KEM Combiners

Federico Giacon[1], Felix Heuer[1], and Bertram Poettering[2]

[1] Ruhr University Bochum
{federico.giacon,felix.heuer}@rub.de
[2] Royal Holloway, University of London
bertram.poettering@rhul.ac.uk

Abstract. Key-encapsulation mechanisms (KEMs) are a common stepping stone for constructing public-key encryption. Secure KEMs can be built from diverse assumptions, including ones related to integer factorization, discrete logarithms, error correcting codes, or lattices. In light of the recent NIST call for post-quantum secure PKE, the zoo of KEMs that are believed to be secure continues to grow. Yet, on the question of which is the *most* secure KEM opinions are divided. While using the best candidate might actually not seem necessary to survive everyday life situations, placing a wrong bet can actually be devastating, should the employed KEM eventually turn out to be vulnerable.

We introduce KEM combiners as a way to garner trust from different KEM constructions, rather than relying on a single one: We present efficient black-box constructions that, given any set of 'ingredient' KEMs, yield a new KEM that is (CCA) secure as long as at least one of the ingredient KEMs is.

As building blocks our constructions use cryptographic hash functions and blockciphers. Some corresponding security proofs require idealized models for these primitives, others get along on standard assumptions.

Keywords: secure combiners, CCA security, practical constructions

1 Introduction

Motivation for PKE combiners. Out of the public-key encryption schemes RSA-OAEP, Cramer–Shoup, ECIES, and a scheme based on the LWE hardness assumption, which one is, security-wise, the best? This question has no clear answer, as all schemes have advantages and disadvantages. For instance, RSA-OAEP is based on the arguably best studied hardness assumption but requires a random oracle. Cramer–Shoup encryption does not require a random oracle but its security reduces 'only' to a decisional assumption (DDH). While one can give a security reduction for ECIES to a computational assumption (CDH), this reduction comes with a tightness gap much bigger than that of RSA-OAEP. On the other hand, the 'security-per-bit ratio' for elliptic curve groups is assumed to be much better than for RSA based schemes. Finally, the LWE scheme is the only quantum-resistant candidate, although the assumption is relatively new and arguably not yet well understood. All in all, the challenge of picking the most secure PKE scheme is arguably impossible to solve. Fortunately, the challenge can be side-stepped by using a 'PKE combiner': Instead of using only one scheme to encrypt a message, one uses all four of them, combining them in a way such that security of any implies security of their combination. Thus, when using a combiner, placing wrong bets is impossible. PKE combiners have been studied in [6,21] and we give some details on encryption combiners below.

Combiners for other cryptographic primitives. In principle, secure combiners can be studied for any cryptographic primitive. For some primitives they are easily constructed and known for quite some time. For instance, sequentially composing multiple independently keyed blockciphers to a single keyed permutation can be seen as implementing a (S)PRP combiner. PRFs can be combined by XORing their outputs into a single value. More intriguing is studying hash function combiners: Parallelly composing hash functions is a good approach if the goal is collision resistance, but pre-image resistance suffers from this. A sequential composition would be better with respect to the latter, but this again harms collision resistance. Hash function combiners that preserve both properties simultaneously exist and can be based on Feistel structures [9]. If indifferentiability from a random oracle is an additional goal, pure Feistel

systems become insecure and more involved combiners are required [10,11]. Recently, also combiners for indistinguishability obfuscation have been proposed [1,8]. For an overview of combiners in cryptography we refer to [14,13].

Our target: KEM combiners. Following the contemporary KEM/DEM design principle of public-key encryption [4], in this work we study combiners for key-encapsulation mechanisms (KEMs). That is, given a set of KEMs, an unknown subset of which might be arbitrarily insecure, we investigate how they can be combined to form a single KEM that is secure if at least one ingredient KEM is. How such a combiner is constructed certainly depends on the specifics of the security goal. For instance, if CPA security shall be reached then it can be expected that combining a set of KEMs by running the encapsulation algorithms in parallel and XORing the established session keys together is sufficient. However, if CCA security is intended this construction is obviously weak.

The focus of this paper is on constructing combiners for CCA security. We propose several candidates and analyze them.[3] We stress that our focus is on practicality, i.e., the combiners we propose do not introduce much overhead and are designed such that system engineers can easily adopt them. Besides the ingredient KEMs, our combiners also mix in further cryptographic primitives like blockciphers, PRFs, or hash functions. We consider this an acceptable compromise, since they make secure constructions very efficient and arguably are not exposed to the threats we want to hedge against. For instance, the damage that quantum computers do on AES and SHA256 are generally assumed to be limited and controllable, tightness gaps can effectively and cheaply be closed by increasing key lengths and block sizes, and their security is often easier to assume than that of number-theoretic assumptions. While, admittedly, for some of our combiners we do require strong properties of the symmetric building blocks (random oracle model, ideal cipher model, etc.), we also construct a KEM combiner that is, at a higher computational cost, secure in the standard model. In the end we offer a selection of combiners, all with specific security and efficiency features, so that for every need there is a suitable one.

1.1 Our results

The KEM combiners treated in this paper have a parallel structure: If the number of KEMs to be combined is n, a public key of the resulting KEM consists of a vector of n public keys, one for each ingredient; likewise for secret keys. The encapsulation procedure performs n independent encapsulations, one for each combined KEM. The ciphertext of the resulting KEM is simply the concatenation of all generated ciphertexts. The session key is obtained as a function W of keys and ciphertexts (which is arguably the *core function* of the KEM combiner). A first proposal for a KEM combiner would be to use as session key the value

$$K = H(k_1, \ldots, k_n, c_1, \ldots, c_n) \ ,$$

where H is a hash function modeled as a random oracle and the pair (k_i, c_i) is the result of encapsulation under the ith ingredient KEM. A slightly more efficient combiner would be

$$K = H(k_1 \oplus \ldots \oplus k_n, c_1, \ldots, c_n) \ ,$$

where the input session keys are XOR-combined before being fed into the random oracle. On the one hand these constructions are secure, as we prove, but somewhat unfortunate is that they depend so strongly on H behaving like a random oracle: Indeed, if the second construction were to be reinterpreted as

$$K = F(k_1 \oplus \ldots \oplus k_n, c_1 \| \ldots \| c_n) \ ,$$

where now F is a (standard model) PRF, then the construction would be insecure (more precisely, we prove that there exists a PRF such that when it is used in the construction the resulting KEM is insecure). The reason for the last construction not working is that the linearity of the XOR operation allows for conducting related-key attacks on the PRF, and PRFs in general are not immune against such attacks.

Our next proposal towards a KEM combiner that is provably secure in the standard model involves thus a stronger "key-mixing component", i.e., one that is stronger than XOR. Concretely, we study the

[3] Obviously, showing *feasibility* is not a concern for KEM combiners as combiners for PKE have already been studied (see Section 1.2) and the step from PKE to KEMs is minimal.

design that derives the PRF key from a chain of blockcipher invocations, each with individual key, on input the fixed value 0. We obtain

$$K = F(\pi_{k_n} \circ \ldots \circ \pi_{k_1}(0), c_1 \| \ldots \| c_n) \ ,$$

where π_k represents a blockcipher π keyed with key k. Unfortunately, also this construction is generally not secure in the standard model. Yet it is—overall—our favorite construction, for the following reason: In practice, one could instantiate F with a construction based on SHA256 (prepend the key to the message before hashing it, or use NMAC or HMAC), and π with AES. Arguably, SHA256 and AES likely behave well as PRFs and PRPs, respectively; further, in principle, SHA256 is a good candidate for a random oracle and AES is a good candidate for an ideal cipher. Our results on above combiner are as follows: While the combiner is not secure if F and π are a standard model PRF and PRP, respectively, two sufficient conditions for the KEM combiner being secure are that F is a random oracle and π a PRP *or* F is a PRF and π an ideal cipher. That is, who uses the named combiner can afford that one of the two primitives, SHA256 or AES, fails to behave like an ideal primitive. Observe that this is a clear advantage over our first two (random oracle based) combiners for which security is likely gone in the moment hash function H fails to be a random oracle.

The attentive reader might have noticed that, so far, we did not propose a KEM combiner secure in the standard model. This is not because we did not find a corresponding design but, rather, because we decided to save it for a climax at the end of the paper. In fact, by following a new approach we propose two standard-model secure KEM combiners, where the one optimizes the efficiency of the other at the price of requiring an ad hoc proof. Concretely, if below we write $c = c_1 \| \ldots \| c_n$ for the ciphertext vector, our first standard model KEM combiner is

$$K = F(k_1, c) \oplus \ldots \oplus F(k_n, c) \ .$$

While being provably secure if F is a (standard model) PRF, the disadvantage over the earlier designs that are secure in idealized models is that this construction is less efficient, requiring n full passes over the ciphertext vector. Whether this is affordable or not depends on the particular application and the size of the KEM ciphertexts (which might be large for post-quantum KEMs).

The promised optimized variant has n PRF invocations as well, but the amount of PRF-processed data is slightly smaller. Exploiting that the ciphertexts of CCA secure KEMs are non-malleable (in the sense of: If a single ciphertext bit flips the session key to which this ciphertext decapsulates is independent of the original one) we observe that the PRF invocation associated with the ith session key actually does not need to process the ith ciphertext component. More precisely, if for all i we write $c^i = c_1 \| \ldots \| c_{i-1} \| c_{i+1} \| \ldots \| c_n$, then also

$$K = F(k_1, c^1) \oplus \ldots \oplus F(k_n, c^n)$$

is a secure KEM combiner.

SPLIT-KEY PSEUDORANDOM FUNCTIONS Note that in all our constructions the session keys output by the KEM combiner are derived via a function of the form

$$K = W(k_1, \ldots, k_n, c) \ ,$$

where k_i denotes the key output by the encapsulation algorithm of KEM K_i and $c = c_1 \| \ldots \| c_n$. We refer to W as *core function*. We can pinpoint a sufficient condition of the core function such that the respective KEM combiner retains CCA security of any of its ingredient KEMs: Intuitively, *split-key pseudorandomness* captures pseudorandom behavior of W as long as any of the keys $k_1, \ldots, k_n$ is uniformly distributed (and the other keys known to or controlled by the adversary).

