g_1, g_1\right)^{\boldsymbol{T}\cdot\boldsymbol{y}} \quad .
\end{aligned}$$

Vrfy checks the last of this statements. It computes all elements of

$$\boldsymbol{x} = \mathrm{e}\left(g_1, g_1\right)^{\boldsymbol{T}\cdot\boldsymbol{y}} \in \mathbb{G}_\mathrm{T}^{\varsigma} \tag{12}$$

using $\boldsymbol{T}$ and $\{Y_\tau = \mathrm{e}(g_1, g_1)^{y_\tau}\}_{\tau\in[w_1]\setminus[l+t+1]}$.

We claim that there exist $\boldsymbol{s} \in \mathbb{Z}_{p_1}^{\varrho}$ such that $\mathrm{e}(g_1, g_1)^{\boldsymbol{T}\cdot\boldsymbol{M}\cdot\boldsymbol{s}} = \mathrm{e}(g_1, g_1)^{\boldsymbol{T}\cdot\boldsymbol{y}}$ if and only if for every zero row τ in $\boldsymbol{T}\cdot\boldsymbol{M} \in \mathbb{Z}_N^{c\times\varrho}$ it holds $x_\tau = \mathbf{1}_{\mathbb{G}_{\mathrm{T}}}$. This is true since $\boldsymbol{T}\cdot\boldsymbol{M}$ is in reduced row echelon form. If this is not the case, Vrfy outputs 0. Otherwise, Vrfy outputs 1.

In summary, we deduce that Vrfy outputs 1 if and only if there exist $s_0 \in \mathbb{Z}_{p_1}$ and $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$ such that the $\mathbb{G}_{p_1}$ components of $\boldsymbol{C}$ are equal to $g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}$. Hence, according to Remark 3.1 P is a verifiable pair encoding scheme. $\square$

We note, that the Vrfy algorithm from the proof might seam to be quite complex. Indeed, the most known predicate based schemes are covered by much simpler algorithm from the proof of Lemma E.1. Furthermore, also this more involved algorithm has to check only those elements in $\boldsymbol{C}$, which are relevant with respect to the reconstruction matrix $\boldsymbol{E}$. That is, the checks in (9) have to be performed only for the relevant elements in $\boldsymbol{C}$ as well as the elements Y_τ have to be computed in (10) and have to be checked as described after (12) only for the relevant elements in $\boldsymbol{C}$. Finally, matrix $\boldsymbol{M}$ is fixed by cInd. For example, in the case of ciphertext-policy attribute-based encryption, where cInd is a monotone span program (MSP), $\boldsymbol{M}$ will be the matrix of the MSP.

F Further Proofs

In this section we present proofs of different lemmas and theorems from the work.

F.1 Correctness of the Framework

In this subsection we will first show that the algorithms of our framework from Subsection 3.3, especially the key generation algorithm and the encapsulation algorithm, are ppt algorithms with respect to the security parameter. Then, we present the correctness proof for our construction.

The computability of the algorithm can be easily checked except for the evaluation of the polynomials in the exponent of group elements. Hence, for the sake of completeness we explicitly show how to compute these elements.

Lemma F.1. *(Key computability) For every security parameter λ, every composite order group description $\mathbb{GD} = (p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_{\mathrm{T}}, \mathrm{e} : \mathbb{G}\times\mathbb{G}\to\mathbb{G}_{\mathrm{T}}) \in [\mathcal{G}(1^\lambda)]$ and every des $\in \Omega$ one can efficiently compute*

$$g_1^{\boldsymbol{k}(\alpha,\boldsymbol{r},\boldsymbol{h})} \in \mathbb{G}_{p_1}^{m_1} \ ,$$

given $\mathbb{GD}_N$, $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, $\alpha \in \mathbb{Z}_N$, and $\boldsymbol{r} \in \mathbb{Z}_N^{m_2}$. Here, $N = p_1p_2p_3$, $\mathbb{GD}_N$ is the restricted description of $\mathbb{GD}$, $\kappa = (\mathrm{des}, N)$, $n = \mathrm{Param}(\kappa)$, $\mathrm{kInd} \in \mathbb{X}_\kappa$ is arbitrary, $(\boldsymbol{k}, m_2) = \mathrm{Enc1}(\kappa, \mathrm{kInd})$, and $m_1 = |\boldsymbol{k}|$.