All KEM combiners studied in this work that retain CCA security may be found in Figure 1.

1.2 Related work

To the best of our knowledge KEM combiners have not been studied in the literature before. However, closely related, encryption combiners were considered. The idea of encrypting multiple times to strengthen security guarantees dates back to the seminal work of Shannon [20].

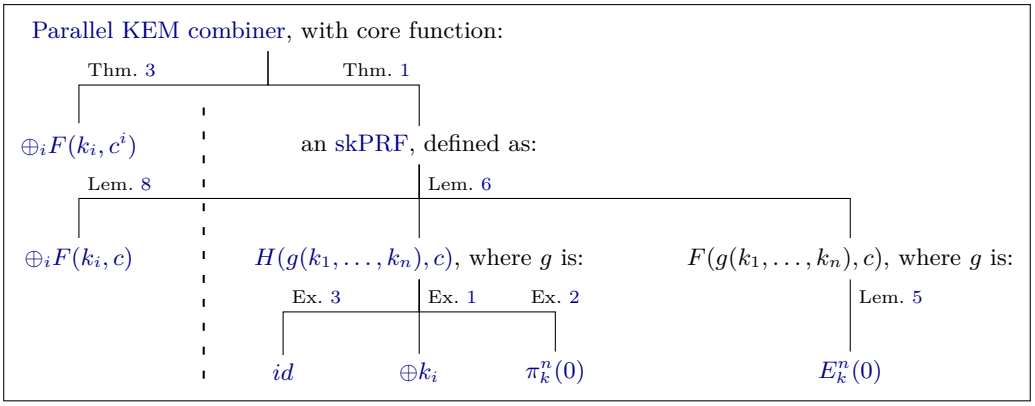

Fig. 1. Overview of our CCA-preserving KEM combiners for n KEMs. F denotes a PRF, H a random oracle, π a keyed permutation, and E an ideal cipher. Moreover, we assume $c = c_1 \,..\, c_n$, $c^i = c_1 \,..\, c_{i-1} c_{i+1} \,..\, c_n$, $k = k_1 \,..\, k_n$ and write $\oplus_i$ for $\oplus_{i=1}^n$. For $x \in \{\pi, E\}$ we write $x_k^n(\cdot)$ to denote $x_{k_n}(\ldots x_{k_1}(\cdot)\ldots)$. Constructions that are secure in the standard model (resp. idealized models) are on the left (resp. right) side of the dashed line.

An immediate and well-studied solution (e.g. [5,18]) to combine various symmetric encryption schemes is to apply them in a cascade fashion where the message is encrypted using the first scheme, the resulting ciphertext then being encrypted with the second scheme, and so on. Even and Goldreich [7] showed that such a chain is at least as secure as its weakest link.

Focusing on combining PKE schemes and improving on prior work (see [22]) Dodis and Katz [6] gave means to employ various PKE schemes that retain CCA security of any 'ingredient' scheme.

More recently, the work of [21] gave another way to combine PKE schemes ensuring that CCA security of any ingredient PKE is passed on to the combined PKE scheme. As a first step, their approach constructs a combiner achieving merely *detectable* CCA (DCCA) security[4] if any ingredient PKE scheme is CCA secure. Secondly, a transformation from DCCA to CCA security (see [16]) is applied to strengthen the PKE combiner.

Conceptually interesting in the context of this paper is the work of [2] where the authors propose an LWE-based key exchange and integrate it into the TLS protocol suite. The goal is to make TLS future proof (against quantum computers). Thereby, they define not only two LWE-based cipher suites, but also two hybrid ones that, conservatively with respect to the security assumptions, combine the LWE techniques with better-studied cyclic group based Diffie–Hellman key exchange.

2 Preliminaries

Notation We use the following operators for assigning values to variables: The symbol '$\leftarrow$' is used to assign to a variable (on the left-hand side) a constant value (on the right-hand side), for example the output of a deterministic algorithm. Similarly, we use '$\leftarrow_{\$}$' to assign to a variable either a uniformly sampled value from a set or the output of a randomized algorithm. If $f\colon A \to B$ is a function or a deterministic algorithm we let $[f] := f(A) \subseteq B$ denote the image of A under f; if $f\colon A \to B$ is a randomized algorithm with randomness space R we correspondingly let $[f] := f(A \times R) \subseteq B$ denote the set of all its possible outputs.

Let T be an associative array (also called array, or table), and b any element. Writing '$T[\cdot] \leftarrow b$' we set $T[a]$ to b for all a. We let $[T]$ denote the space of all elements the form $T[a]$ for some a, excluding the rejection symbol $\perp$. Moreover, $[T[a, \cdot]]$ is the set of all the elements assigned to $T[a, a']$ for any value a'.

Games. Our security definitions are given in terms of *games* written in pseudocode. Within a game a (possibly) stateful *adversary* is explicitly invoked. Depending on the game, the adversary may have oracle access to specific procedures. We write $\mathcal{A}^{\mathcal{O}}$, to indicate that algorithm $\mathcal{A}$ has oracle access to $\mathcal{O}$. Within an oracle, command 'Return X' returns X to the algorithm that called the oracle.

[4] A confidentiality notion that interpolates between CPA and CCA security. Here, an adversary is given a crippled decryption oracle that refuses to decrypt a specified set of efficiently recognizable ciphertexts. See [16].

A game terminates when a 'Stop with X' command is executed; X then serves as the output of the game. We write 'Abort' as an abbreviation for 'Stop with 0'. With '$\mathrm{G} \Rightarrow 1$' we denote the random variable (with randomness space specified by the specifics of the game G) that returns true if the output of the game is 1 and false otherwise.

In proofs that employ game hopping, lines of code that end with a comment of the form '$|\mathrm{G}_i$-G_j' (resp. '$|\mathrm{G}_i, \mathrm{G}_j$', '$|\mathrm{G}_i$') are only executed when a game in G_i–G_j (resp. G_i and G_j, G_i) is run.

Key encapsulation. A key-encapsulation mechanism (KEM) $\mathsf{K} = (\mathsf{K.gen}, \mathsf{K.enc}, \mathsf{K.dec})$ for a finite session-key space $\mathcal{K}$ is a triple of algorithms together with a public-key space $\mathcal{PK}$, a secret-key space $\mathcal{SK}$, and a ciphertext space $\mathcal{C}$. The randomized key-generation algorithm $\mathsf{K.gen}$ returns a public key $pk \in \mathcal{PK}$ and a secret key $sk \in \mathcal{SK}$. The randomized encapsulation algorithm $\mathsf{K.enc}$ takes a public key $pk \in \mathcal{PK}$ and produces a session key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$. Finally, the deterministic decapsulation algorithm $\mathsf{K.dec}$ takes a secret key $sk \in \mathcal{SK}$ and a ciphertext $c \in \mathcal{C}$, and outputs either a session key $k \in \mathcal{K}$ or the special symbol $\perp \notin \mathcal{K}$ to indicate rejection. For correctness we require that for all $(pk, sk) \in [\mathsf{K.gen}]$ and $(k, c) \in [\mathsf{K.enc}(pk)]$ we have $\mathsf{K.dec}(sk, c) = k$.

We now give a security definition for KEMs that formalizes session-key indistinguishability. For a KEM K, associate with any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ its distinguishing advantage $\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}}(\mathcal{A})$ defined as $|\Pr[\mathrm{KIND}^0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{KIND}^1(\mathcal{A}) \Rightarrow 1]|$, where the games are in Figure 2. We sometimes refer to adversaries that refrain from posing queries to the Dec oracle as *passive* or *CPA*, while we refer to adversaries that pose such queries as *active* or *CCA*. Intuitively, a KEM is *CPA secure* (respectively, *CCA secure*) if all practical CPA (resp., CCA) adversaries achieve a negligible distinguishing advantage.

Game $\mathrm{KIND}^b(\mathcal{A})$	**Oracle** $\mathrm{Dec}(c)$
00 $C^* \leftarrow \emptyset$	08 If $c \in C^*$: Abort
01 $(pk, sk) \leftarrow_{\$} \mathsf{K.gen}$	09 $k \leftarrow \mathsf{K.dec}(sk, c)$
02 $st \leftarrow_{\$} \mathcal{A}_1^{\mathrm{Dec}}(pk)$	10 Return k
03 $(k^*, c^*) \leftarrow_{\$} \mathsf{K.enc}(pk)$	
04 $k^0 \leftarrow k^*$; $k^1 \leftarrow_{\$} \mathcal{K}$	
05 $C^* \leftarrow C^* \cup \{c^*\}$	
06 $b' \leftarrow_{\$} \mathcal{A}_2^{\mathrm{Dec}}(st, c^*, k^b)$	
07 Stop with b'	

Fig. 2. Security experiments KIND^b, $b \in \{0, 1\}$, modeling the session-key indistinguishability of KEM K. With st we denote internal state information of the adversary.

Pseudorandom functions. Fix a finite key space $\mathcal{K}$, an input space $\mathcal{X}$, a finite output space $\mathcal{Y}$, and a function $F\colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. Towards defining what it means for F to behave pseudorandomly, associate with any adversary $\mathcal{A}$ its advantage $\mathrm{Adv}^{\mathrm{pr}}_F(\mathcal{A}) := |\Pr[\mathrm{PR}^0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{PR}^1(\mathcal{A}) \Rightarrow 1]|$, where the games are in Figure 3. Intuitively, F is a *pseudorandom function* (PRF) if all practical adversaries achieve a negligible advantage.

Game $\mathrm{PR}^b(\mathcal{A})$	**Oracle** $\mathrm{Eval}(x)$
00 $X \leftarrow \emptyset$	04 If $x \in X$: Abort
01 $k \leftarrow_{\$} \mathcal{K}$	05 $X \leftarrow X \cup \{x\}$
02 $b' \leftarrow_{\$} \mathcal{A}^{\mathrm{Eval}}$	06 $y \leftarrow F(k, x)$
03 Stop with b'	07 $y^0 \leftarrow y$; $y^1 \leftarrow_{\$} \mathcal{Y}$
	08 Return y^b

Fig. 3. Security experiments PR^b, $b \in \{0, 1\}$, modeling the pseudorandomness of function F. Line 04 and 05 implement the requirement that Eval not be queried on the same input twice.

Pseudorandom permutations. Intuitively, a pseudorandom permutation (PRP) is a bijective PRF. More precisely, if $\mathcal{K}$ is a finite key space and $\mathcal{D}$ a finite domain, then function $\pi\colon \mathcal{K}\times\mathcal{D}\to\mathcal{D}$ is a PRP if for all $k\in\mathcal{K}$ the partial function $\pi(k,\cdot)\colon \mathcal{D}\to\mathcal{D}$ is bijective and if $\pi(k,\cdot)$ behaves like a random permutation $\mathcal{D}\to\mathcal{D}$ once $k\in\mathcal{K}$ is assigned uniformly at random. A formalization of this concept would be in the spirit of Figure 3. In practice, PRPs are often implemented with blockciphers.

Random oracle model, ideal cipher model. We consider a cryptographic scheme defined with respect to a hash function $H\colon \mathcal{X}\to\mathcal{Y}$ in the *random oracle model* for H by replacing the scheme's internal invocations of H by calls to an oracle H that implements a uniform mapping $\mathcal{X}\to\mathcal{Y}$. In security analyses of the scheme, also the adversary gets access to this oracle. Similarly, a scheme defined with respect to a keyed permutation $\pi\colon \mathcal{K}\times\mathcal{D}\to\mathcal{D}$ is considered in the *ideal cipher model* for π if all computations of $\pi(\cdot,\cdot)$ in the scheme algorithms are replaced by calls to an oracle $\mathrm{E}(\cdot,\cdot)$ that implements a uniform mapping $\mathcal{K}\times\mathcal{D}\to\mathcal{D}$ such that $\mathrm{E}(k,\cdot)$ is a bijection for all k, and all computations of $\pi^{-1}(\cdot,\cdot)$ are replaced by calls to an oracle $\mathrm{D}(\cdot,\cdot)$ that implements a uniform mapping $\mathcal{K}\times\mathcal{D}\to\mathcal{D}$ such that $\mathrm{D}(k,\cdot)$ is a bijection for all k, and the partial oracles $\mathrm{E}(k,\cdot)$ and $\mathrm{D}(k,\cdot)$ are inverses of each other (again for all k). In corresponding security analyses the adversary gets access to both E and D. We write E (resp. D) to denote π (resp. π^{-1}) every time that we want to remark that π will be considered in the ideal cipher model.

3 KEM combiners

A KEM combiner is a template that specifies how a set of existing KEMs can be joined together, possibly with the aid of further cryptographic primitives, to obtain a new KEM. In this paper we are exclusively interested in combiners that are security preserving: The resulting KEM shall be at least as secure as any of its ingredient KEMs (assuming all further primitives introduced by the combiner are secure). While for public-key encryption a serial combination process is possible and plausible (encrypt the message with the first PKE scheme, the resulting ciphertext with the second PKE scheme, and so on, for KEMs a parallel approach, where the ciphertext consists of a set of independently generated ciphertext components (one component per ingredient KEM), seems more natural. We formalize a general family of parallel combiners that are parameterized by a *core function* that derives a combined session key from a vector of session keys and a vector of ciphertexts.

Parallel KEM combiner. Let $\mathsf{K}_1,\ldots,\mathsf{K}_n$ be (ingredient) key-encapsulation mechanism such that each KEM $\mathsf{K}_i=(\mathsf{K.gen}_i,\mathsf{K.enc}_i,\mathsf{K.dec}_i)$ has session-key space $\mathcal{K}_i$, public-key space $\mathcal{PK}_i$, secret-key space $\mathcal{SK}_i$, and ciphertext space $\mathcal{C}_i$. Let $\mathcal{K}_*=\mathcal{K}_1\times\ldots\times\mathcal{K}_n$ and $\mathcal{PK}=\mathcal{PK}_1\times\ldots\times\mathcal{PK}_n$ and $\mathcal{SK}=\mathcal{SK}_1\times\ldots\times\mathcal{SK}_n$ and $\mathcal{C}=\mathcal{C}_1\times\ldots\times\mathcal{C}_n$. Let further $\mathcal{K}$ be an auxiliary finite session-key space. For any *core function* $W\colon\mathcal{K}_*\times\mathcal{C}\to\mathcal{K}$, the *parallel combination* $\mathsf{K}:=\mathsf{K}_1\|\ldots\|\mathsf{K}_n$ with respect to W is a KEM with session-key space $\mathcal{K}$ that consists of the algorithms $\mathsf{K.gen},\mathsf{K.enc},\mathsf{K.dec}$ specified in Figure 4. The combined KEM K has public-key space $\mathcal{PK}$, secret-key space $\mathcal{SK}$, and ciphertext space $\mathcal{C}$. A quick inspection of the algorithms shows that if all ingredient KEMs K_i are correct, then so is K.

Algo $\mathsf{K.gen}$	**Algo** $\mathsf{K.enc}(pk)$	**Algo** $\mathsf{K.dec}(sk,c)$
00 For $i\leftarrow 1$ to n:	05 $(pk_1,\ldots,pk_n)\leftarrow pk$	11 $(sk_1,\ldots,sk_n)\leftarrow sk$
01 $\quad(pk_i,sk_i)\leftarrow_{\$}\mathsf{K.gen}_i$	06 For $i\leftarrow 1$ to n:	12 $c_1\,..\,c_n\leftarrow c$
02 $pk\leftarrow(pk_1,\ldots,pk_n)$	07 $\quad(k_i,c_i)\leftarrow_{\$}\mathsf{K.enc}_i(pk_i)$	13 For $i\leftarrow 1$ to n:
03 $sk\leftarrow(sk_1,\ldots,sk_n)$	08 $c\leftarrow c_1\,..\,c_n$	14 $\quad k_i\leftarrow\mathsf{K.dec}_i(sk_i,c_i)$
04 Return (pk,sk)	09 $k\leftarrow W(k_1,\ldots,k_n,c)$	15 $\quad$If $k_i=\perp$: Return $\perp$
	10 Return (k,c)	16 $k\leftarrow W(k_1,\ldots,k_n,c)$
		17 Return k

Fig. 4. Parallel KEM combiner, defined with respect to some core function W.

The security properties of the parallel combiner depend crucially on the choice of the core function W. For instance, if W maps all inputs to one fixed session key $\bar{k}\in\mathcal{K}$, the obtained KEM does not inherit any security from the ingredient KEMs. We are thus left with finding good core functions W.

3.1 The XOR combiner

Assume ingredient KEMs that share a common binary-string session-key space: $\mathcal{K}_1 = \ldots = \mathcal{K}_n = \{0,1\}^k$ for some k. Consider the XOR core function that, disregarding its ciphertext inputs, outputs the binary sum of the key inputs. Formally, after letting $\mathcal{K} = \{0,1\}^k$ this means $\mathcal{K}_* = \mathcal{K}^n$ and

$$W\colon\ \mathcal{K}_* \times \mathcal{C} \to \mathcal{K}\ ;\quad (k_1, \ldots, k_n, c_1 \,..\, c_n) \mapsto k_1 \oplus \ldots \oplus k_n \quad .$$

On W we prove two statements: If the overall goal is to obtain a CPA-secure KEM, then W is useful, in the sense that the parallel combination of KEMs with respect to W is CPA secure if at least one of the ingredient KEMs is. However, if the overall goal is CCA security, then one weak ingredient KEM is sufficient to break any parallel combination with respect to W.

Lemma 1 (XOR combiner retains CPA security). *Let $\mathsf{K}_1, \ldots, \mathsf{K}_n$ be KEMs and let W be the XOR core function. Consider the parallel combination $\mathsf{K} = \mathsf{K}_1 \,\|\, \ldots \,\|\, \mathsf{K}_n$ with respect to W. If at least one K_i is CPA secure, then also K is CPA secure. Formally, for all indices $i \in [1\,..\,n]$ and every adversary $\mathcal{A}$ that poses no queries to the decapsulation oracle there exists an adversary $\mathcal{B}$ such that*

$$\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}}(\mathcal{A}) = \mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}_i}(\mathcal{B})\ ,$$

where also $\mathcal{B}$ poses no decapsulation query and its running time is about that of $\mathcal{A}$.

Proof. From any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against K we construct an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against K_i as follows. Algorithm $\mathcal{B}_1$, on input $pk_i \in \mathcal{PK}_i$, first generates the $n-1$ public keys $pk_1, \ldots, pk_{i-1}$, $pk_{i+1}, \ldots, pk_n$ as per $(pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$. Then it sets $pk \leftarrow (pk_1, \ldots, pk_n)$, invokes $st \leftarrow_{\$} \mathcal{A}_1(pk)$, and outputs $st' \leftarrow (st, pk_1, \ldots, pk_{i-1}, pk_{i+1}, \ldots, pk_n)$. Algorithm $\mathcal{B}_2$, on input (st', c_i^*, k_i^*), first invokes $(k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j(pk_j)$ for all $j \neq i$, and then sets $c^* \leftarrow c_1^* \,..\, c_i^* \,..\, c_n^*$ and $k^* \leftarrow k_1^* \oplus \ldots \oplus k_i^* \oplus \ldots \oplus k_n^*$. Finally it invokes $b' \leftarrow_{\$} \mathcal{A}_2(st, c^*, k^*)$ and outputs b'. It is easy to see that the advantages of $\mathcal{A}$ and $\mathcal{B}$ coincide. □

Remark. Consider a CCA secure KEM (for instance from the many submissions to NIST's recent Post-Quantum initiative [19]) that is constructed by, first, taking a CPA secure KEM and then applying a Fujisaki–Okamoto-like transformation [12,15,17] to it in order to obtain a CCA secure KEM.