Proof. Due to the restrictions on polynomials, for every $\tau \in [m_1]$ the polynomials in $\boldsymbol{k}$ are of the form $k_\tau = a_\tau \cdot \mathrm{X}_\alpha + \sum_{i\in[m_2]}\left(a_{\tau,i}\cdot\mathrm{X}_{r_i} + \sum_{j\in[n]} a_{\tau,i,j}\cdot\mathrm{X}_{h_j}\cdot\mathrm{X}_{r_i}\right)$. Hence, given the coefficient of the polynomials (given by $\mathrm{Enc1}(\kappa, \mathrm{kInd})$), we can compute for every τ:

$$g_1^{k_\tau(\alpha,\boldsymbol{r},\boldsymbol{h})} = (g_1^\alpha)^{a_\tau}\cdot g_1^{\sum_{i\in[m_2]} a_{\tau,i}\cdot r_i}\cdot\prod_{j\in[n]}\left(g_1^{h_j}\right)^{\sum_{i\in[m_2]} a_{\tau,i,j}\cdot r_i} \ .$$

This proves the lemma. $\square$

The following lemma and the proof are analogous for the encapsulation algorithm.

Lemma F.2. *(Ciphertext computability) For every security parameter λ, every group description $\mathbb{GD} = (p_1, p_2, p_3, (g, \mathbb{G}), \mathbb{G}_{\mathrm{T}}, \mathrm{e} : \mathbb{G}\times\mathbb{G}\to\mathbb{G}_{\mathrm{T}}) \in [\mathcal{G}(1^\lambda)]$ and every des $\in \Omega$ one can efficiently compute*

$$g_1^{\boldsymbol{c}(s_0,\boldsymbol{s},\boldsymbol{h})} \in \mathbb{G}_{p_1}^{w_1} \ ,$$

given $\mathbb{GD}_N$, $g_1 \in \mathbb{G}_{p_1}$, $g_1^{\boldsymbol{h}} \in \mathbb{G}_{p_1}^n$, $s_0 \in \mathbb{Z}_N$, and $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$. Here, $N = p_1p_2p_3$, $\mathbb{GD}_N$ is the restricted description of $\mathbb{GD}$, $\kappa = (\mathrm{des}, N)$, $n = \mathrm{Param}(\kappa)$, $\mathrm{cInd} \in \mathbb{Y}_\kappa$ is arbitrary, $(\boldsymbol{c}, w_2) = \mathrm{Enc2}(\kappa, \mathrm{cInd})$, and $w_1 = |\boldsymbol{c}|$.

Proof. Due to the restrictions on polynomials, for every $\tau \in [w_1]$ the polynomials in $\boldsymbol{c}$ are of the form $c_\tau = \sum_{i\in[w_2]_0} \left(b_{\tau,i} \cdot \mathrm{X}_{s_i} + \sum_{j\in[n]} b_{\tau,i,j} \cdot \mathrm{X}_{h_j} \cdot \mathrm{X}_{s_i} \right)$. Hence, given the coefficient of the polynomials we can compute:

$$g_1^{c_\tau(s_0,\boldsymbol{s},\boldsymbol{h})} = g_1^{\sum_{i\in[w_2]_0} b_{\tau,i}\cdot s_i} \cdot \prod_{j\in[n]} \left(g_1^{h_j} \right)^{\sum_{i\in[w_2]_0} b_{\tau,i,j}\cdot s_i} \quad .$$

This proves the lemma. □

Next we present the correctness proof for our framework.