To combine multiple KEMs that follow the above design principle, Lemma 1 already provides a highly efficient solution that retains CCA security: To this end, one would strip away the FO-like transformation of the KEMs to be combined and apply the XOR-combiner to the various CPA secure KEMs. Eventually one would apply an FO-like transformation to the XOR-combiner.

However, besides results shedding doubts on the instantiability of FO in the presence of indistinguishability obfuscation [3], we pursue generic KEM combiners that retain CCA security independently of how the ingredient KEMs achieve their security.

While it is rather obvious that the XOR-combiner is incapable of retaining CCA security of an ingredient KEM, we formally state and prove it next.

Lemma 2 (XOR combiner does not retain CCA security). *In general, the result of parallelly combining a CCA-secure KEM with other KEMs using the XOR core function is not CCA secure.*

Formally, if $n \in \mathbb{N}$ and W is the XOR core function, then for all $1 \leq i \leq n$ there exists a KEM K_i such that for any set of $n-1$ KEMs $\mathsf{K}_1, \ldots, \mathsf{K}_{i-1}, \mathsf{K}_{i+1}, \ldots, \mathsf{K}_n$ (e.g., all of them CCA secure) there exists an efficient adversary $\mathcal{A}$ that poses a single decapsulation query and achieves an advantage of $\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}}(\mathcal{A}) = 1 - 1/|\mathcal{K}|$, where $\mathsf{K} = \mathsf{K}_1 \,\|\, \ldots \,\|\, \mathsf{K}_n$ is the parallel combination of $\mathsf{K}_1, \ldots, \mathsf{K}_n$ with respect to W.

Proof. We construct KEM K_i such that public and secret keys play no role, it has only two ciphertexts, and it establishes always the same session key: Fix any $\bar{k} \in \mathcal{K}$, let $\mathcal{C}_i = \{0,1\}$, and let $\mathsf{K.enc}_i$ and $\mathsf{K.dec}_i$ always output $(\bar{k}, 0) \in \mathcal{K} \times \mathcal{C}_i$ and $\bar{k} \in \mathcal{K}$, respectively. Define adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1$ does nothing and $\mathcal{A}_2$, on input of c^* and k^*, parses c^* as $c_1^* \,..\, c_i^* \,..\, c_n^*$ (where $c_i^* = 0$), poses a decapsulation query $k^{**} \leftarrow \mathrm{Dec}(c^{**})$ on ciphertext $c^{**} = c_1^* \,..\, c_{i-1}^* 1 c_{i+1}^* \,..\, c_n^*$, and outputs 1 iff $k^* = k^{**}$. It is easy to see that $\mathcal{A}$ achieves the claimed advantage. □

3.2 The XOR-then-PRF combiner

We saw that the KEM combiner that uses the core function that simply outputs the XOR sum of the session keys fails miserably to provide security against active adversaries. The main reason is that it completely ignores the ciphertext inputs, so that the latter can be altered by an adversary without affecting the corresponding session key. As an attempt to remedy this, we next consider a core function that, using a PRF, mixes all ciphertext bits into the session key that it outputs. The PRF is keyed with the XOR sum of the input session keys and shall serve as an integrity protection on the ciphertexts.

Formally, under the same constraints on $\mathcal{K}, \mathcal{K}_1, \ldots, \mathcal{K}_n, \mathcal{K}_*$ as in Section 3.1, and assuming a (pseudorandom) function $F\colon \mathcal{K} \times \mathcal{C} \to \mathcal{K}$, the XOR-then-PRF core function W_F is defined as per

$$W_F\colon\ \mathcal{K}_* \times \mathcal{C} \to \mathcal{K}\ ; \quad (k_1, \ldots, k_n, c_1 \,..\, c_n) \mapsto F(k_1 \oplus \ldots \oplus k_n, c_1 \,..\, c_n) \quad .$$

Of course, to leverage on the pseudorandomness of the function F its key has to be uniform. The hope, based on the intuition that at least one of the ingredient KEMs is assumed secure and thus the corresponding session key uniform, is that the XOR sum of all session keys works fine as a PRF key. Unfortunately, as we prove next, this is not the case in general. The key insight is that the pseudorandomness definition does not capture robustness against related-key attacks: We present a KEM/PRF combination where manipulating KEM ciphertexts allows to exploit a particular structure of the PRF.[5]

Lemma 3 (XOR-then-PRF combiner does not retain CCA security). *There exist KEM/PRF configurations such that if the KEM is parallelly combined with other KEMs using the XOR-then-PRF core function, then the resulting KEM is weak against active attacks. More precisely, for all $n \in \mathbb{N}$ and $i \in [1\,..\,n]$ there exists a KEM K_i and a (pseudorandom) function F such that for any set of $n-1$ (arbitrarily secure) KEMs $\mathsf{K}_1, \ldots, \mathsf{K}_{i-1}, \mathsf{K}_{i+1}, \ldots, \mathsf{K}_n$ there exists an efficient adversary $\mathcal{A}$ that poses a single decapsulation query and achieves advantage $\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}}(\mathcal{A}) = 1 - 1/|\mathcal{K}|$, where $\mathsf{K} = \mathsf{K}_1 \,\|\, \ldots \,\|\, \mathsf{K}_n$ is the parallel combination of $\mathsf{K}_1, \ldots, \mathsf{K}_n$ with respect to the XOR-then-PRF core function W_F. Function F is constructed from a function F' such that if F' is pseudorandom then so is F.*

Proof. In the following we write $\mathcal{K} = \{0,1\} \times \mathcal{K}'$ (where $\mathcal{K}' = \{0,1\}^{k-1}$). We construct K_i such that public and secret keys play no role, there are only two ciphertexts, and the two ciphertexts decapsulate to different session keys: Fix any $\bar{k} \in \mathcal{K}'$, let $\mathcal{C}_i = \{0,1\}$, let $\mathsf{K}.\mathsf{enc}_i$ always output $((0,\bar{k}),0) \in \mathcal{K} \times \mathcal{C}_i$, and let $\mathsf{K}.\mathsf{dec}_i$, on input ciphertext $B \in \mathcal{C}_i$, output session key $(B,\bar{k}) \in \mathcal{K}$.

We next construct a specific function F and argue that it is pseudorandom. Consider the involution $\pi\colon \mathcal{C} \to \mathcal{C}$ that flips the bit value of the ith ciphertext component, i.e.,

$$\pi(c_1 \,..\, c_{i-1}\, B\, c_{i+1} \,..\, c_n) = c_1 \,..\, c_{i-1}\, (1-B)\, c_{i+1} \,..\, c_n \quad ,$$

and let $F'\colon \mathcal{K}' \times \mathcal{C} \to \mathcal{K}$ be a (pseudorandom) function. Construct $F\colon \mathcal{K} \times \mathcal{C} \to \mathcal{K}$ from π and F' as per

$$F((D,k'),c) = \begin{cases} F'(k',c) & \text{if } D = 0 \\ F'(k',\pi(c)) & \text{if } D = 1 \end{cases} \quad . \tag{1}$$

It is not difficult to see that if F' is pseudorandom then so is F. For completeness, we give a formal statement and proof immediately after this proof.

Consider now the following adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$: Let algorithm $\mathcal{A}_1$ do nothing, and let algorithm $\mathcal{A}_2$, on input of c^* and k^*, parse the ciphertext as $c^* = c_1^* \,..\, c_i^* \,..\, c_n^*$ (where $c_i^* = 0$), pose a decapsulation query $k^{**} \leftarrow \mathrm{Dec}(c^{**})$ on ciphertext $c^{**} = c_1^* \,..\, c_{i-1}^* 1 c_{i+1}^* \,..\, c_n^*$, and output 1 iff $k^* = k^{**}$.

Let us analyze the advantage of $\mathcal{A}$. For all $1 \le j \le n$, let $(d_j, k'_j) \in \mathcal{K}$ be the session keys to which ciphertext components c_j^* decapsulate. That is, the session key k to which c^* decapsulates can be computed as $k = F((d_1,k'_1) \oplus \ldots \oplus (d_n,k'_n), c^*)$, by specification of W_F. By setting $D = d_1 \oplus \ldots \oplus d_n$ and expanding F into F' and π we obtain

$$k = F'(k'_1 \oplus \ldots \oplus k'_n, c_1^* \,..\, c_{i-1}^* D c_{i+1}^* \,..\, c_n^*) \quad .$$

[5] Note that in Lemma 6 we prove that if F behaves like a random oracle and is thus free of related-key conditions, the XOR-then-PRF core function actually does yield a secure CCA combiner.

Consider next the key k^{**} that is returned by the Dec oracle. Internally, the oracle recovers the same keys $(d_1, k'_1), \ldots, (d_n, k'_n)$ as above, with exception of d_i which is inverted. Let $D^{**} = d_1 \oplus \ldots \oplus d_n$ be the corresponding (updated) sum. We obtain

$$k^{**} = F'(k'_1 \oplus \ldots \oplus k'_n, c^*_1 .. c^*_{i-1}(1 - D^{**})c^*_{i+1} .. c^*_n) \ .$$

Thus, as D^{**} is the inverse of D, we have $k = k^{**}$ and adversary $\mathcal{A}$ achieves the claimed advantage. □

We now give the formal statement that, if F' is a PRF then the same holds for F as defined in (1).

Lemma 4. *Let $\mathcal{K}', \mathcal{X}, \mathcal{Y}$ be sets such that $\mathcal{K}', \mathcal{Y}$ are finite. Let $F' \colon \mathcal{K}' \times \mathcal{X} \to \mathcal{Y}$ be a function, and let $\pi \colon \mathcal{X} \to \mathcal{X}$ be any (efficient) bijection.*[6] *Let $\mathcal{K} = \{0,1\} \times \mathcal{K}'$ and define function $F \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ such that*

$$F((D, k'), x) = \begin{cases} F'(k', x) & \text{if } D = 0 \\ F'(k', \pi(x)) & \text{if } D = 1 \end{cases} \ . \tag{2}$$

Then if F' is a PRF, the same holds for F. More precisely, for every adversary $\mathcal{A}$ there is an adversary $\mathcal{B}$ such that

$$\mathrm{Adv}^{\mathrm{pr}}_{F}(\mathcal{A}) = \mathrm{Adv}^{\mathrm{pr}}_{F'}(\mathcal{B}) \ ,$$

the running times of $\mathcal{A}$ and $\mathcal{B}$ are roughly the same, and if $\mathcal{A}$ queries its evaluation oracle q_e times then $\mathcal{B}$ queries its own evaluation oracle q_e times.

Proof. Let $\mathcal{A}$ be an adversary against the pseudorandomness of F. We build an adversary $\mathcal{B}$ against the pseudorandomness of F' as follows. $\mathcal{B}$ generates a bit D and runs $\mathcal{A}$. For every Eval query of $\mathcal{A}$ on input x, adversary $\mathcal{B}$ queries its own evaluation oracle on input x if $D = 0$, or $\pi(x)$ if $D = 1$. The output of this query is returned to $\mathcal{A}$. At the end of $\mathcal{A}$'s execution its output is returned by $\mathcal{B}$.

We argue that $\mathcal{B}$ provides a correct simulation of the pseudorandomness games to $\mathcal{A}$. First we notice that if the input values to Eval by $\mathcal{A}$ are unique, so are the input values to Eval by $\mathcal{B}$, since π is a bijection and D is constant during each run of the simulation. Conversely, any input repetition by $\mathcal{A}$ leads to an input repetition by $\mathcal{B}$, thus aborting the pseudorandomness game. If $\mathcal{B}$ is playing against the real game PR^0 for F' then it correctly computes the function F for $\mathcal{A}$ and the distribution of the output to $\mathcal{A}$ is the same as that in game PR^0 for F. Otherwise $\mathcal{B}$ receives uniform independent elements from its oracle Eval, and hence correctly simulates the game PR^1 for F to $\mathcal{A}$. This proves our statement. □

3.3 KEM combiners from split-key PRFs

The two core functions for the parallel KEM combiner that we studied so far did not achieve security against active attacks. We next identify a sufficient condition that guarantees satisfactory results: If the core function is *split-key pseudorandom*, and at least one of the ingredient KEMs of the parallel combiner from Figure 4 is CCA secure, then the resulting KEM is CCA secure as well.

Split-key pseudorandom functions. We say a symmetric key primitive (syntactically) uses split keys if its key space $\mathcal{K}$ is the Cartesian product of a finite number of (sub)key spaces $\mathcal{K}_1, \ldots, \mathcal{K}_n$. In the following we study the corresponding notion of split-key pseudorandom function. In principle, such functions are just a special variant of PRFs, so that the security notion of pseudorandomness (see Figure 3) remains meaningful. However, we introduce *split-key pseudorandomness* as a dedicated, refined property. In brief, a split-key function has this property if it behaves like a random function if at least one component of its key is picked uniformly at random (while the other components may be known or even chosen by the adversary).

For formalizing this, fix finite key spaces $\mathcal{K}_1, \ldots, \mathcal{K}_n$, an input space $\mathcal{X}$, and a finite output space $\mathcal{Y}$. Let $\mathcal{K} = \mathcal{K}_1 \times \ldots \times \mathcal{K}_n$ and consider a function $F \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$. For each index $i \in [1 .. n]$, associate with an adversary $\mathcal{A}$ its advantage $\mathrm{Adv}^{\mathrm{pr}}_{F,i}(\mathcal{A}) := |\Pr[\mathrm{PR}^0_i(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{PR}^1_i(\mathcal{A}) \Rightarrow 1]|$, where the game is given in Figure 5. Observe that, for any index i, in the game PR^b_i, $b \in \{0,1\}$, the ith key component of F is assigned at random (in line 01), while the adversary contributes the remaining $n-1$ components

Games $\mathrm{PR}_i^b(\mathcal{A})$	**Oracle** $\mathrm{Eval}(k', x)$
00 $X \leftarrow \emptyset$	04 If $x \in X$: Abort
01 $k_i \leftarrow_\$ \mathcal{K}_i$	05 $X \leftarrow X \cup \{x\}$
02 $b' \leftarrow_\$ \mathcal{A}^{\mathrm{Eval}}$	06 $k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n \leftarrow k'$
03 Stop with b'	07 $y \leftarrow F(k_1 \,..\, k_i \,..\, k_n, x)$
	08 $y^0 \leftarrow y$; $y^1 \leftarrow_\$ \mathcal{Y}$
	09 Return y^b

Fig. 5. Security game PR_i^b, $b \in \{0,1\}$, $1 \le i \le n$, modeling the split-key pseudorandomness of function F. Lines 04 and 05 implement the requirement that Eval not be queried on the same input twice.

on a per-query basis (see line 06). We say that F is a *split-key pseudorandom function* (skPRF) if the advantages $\mathrm{Adv}^{\mathrm{pr}}_{F,i}$ for all key indices are negligible for all practical adversaries.

With lines 04 and 05 we require that the oracle Eval be executed at most once on an input value x, independently on the input value k'. One could imagine a relaxed version of this requirement, where Eval accepts any non-repeating input pair (k', x), thus permitting repeated values of x in distinct queries to Eval. Most of the following proofs are however not straightforward to be adapted to the relaxed definition, and in many case this would directly lead to an insecure construction. Notice, however, that our current definition of split-key pseudorandomness for a function F still suffices to prove that F is a standard PRF.

Theorem 1. *If the core function used in the parallel composition is split-key pseudorandom, the parallel combiner yields a CCA-secure KEM if at least one of the ingredient KEMs is CCA secure.*

More precisely, for all $n, \mathsf{K}_1, \ldots, \mathsf{K}_n$, if $\mathsf{K} = \mathsf{K}_1 \,\|\, \ldots \,\|\, \mathsf{K}_n$ with core function W then for all indices i and all adversaries $\mathcal{A}$ against the key indistinguishability of K there exist adversaries $\mathcal{B}$ against the key indistinguishability of K_i and $\mathcal{C}$ against the split-key pseudorandomness of W such that

$$\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}}(\mathcal{A}) \le 2 \cdot \left(\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}_i}(\mathcal{B}) + \mathrm{Adv}^{\mathrm{pr}}_{W,i}(\mathcal{C}) \right) \quad .$$

Moreover, if adversary $\mathcal{A}$ calls at most q_d times the oracle Dec*, then $\mathcal{B}$ calls at most q_d times the oracle* Dec*, and $\mathcal{C}$ calls at most $q_d + 1$ times the oracle* Eval*. The running times of $\mathcal{B}$ and $\mathcal{C}$ are roughly the same as that of $\mathcal{A}$.*

Proof sketch. The proof constitutes of a sequence of games interpolating between games G_0 and G_4. Noting that the KEMs we consider are perfectly correct, those two games correspond respectively to games KIND^0 and KIND^1 for the KEM $\mathsf{K} = \mathsf{K}_1 \,\|\, \ldots \,\|\, \mathsf{K}_n$. Code detailing the games involved is in Figure 7 and the main differences between consecutive games are explained in Figure 6. In a nutshell, we proceed as follows: In game G_1 we replace the key k_i^* output by $(k_i^*, c_i^*) \leftarrow_\$ \mathsf{K}.\mathsf{enc}_i(pk_i)$ by a uniform key. As K_i is CCA secure this modification is oblivious to $\mathcal{A}$. As a second step, we replace the real challenge session key k^* as obtained via $k^* \leftarrow W(k_1^*, \ldots, k_n^*, c_1^* \,..\, c_n^*)$ with a uniform session key in game G_2. Since the core function W is split-key pseudorandom and k_i^* is uniform, this step is oblivious to $\mathcal{A}$ as well. However—for technical reasons within the reduction—replacing the challenge session key will introduce an artifact to the decapsulation procedure: queries of the form $\mathrm{Dec}(\ldots, c_i^*, \ldots)$ will not be processed using W but answered with uniform session keys. In the transition to game G_3 we employ the split-key pseudorandomness of W again to remove the artifact from the decapsulation oracle. Eventually, in game G_4 we undo our first modification and replace the currently uniform key k_i^* with the actual key obtained by running $\mathsf{K}.\mathsf{enc}_i(pk_i)$. Still, the challenge session key k^* remains uniform. Again, the CCA security of K_i ensures that $\mathcal{A}$ will not detect the modification.

We proceed with a detailed proof.

Proof (Theorem 1). Let $\mathcal{A}$ denote an adversary attacking the CCA security of the KEM K that issues at most q_d queries to the decapsulation oracle. We proceed with detailed descriptions of the games (see Figure 7) used in our proof.

[6] No cryptographic property is required of π, just that it can be efficiently computed. An easy example is the flip-the-first-bit function.

Game	k_i^*	k^*	$\mathrm{Dec}(\ldots, c_i^*, \ldots)$	Δ
G_0 ($\equiv \mathrm{KIND}^0$)	real	real	real	
G_1	random	real	real	$\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}_i}$
G_2	random	random	random	$\mathrm{Adv}^{\mathrm{pr}}_{W,i}$
G_3	random	random	real	$\mathrm{Adv}^{\mathrm{pr}}_{W,i}$
G_4 ($\equiv \mathrm{KIND}^1$)	real	random	real	$\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}_i}$

Fig. 6. Outline of games used in the proof of Theorem 1. We have $(k_i^*, c_i^*) \leftarrow_{\$} \mathsf{K.enc}_i(pk_i)$. Furthermore, $k^* \leftarrow W(k_1^*, \ldots, k_n^*, c_1^* \,..\, c_n^*)$ denotes the challenge session key given to $\mathcal{A}_2$ along with $c_1^* \,..\, c_n^*$.