Proof. (*Correctness of P-KEM* Π *from Subsection 3.3*) The elements from $g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})}$ can be efficiently computed from $g_1^{\boldsymbol{h}}$, msk, and $\boldsymbol{r}$ due to Lemma F.1. Analogously, the elements from $g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}$ can be computed due to Lemma F.2. Since $\boldsymbol{K} \in \mathbb{G}^{m_1}$, the element $\boldsymbol{K}^{\boldsymbol{E}}$ in the decapsulation algorithm can be computed efficiently as described in Subsection 2.3.

Let λ, des $\in \Omega$, and $\left(\mathrm{msk}, \left(\mathrm{des}, \mathbb{GD}_N, g_1, g_1^{\boldsymbol{h}}, U_1, V_1, g_3, Y, \mathrm{H}\right)\right) \in \left[\mathrm{Setup}\left(1^\lambda, \mathrm{des}\right)\right]$ be arbitrary but fixed and let $N = (\mathrm{des}, N)$. Furthermore, let kInd $\in \mathbb{X}_\kappa$ and cInd $\in \mathbb{Y}_\kappa$ be arbitrary but fixed such that $\mathrm{R}_\kappa(\mathrm{kInd}, \mathrm{cInd}) = 1$. In turn, let $(\mathrm{kInd}, \boldsymbol{K}) \in [\mathrm{KeyGen}(\mathrm{msk}, \mathrm{kInd})]$ and $(\mathrm{K}, (\mathrm{cInd}, \boldsymbol{C}, C'')) \in [\mathrm{Encaps}(\mathrm{cInd})]$ be arbitrary. Suppose $\mathrm{K} = Y^s$ for some $s \in \mathbb{Z}_N$. Then, by construction, there exist $\boldsymbol{r} \in \mathbb{Z}_N^{m_2}$, $\boldsymbol{s} \in \mathbb{Z}_N^{w_2}$, and $\boldsymbol{R}_3 \in \mathbb{G}_{p_3}^{m_1}$ such that $\boldsymbol{K} = g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot \boldsymbol{R}_3$, and $\boldsymbol{C} = g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})}$, where $\boldsymbol{h} \pmod{p_1}$ is defined by $g_1^{\boldsymbol{h}} \in \mathrm{pp}_\kappa$. Furthermore, by construction $C'' = \left(U_1^t \cdot V_1\right)^s$, where t is the hash value for the encapsulation as defined in the algorithm Encaps. By the normality of pair encoding $C_1 = g_1^s$.

The decapsulation algorithm Decaps computes $\boldsymbol{E} \leftarrow \mathrm{Pair}(\kappa, \mathrm{kInd}, \mathrm{cInd})$. The checks in (4) are satisfied since the elements in $\boldsymbol{C}$ and C'' do not contain $\mathbb{G}_{p_3}$ components. The check in (3) is satisfied due to the form of C_1 and C''. The check in (5) is satisfied since Vrfy outputs 1 due to its completeness property.

After the checks the decapsulation algorithm outputs

$$\begin{aligned} \mathrm{e}\left(\boldsymbol{K}^{\boldsymbol{E}}, \boldsymbol{C}\right) &= \mathrm{e}\left(\left(g_1^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})} \cdot \boldsymbol{R}_3 \right)^{\boldsymbol{E}}, g_1^{\boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})} \right) \\ &= \mathrm{e}\left(g_1, g_1\right)^{\boldsymbol{k}(\mathrm{msk},\boldsymbol{r},\boldsymbol{h})\cdot \boldsymbol{E} \cdot \boldsymbol{c}(s,\boldsymbol{s},\boldsymbol{h})} = \mathrm{e}\left(g_1, g_1\right)^{\mathrm{msk}\cdot s} = \mathrm{K} \quad , \end{aligned}$$

where the third equation holds due to the correctness of the pair encoding scheme. Hence, the predicate based key encapsulation mechanism Π is correct by definition. □

F.2 Hardness of Factorization under Subgroup Decision Assumptions

Proof. (Proof of Lemma 2.1) We prove that the following Alg. 15 satisfies the required properties.