Games G_0 **to** G_4		**Oracle** $\mathrm{Dec}(c)$	
00 $C^*, C_i^* \leftarrow \emptyset$; $L[\cdot] \leftarrow \bot$		14 If $c \in C^*$: Abort	
01 For $j \leftarrow 1$ to n:		15 If $L[c] \neq \bot$: Return $L[c]$	
02 $\quad(pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$		16 $c_1 \,..\, c_n \leftarrow c$	
03 $pk \leftarrow (pk_1, \ldots, pk_n)$		17 For $j \in [1\,..\,n] \setminus \{i\}$:	
04 $st \leftarrow_{\$} \mathcal{A}_1^{\mathrm{Dec}}(pk)$		18 $\quad k_j \leftarrow \mathsf{K.dec}_j(sk_j, c_j)$	
05 For $j \leftarrow 1$ to n:		19 $\quad$ If $k_j = \bot$: Return $\bot$	
06 $\quad(k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j(pk_j)$		20 If $c_i \in C_i^*$:	
07 $k_i^* \leftarrow_{\$} \mathcal{K}_i$	$\mid \mathrm{G}_1$-G_3	21 $\quad k_i \leftarrow k_i^*$	
08 $c^* \leftarrow c_1^* \,..\, c_n^*$		22 Else:	
09 $k^* \leftarrow W(k_1^*, \ldots, k_n^*, c^*)$		23 $\quad k_i \leftarrow \mathsf{K.dec}_i(sk_i, c_i)$	
10 $k^* \leftarrow_{\$} \mathcal{K}$	$\mid \mathrm{G}_2$-G_4	24 $\quad$ If $k_i = \bot$: Return $\bot$	
11 $C^* \leftarrow C^* \cup \{c^*\}$; $C_i^* \leftarrow C_i^* \cup \{c_i^*\}$		25 $L[c] \leftarrow W(k_1, \ldots, k_n, c)$	
12 $b' \leftarrow_{\$} \mathcal{A}_2^{\mathrm{Dec}}(st, c^*, k^*)$		26 If $c_i \in C_i^*$: $L[c] \leftarrow_{\$} \mathcal{K}$	$\mid \mathrm{G}_2$
13 Stop with b'		27 Return $L[c]$	

Fig. 7. Games G_0–G_4 as used in the proof of Theorem 1. Note that i is implicitly a parameter of all games above.

Game G_0 The KIND^0 game instantiated with the KEM K as given in Figure 4. Beyond that we made merely syntactical changes: In line 00 a set C_i^* and an array L are initialized as empty. In line 15 we check if the adversary has already queried the oracle for the same input and we return the same output. Lines 20 and 21 are added such that, instead of using sk_i to decapsulate c_i^*, the key k_i^* is used. Note that if line 21 is executed then key k_i^* is already defined, since $C_i^* \neq \emptyset$.

Claim 1. $\Pr[\mathrm{KIND}^0 \Rightarrow 1] = \Pr[\mathrm{G}_0 \Rightarrow 1]$.

This follows immediately from the correctness of K_i and the fact that the decapsulation algorithm is deterministic.

Game G_1 Line 07 is added to replace the key k_i^* with a uniform key from $\mathcal{K}_i$.

Claim 2. There is an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against session-key indistinguishability of K_i (see Figure 8) that issues at most q_d decapsulation queries such that

$$|\Pr[\mathrm{G}_0 \Rightarrow 1] - \Pr[\mathrm{G}_1 \Rightarrow 1]| \leq \mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}_i}(\mathcal{B}) \ ,$$

and the running time of $\mathcal{B}$ is roughly the running time of $\mathcal{A}$.

Proof. We construct $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ as given in Figure 8: Adversary $\mathcal{B}_1$ gets pk_i as input, and runs $(pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$ for all $j \in [1\,..\,n] \setminus \{i\}$ to instantiate the other KEMs (see lines 01–03). To answer the decapsulation queries of $\mathcal{A}_1$, $\mathcal{B}_1$ decapsulates all c_i for $j \neq i$ using sk_j (lines 16–18) and queries its own decapsulation oracle to decapsulate c_i (lines 21–23).

Adversary $\mathcal{B}_2$, run on the challenge (c_i^*, k_i^*), executes $(k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j$ for $j \neq i$ on its own (lines 07, 08). Then it computes the challenge session key $k^* \leftarrow W(k_1^*, \ldots, k_n^*, c_1^*, \ldots, c_n^*)$ (line 10) and runs $\mathcal{A}_2$ on $(c_1^*, \ldots, c_n^*, k^*)$ (line 12). Decryption queries are answered as in phase one unless $\mathcal{B}_2$ has to decapsulate c_i^* where it uses k_i^* instead (lines 19, 20). At the end $\mathcal{B}_2$ relays $\mathcal{A}_2$'s output and halts (line 13).

Adversary $\mathcal{B}_1^{\mathrm{Dec}}(pk_i)$
00 $C^*, C_i^* \leftarrow \emptyset$
01 For $j \in [1\,..\,n] \setminus \{i\}$:
02 $\quad (pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$
03 $pk \leftarrow (pk_1, \ldots, pk_n)$
04 $st \leftarrow_{\$} \mathcal{A}_1^{\mathrm{Dec}}(pk)$
05 $st' \leftarrow (st, pk, sk_1, \ldots, sk_{i-1}, sk_{i+1}, \ldots, sk_n)$
06 Return st'

Adversary $\mathcal{B}_2^{\mathrm{Dec}}(st', c_i^*, k_i^*)$
07 For $j \in [1\,..\,n] \setminus \{i\}$:
08 $\quad (k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j(pk_j)$
09 $c^* \leftarrow c_1^*\,..\,c_n^*$
10 $k^* \leftarrow W(k_1^*, \ldots, k_n^*, c^*)$
11 $C^* \leftarrow C^* \cup \{c^*\}$; $C_i^* \leftarrow C_i^* \cup \{c_i^*\}$
12 $b' \leftarrow_{\$} \mathcal{A}_2^{\mathrm{Dec}}(st, c^*, k^*)$
13 Stop with b'

If $\mathcal{A}$ calls $\mathrm{Dec}(c)$:
14 If $c \in C^*$: Abort
15 $c_1\,..\,c_n \leftarrow c$
16 For $j \in [1\,..\,n] \setminus \{i\}$:
17 $\quad k_j \leftarrow \mathsf{K.dec}_j(sk_j, c_j)$
18 $\quad$ If $k_j = \perp$: Return $\perp$
19 If $c_i \in C_i^*$:
20 $\quad k_i \leftarrow k_i^*$
21 Else:
22 $\quad k_i \leftarrow \mathrm{Dec}(c_i)$
23 $\quad$ If $k_i = \perp$: Return $\perp$
24 $k \leftarrow W(k_1, \ldots, k_n, c)$
25 Return k

Fig. 8. Adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against session-key indistinguishability of K_i from adversary $(\mathcal{A}_1, \mathcal{A}_2)$ against session-key indistinguishability of K.

Analysis Games G_0 and G_1 only differ on the key k_i^* used to compute k^* for $\mathcal{A}_2$, and, consequently, when answering $\mathcal{A}_2$'s decapsulation queries involving c_i^*. If $\mathcal{B}$ is run by the game KIND^0, that is, key k_i^* is a real key output of $\mathsf{K.enc}_i$, then $\mathcal{B}$ perfectly emulates game G_0. Otherwise, if $\mathcal{B}$ is run by the game KIND^1, and thus the key k_i^* is uniform, then $\mathcal{B}$ emulates G_1. Hence

$$\Pr[\mathrm{G}_0 \Rightarrow 1] = \Pr[\mathrm{KIND}^0 \Rightarrow 1]$$

and

$$\Pr[\mathrm{G}_1 \Rightarrow 1] = \Pr[\mathrm{KIND}^1 \Rightarrow 1] \ .$$

Lastly we observe that $\mathcal{B}$ issues at most as many decapsulation queries as $\mathcal{A}$. Our claim follows. □

Game G_2 We add line 10 and line 26. Thus, whenever W is evaluated on a ciphertext whose ith component is c_i^* (that is, either when computing the challenge session key k^* or when answering decapsulation queries involving c_i^* as the ith ciphertext component) the output is overwritten with a uniform value from $\mathcal{Y}$.

Claim 3. There exists an adversary $\mathcal{C}$ against the split-key pseudorandomness security of W that issues at most $q_d + 1$ evaluation queries such that

$$|\Pr[\mathrm{G}_1 \Rightarrow 1] - \Pr[\mathrm{G}_2 \Rightarrow 1]| \leq \mathrm{Adv}_{W,i}^{\mathrm{pr}}(\mathcal{C}) \ ,$$

and the running time of $\mathcal{C}$ is roughly the running time of $\mathcal{A}$.

Proof. We construct an adversary $\mathcal{C}$ that breaks the split-key pseudorandomness of W on the ith key if $\mathcal{A}$ distinguishes between games G_1 and G_2.

Adversary $\mathcal{C}$ runs $\mathsf{K.gen}_j$ for all $j \in [1\,..\,n]$ to instantiate all KEMs (see lines 01–03). Then for each KEM K_j it generates a pair key-ciphertext (k_j^*, c_j^*) (lines 05 and 06). All ciphertexts, and all the keys k_j^* for $j \neq i$, are collected and used as input for a call to Eval to generate $\mathcal{A}_2$'s challenge (lines 07–09). To answer the decapsulation queries of $\mathcal{A}$ on input $c_1\,..\,c_n$, the adversary keeps track of previous decapsulation queries and returns the same result for two queries with the same input (line 14). $\mathcal{C}$ uses the secret keys it generated to decapsulate all ciphertext components c_j for $j \neq i$ (lines 16–18). The same procedure is used to decapsulate c_i if $c_i \neq c_i^*$; otherwise it queries its own decapsulation oracle (lines 19–25).

Analysis First we note that by the conditions in lines 13 and 14 in Figure 9 all calls to Eval by $\mathcal{C}^b$ have different input and thus we can always use Eval to simulate W.

Adversary $\mathcal{C}^{\text{Eval}}$

00 $C^*, C_i^* \leftarrow \emptyset$; $L[\cdot] \leftarrow \perp$
01 For $j \leftarrow 1$ to n:
02 $\quad (pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$
03 $pk \leftarrow (pk_1, \ldots, pk_n)$
04 $st \leftarrow_{\$} \mathcal{A}_1^{\text{Dec}}(pk)$
05 For $j \leftarrow 1$ to n:
06 $\quad (k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j(pk_j)$
07 $k'^* \leftarrow k_1^* \,..\, k_{i-1}^* k_{i+1}^* \,..\, k_n^*$
08 $c^* \leftarrow (c_1^*, \ldots, c_n^*)$
09 $k^* \leftarrow \text{Eval}(k'^*, c^*)$
10 $C^* \leftarrow C^* \cup \{c^*\}$; $C_i^* \leftarrow C_i^* \cup \{c_i^*\}$
11 $b' \leftarrow_{\$} \mathcal{A}_2^{\text{Dec}}(st, c^*, k^*)$
12 Stop with b'

If $\mathcal{A}$ calls $\text{Dec}(c)$:
13 If $c \in C^*$: Abort
14 If $L[c] \neq \perp$: Return $L[c]$
15 $(c_1, \ldots, c_n) \leftarrow c$
16 For $j \in [1 \,..\, n] \setminus \{i\}$:
17 $\quad k_j \leftarrow \mathsf{K.dec}_j(sk_j, c_j)$
18 $\quad$ If $k_j = \perp$: Return $\perp$
19 If $c_i \in C_i^*$:
20 $\quad k' \leftarrow k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n$
21 $\quad L[c] \leftarrow \text{Eval}(k', c)$
22 Else:
23 $\quad k_i \leftarrow \mathsf{K.dec}_i(sk_i, c_i)$
24 $\quad$ If $k_i = \perp$: Return $\perp$
25 $\quad L[c] \leftarrow W(k_1 \,..\, k_i \,..\, k_n, c)$
26 Return $L[c]$

Fig. 9. Adversary $\mathcal{C}$ against multi-key pseudorandomness of F.

Observe that when $\mathcal{C}$ plays against PR_i^0 we are implicitly setting k_i^* as the key internally generated by PR_i^0. Hence $\mathcal{C}$ correctly simulates game G_1 to $\mathcal{A}$. Otherwise when $\mathcal{C}$ plays against PR_i^1 the oracle Eval consistently outputs random elements in $\mathcal{K}$. Thus $\mathcal{C}$ correctly simulates game G_2 to $\mathcal{A}$. Therefore

$$\Pr[\text{G}_1 \Rightarrow 1] = \Pr[\text{PR}_i^0 \Rightarrow 1]$$

and

$$\Pr[\text{G}_2 \Rightarrow 1] = \Pr[\text{PR}_i^1 \Rightarrow 1] \ .$$

Thus

$$|\Pr[\text{G}_1 \Rightarrow 1] - \Pr[\text{G}_2 \Rightarrow 1]| \leq \text{Adv}_{W,i}^{\text{pr}}(\mathcal{C}) \ .$$

We count the number of Eval queries by $\mathcal{C}$. From the definition of $\mathcal{C}$ we see that the oracle Eval is called once to generate the challenge. Further, for each Dec query by $\mathcal{A}$, $\mathcal{C}$ queries Eval at most once. □

Game G_3 We remove lines 26 to undo the modifications of the Dec oracle introduced in game G_2. Thus, during decapsulation, whenever W is evaluated on a ciphertext whose ith component is c_i^* the output is computed evaluating the function W on the decapsulated keys instead of returning a uniform input.

Claim 4. There exists an adversary $\mathcal{C}'$ against the split-key pseudorandomness security of W that issues at most q_d evaluation queries such that

$$|\Pr[\text{G}_2 \Rightarrow 1] - \Pr[\text{G}_3 \Rightarrow 1]| \leq \text{Adv}_{W,i}^{\text{pr}}(\mathcal{C}') \ ,$$

and the running time of $\mathcal{C}'$ is roughly the running time of $\mathcal{A}$.

Proof. Adversary $\mathcal{C}'$ is essentially the same as adversary $\mathcal{C}$ in Figure 9, with the exception that we replace line 09 with the generation of a uniform session key ($k^* \leftarrow_{\$} \mathcal{K}$). The proof analysis is the same as in Claim 3. Notice that since this time the challenge session key is uniform, $\mathcal{C}'$ calls Eval just q_d times instead of $q_d + 1$. □

Note that, currently, the only difference from game G_1 is the addition of line 10, i.e., the challenge session key k^* is uniform.

Game G_4 Line 07 is removed to undo the modification introduced in game G_1. That is, we replace the uniform key k_i^* with a real key output by $\mathsf{K.enc}_i(pk_i)$.

Claim 5. There exists an adversary $\mathcal{B}' = (\mathcal{B}_1', \mathcal{B}_2')$ against the session-key indistinguishability of K_i that issues at most q_d decapsulation queries such that

$$|\Pr[\text{G}_3 \Rightarrow 1] - \Pr[\text{G}_4 \Rightarrow 1]| \leq \text{Adv}_{\mathsf{K}_i}^{\text{kind}}(\mathcal{B}') \ ,$$

and the running time of $\mathcal{B}'$ is roughly the running time of $\mathcal{A}$.

Proof. Adversary $\mathcal{B}'$ is the same as adversary $\mathcal{B}$ in Figure 8, with the exception that we replace line 10 with the generation of a uniform session key ($k^* \leftarrow_{\$} \mathcal{K}$). The proof analysis is the same as in Claim 2. □

Claim 6. $\Pr[\mathrm{G}_4 \Rightarrow 1] = \Pr[\mathrm{KIND}^1 \Rightarrow 1]$.

This follows immediately from the correctness of K_i and the fact that the decapsulation algorithm is deterministic.

The proof of the main statement follows from collecting the statements from Claims 1 to 6. □

4 Split-key PRFs in idealized models

In the previous section we have shown that if the core function of the parallel combiner is split-key pseudorandom, then said combiner preserves CCA security of any of its ingredient KEMs. It remains to present explicit, practical constructions of skPRFs.

In our first approach we proceed as follows: Given some keys $k_1, \ldots, k_n$ and some input x, we mingle together the keys to build a new key k for some (single-key) pseudorandom function F. The output of our candidate skPRF is obtained evaluating $F(k, x)$. In this section we consider variations on how to compute the PRF key k, along with formal proofs for the security of the corresponding candidate skPRFs.

Considering our parallel combiner with such skPRF, evaluating a session key becomes relatively efficient compared to the unavoidable cost of running n distinct encapsulations. Alas, the security of the constructions in this section necessitates some idealized building block, that is, a random oracle or an ideal cipher.

We attempt to abate this drawback by analyzing the following construction form different angles:

$$W(k_1, \ldots, k_n, x) \coloneqq F(\pi(k_n, \pi(\ldots \pi(k_1, 0) \ldots)), x) \ , \tag{3}$$

where F is a pseudorandom function and π is a pseudorandom permutation. Specifically, we show that W is an skPRF if π is modeled as an ideal cipher (Lemma 5) or F is modeled as a random oracle (Lemma 6 in combination with Example 2).

This statement might be interesting in practice: When implementing such construction the real world, F could reasonably be fixed to SHA-2 (prepending the key), while AES could reasonably be chosen as π. Both primitives are believed to possess good cryptographic properties, arguably so to behave as idealized primitives. Moreover, there is no indication to assume that if one primitive failed to behave 'ideally', then the other would be confronted with the same problem.

In Section 4.1 we prove that the construction above is secure in the ideal cipher model. In Section 4.2 we give some secure constructions in the case that F is modeled as a random oracle.

4.1 Split-key PRFs in the ideal cipher model

Here we consider constructions of skPRFs where the key-mixing step is conducted in the ideal cipher model followed by a (standard model) PRF evaluation.

Before stating the main result of this section we introduce two additional security notions for keyed functions. The first one is a natural extension of pseudorandomness, whereby an adversary is given access to *multiple instances* of a keyed function (under uniform keys) or truly random functions.

Multi-instance pseudorandomness The security game to define multi-instance pseudorandomness of F is given in Figure 10. To any adversary $\mathcal{A}$ and any number of instances n we associate its advantage $\mathrm{Adv}^{\mathrm{mipr}}_{F,n}(\mathcal{A}) \coloneqq |\Pr[\mathrm{MIPR}^0(\mathcal{A}) \Rightarrow 1] - \Pr[\mathrm{MIPR}^1(\mathcal{A}) \Rightarrow 1]|$. Intuitively, F is *multi-instance pseudorandom* if all practical adversary achieve a negligible advantage.

While one usually considers indistinguishability between outputs of a pseudorandom functions and uniform elements, key inextractability requires instead that the PRF key be hidden from any efficient adversary. We give a formalization of the latter property in the multi-instance setting next.

Game $\mathrm{MIPR}^b(\mathcal{A})$	**Oracle** $\mathrm{Eval}(i, x)$
00 $X_1, \ldots, X_n \leftarrow \emptyset$	04 If $x \in X_i$: Abort
01 $k_1, \ldots, k_n \leftarrow_\$ \mathcal{K}$	05 $X_i \leftarrow X_i \cup \{x\}$
02 $b' \leftarrow_\$ \mathcal{A}^{\mathrm{Eval}}$	06 $y \leftarrow F(k_i, x)$
03 Stop with b'	07 $y^0 \leftarrow y$; $y^1 \leftarrow_\$ \mathcal{Y}$
	08 Return y^b

Fig. 10. Security experiments MIPR^b, $b \in \{0, 1\}$, modeling multi-instance pseudorandomness of F for n instances.

Game $\mathrm{MIKI}(\mathcal{A})$	**Oracle** $\mathrm{Eval}(i, x)$	**Oracle** $\mathrm{Check}(k)$
00 $k_1, \ldots, k_n \leftarrow_\$ \mathcal{K}$	03 $y \leftarrow F(k_i, x)$	05 If $k \in \{k_1, \ldots, k_n\}$:
01 Run $\mathcal{A}^{\mathrm{Eval},\mathrm{Check}}$	04 Return y	06 Stop with 1
02 Stop with 0		

Fig. 11. Security experiment MIKI modeling multi-instance key inextractability of F for n instances.

Multi-instance key inextractability Next we introduce multi-instance key inextractability for a keyed function F. To this end, consider the game MIKI given in Figure 11. To any adversary $\mathcal{A}$ and any number of instances n we associate its advantage $\mathrm{Adv}^{\mathrm{miki}}_{F,n}(\mathcal{A}) := \Pr[\mathrm{MIKI}(\mathcal{A}) \Rightarrow 1]$. Intuitively, F satisfies *multi-instance key inextractability* if all practical adversaries achieve a negligible advantage.