Algorithm 15: $\mathcal{B}$ against Assumption SD2 given a nontrivial factor of N

```
Input   : (D, Z, F)
Require: D = (GD_N, g_1, X_1X_2, Y_2Y_3, g_3), Z ∈ G, F ∈ N, 1 < F < N, and F | N.
Set a := min(F, N/F) and b := max(F, N/F);
if (Y_2Y_3)^b = 1_G then
    if e(Z^a, X_1X_2) = 1_{G_T} then output 0;
    else output 1;
if (X_1X_2)^b = 1_G then
    if e(Z^a, Y_2Y_3) = 1_{G_T} then output 0;
    else output 1;
if Z^b = 1_G then output 0;
else output 1;
```

$\mathcal{B}$ is a ppt algorithm with respect to λ by construction. Next, we analyze the success probability of the algorithm. Let $\hat{g}_1$, $\hat{g}_2$, $\hat{g}_3$ be arbitrary but fixed generators of $\mathbb{G}_{p_1}$, $\mathbb{G}_{p_2}$ and $\mathbb{G}_{p_3}$ respectively. By the definition of Experiment SD2 we have $X_1 = \hat{g}_1^{x_1}$, $X_2 = \hat{g}_2^{x_2}$, $Y_2 = \hat{g}_2^{y_2}$, $Y_3 = \hat{g}_3^{y_3}$, and $Z = \hat{g}_1^{z_1} \cdot \hat{g}_2^{z_2} \cdot \hat{g}_3^{z_3}$. Thereby x_1 and z_1 are uniformly distributed in $\mathbb{Z}_{p_1}^*$, x_2 and y_2 are uniformly distributed in $\mathbb{Z}_{p_2}^*$, y_3 and z_3 are uniformly distributed in $\mathbb{Z}_{p_3}^*$. Furthermore, $z_2 = 0$ if $Z = Z_0$, whereas z_2 is uniformly distributed in $\mathbb{Z}_{p_2}^*$ if $Z = Z_1$.

Given a non-trivial factor F of N we have three possible cases:

$$1)\ a = p_1 \wedge b = p_2 p_3, \quad 2)\ a = p_2 \wedge b = p_1 p_3, \quad 3)\ a = p_3 \wedge b = p_1 p_2\ .$$

The condition $(Y_2 Y_3)^b = 1_{\mathbb{G}}$ from Line 2 is satisfied if and only if $b = p_2 p_3$. In this case $a = p_1$ and the $\mathbb{G}_{p_1}$ component of Z disappears in Z^a. Hence, $Z = Z_0 \in \mathbb{G}_{p_1 p_3}$ if and only if $\mathrm{e}(Z^a, X_1 X_2) = 1_{\mathbb{G}_\mathrm{T}}$. Analogously, the condition $(X_1 X_2)^b = 1_{\mathbb{G}}$ from Line 5 is satisfied if and only if $b = p_1 p_2$. In this case $a = p_3$ and the $\mathbb{G}_{p_3}$ component of Z disappears in Z^a. Hence, $Z = Z_0 \in \mathbb{G}_{p_1 p_3}$ if and only if $\mathrm{e}(Z^a, Y_2 Y_3) = 1_{\mathbb{G}_\mathrm{T}}$. Due to the observations from above, Line 5 is executed if and only if $b = p_1 p_3$. In this case the $\mathbb{G}_{p_1}$ and the $\mathbb{G}_{p_3}$ components of Z disappears in Z^b. Hence, $Z = Z_0 \in \mathbb{G}_{p_1 p_3}$ if and only if $Z^b = 1_{\mathbb{G}}$.

In summary, under the assumption from above, $\mathcal{B}$ outputs 1 if and only if $Z = Z_1$. Hence, it holds $\Pr[\mathcal{B}(D, Z_1, F) = 1] = 1$and $\Pr[\mathcal{B}(D, Z_0, F) = 1] = 0$.Consequently,

$$|\Pr[\mathcal{B}(D, Z_0, F) = 1] - \Pr[\mathcal{B}(D, Z_1, F) = 1]| = 1$$

This proves the lemma. □