Lemma 5. *Let $\mathcal{K}$, $\mathcal{H}$ and $\mathcal{Y}$ be finite sets, $\mathcal{X}$ be a set and n a positive integer. Let $F\colon \mathcal{H} \times \mathcal{X} \to \mathcal{Y}$, $E\colon \mathcal{K} \times \mathcal{H} \to \mathcal{H}$, and $D\colon \mathcal{K} \times \mathcal{H} \to \mathcal{H}$ be functions such that for all $k \in \mathcal{K}$ the function $E(k, \cdot)$ is invertible with inverse $D(k, \cdot)$. Consider the function W defined by:*

$$W\colon \mathcal{K}^n \times \mathcal{X} \to \mathcal{Y} \ , \quad W(k_1, \ldots, k_n, x) := F(E(k_n, E(\ldots E(k_1, 0)\ldots)), x) \ .$$

If F is pseudorandom then W is split-key pseudorandom in the ideal cipher model.

More precisely, suppose that E is modeled as an ideal cipher with inverse D. Then for any $i \in [1\,..\,n]$ and for any adversary $\mathcal{A}$ against the split-key pseudorandomness of W there exists an adversary $\mathcal{B}$ against the multi-instance key inextractability of F and an adversary $\mathcal{C}$ against the multi-instance pseudorandomness of F such that:

$$\mathrm{Adv}^{\mathrm{pr}}_{W,i}(\mathcal{A}) \leq \frac{Q + nq_e}{|\mathcal{K}| - n} + 6 \cdot \frac{(Q + 2nq_e)^2}{|\mathcal{H}| - 2Q - 2nq_e} + \mathrm{Adv}^{\mathrm{miki}}_{F,q_e}(\mathcal{B}) + \mathrm{Adv}^{\mathrm{mipr}}_{F,q_e}(\mathcal{C}) \ ,$$

where q_e (resp. Q) is the maximum number of calls by $\mathcal{A}$ to the oracle Eval *(resp. to the ideal cipher or its inverse). Moreover, $\mathcal{B}$ calls at most q_e (resp. $2Q + nq_e$) times the oracle* Eval *(resp.* Check*), and $\mathcal{C}$ calls at most q_e times the oracle* Eval*. The running times of $\mathcal{B}$ and $\mathcal{C}$ are roughly the same as that of $\mathcal{A}$.*

Proof sketch. The proof consists of a sequence of games interpolating between the games PR^0_i and PR^1_i for any $i \in [1\,..\,n]$. Our final goal is to make the PRF keys used in Eval as input to F uniform, and then employ the PRF security of F. To achieve this we show that, except with a small probability, the adversary cannot manipulate the game to use anything but independent, uniformly generated values as key input to F.

The PRF keys are sequences of the form $h = E(k_n, E(\ldots E(k_1, 0)\ldots))$ for some keys $k_1\,..\,k_n$. We fix an index i: The key k_i is uniformly generated by the pseudorandomness game, and the remaining keys are chosen by the adversary on each query to Eval. The proof can be conceptually divided into two parts. Initially (games G_0–G_3) we work on the first part of the sequence, namely $h' = E(k_i, E(\ldots E(k_1, 0)\ldots))$. Here we build towards a game in which all elements h' that are generated from different key vectors $k_1\,..\,k_{i-1}$ are independent uniform values. In the next games (games G_4–G_9) we work on the second part of the sequence, namely $h = E(k_n, E(\ldots E(k_{i+1}, h')\ldots))$. Again, we show that all elements h are independent and uniform, assuming independent uniform values h'.

We describe now each single game hop. We start from game G_0, equivalent to the real game PR^0_i, and we proceed as follows. Game G_1 aborts if the key k_i is directly used as input by the adversary in one of its oracle queries. In game G_2 the output of E under the uniform key k_i is precomputed and stored in a list R, which is then used by Eval. Game G_3 aborts when, in a query to Eval, the adversary triggers

an evaluation of $E(k_{i-1}, E(\ldots E(k_1, 0)\ldots))$ that gives the same output as one of a previous evaluations using a different key vector. At this point we want to argue that an adversary sequentially evaluating $n-i$ times the ideal cipher under know keys but uniform initial input still cannot obtain anything but a(n almost) uniform output. This will be achieved by uniformly pre-generating the enciphering output used to evaluate the sequences $E(k_n, E(\ldots E(k_{i+1}, h')\ldots))$. These elements are precomputed in game G_4 and stored in a list R, but not yet used. In game G_5 the elements stored in R are removed from the range of the ideal cipher. In game G_6, the oracle Eval uses the values in R to sample the ideal cipher. Since this might not always be possible, the oracle Eval resumes standard sampling if any value to be sampled has already been set in E or D. The next game makes a step forward to guarantee that the previous condition does not occur: If the two oracles E and D have never been queried with input any value that is used as key to the PRF F, then the game aborts if any element stored in R (but not used as a PRF key) is queried to E or D. All previous steps have only involved information-theoretical arguments. In game G_8 we disjoin our simulated ideal cipher from the PRF keys. This requires many small changes to the game structure, but eventually the price paid to switch from game G_7 is the advantage in breaking multi-instance key inextractability of the PRF, i.e., to recover one of the PRF keys from the PRF output. At this point, for any fixed input $k' = k_1 \,..\, k_{i-1}k_{i+1} \,..\, k_n$ to Eval we are sampling independent, uniformly generated elements to be used as the PRF keys. Finally endowed with uniform keys, in G_9 the PRF output is replaced with uniform values. If no abort condition is triggered, then the output distributions of G_9 and PR_i^1 are identical.

We proceed now with the full proof.

Proof (Lemma 5). The description of games G_0-G_3 is in Figure 12.

Games G_0 **to** G_3
00 $T[\cdot], L[\cdot], R[\cdot], V[\cdot] \leftarrow \perp$
01 $l \leftarrow 0$
02 $X \leftarrow \emptyset$
03 For $j \leftarrow 1$ to q_e: | G_2,G_3
04 $\quad R[j] \leftarrow_\$ \mathcal{H} \setminus [R]$ | G_2,G_3
05 $k_i \leftarrow_\$ \mathcal{H}$
06 $b' \leftarrow_\$ \mathcal{A}^{\mathrm{Eval,E,D}}$
07 Stop with b'

Oracle $\mathrm{E}(k, h)$
08 If $k = k_i$: Abort | G_1-G_3
09 If $T[k, h] = \perp$:
10 $\quad T[k, h] \leftarrow_\$ \mathcal{H} \setminus [T[k, \cdot]]$
11 Return $T[k, h]$

Oracle $\mathrm{D}(k, h)$
12 If $k = k_i$: Abort | G_1-G_3
13 If $T^{-1}[k, h] = \perp$:
14 $\quad T^{-1}[k, h] \leftarrow_\$ \mathcal{H} \setminus [T^{-1}[k, \cdot]]$
15 Return $T^{-1}[k, h]$

Oracle $\mathrm{Eval}(k', x)$
16 If $x \in X$: Abort
17 $X \leftarrow X \cup \{x\}$
18 $k_1 \,..\, k_{i-1}k_{i+1} \,..\, k_n \leftarrow k'$
19 If $\exists j \in [1 \,..\, n] \setminus \{i\}, k_j = k_i$: Abort | G_1-G_3
20 $h \leftarrow 0$
21 For $j \leftarrow 1$ to $i-1$:
22 $\quad$ If $T[k_j, h] = \perp$:
23 $\quad\quad T[k_j, h] \leftarrow_\$ \mathcal{H} \setminus [T[k_j, \cdot]]$
24 $\quad h \leftarrow T[k_j, h]$
25 If $V[h] \notin \{\perp, k_1 \,..\, k_{i-1}\}$: Abort | G_3
26 $V[h] \leftarrow k_1 \,..\, k_{i-1}$ | G_3
27 If $L[h] = \perp$: | G_2,G_3
28 $\quad l \leftarrow l + 1$ | G_2,G_3
29 $\quad L[h] \leftarrow l$ | G_2,G_3
30 If $T[k_i, h] = \perp$:
31 $\quad T[k_i, h] \leftarrow_\$ \mathcal{H} \setminus [T[k_i, \cdot]]$
32 $h \leftarrow T[k_i, h]$
33 $h \leftarrow R[L[h]]$ | G_2,G_3
34 For $j \leftarrow i+1$ to n:
35 $\quad$ If $T[k_j, h] = \perp$:
36 $\quad\quad T[k_j, h] \leftarrow_\$ \mathcal{H} \setminus [T[k_j, \cdot]]$
37 $\quad h \leftarrow T[k_j, h]$
38 $y \leftarrow F(h, x)$
39 Return y

Fig. 12. Games G_0–G_3 as used in the proof of Lemma 5. Note that i is implicitly a parameter of all games above. Implicitly, if $T[k, h] = h'$ then $T^{-1}[k, h'] = h$, and vice versa.

Game G_0 Game G_0 is equivalent to game PR_i^0. The ideal cipher is correctly simulated each time the oracle E and D are queried, as well as internally to simulate Eval.

Game G_1 Game G_1 differs from G_0 by the addition of three abort conditions. Namely, every time that the key k_i is used as key input to E, D, or Eval the game aborts. The key k_i is randomly generated and

hidden from the adversary; hence the adversary can only try to guess it. The probability to guess k_i in all Q queries to E or D is $Q/|\mathcal{K}|$. The probability to guess the correct key in all q_e to Eval is smaller than $nq_e/(|\mathcal{K}| - n)$. Summing up over all queries to the oracles we get:

$$|\Pr[\mathrm{G}_0 \Rightarrow 1] - \Pr[\mathrm{G}_1 \Rightarrow 1]| \leq \frac{Q}{|\mathcal{K}|} + \frac{nq_e}{|\mathcal{K}| - n} \ . \tag{4}$$

Game G_2 Game G_2 differs from G_1 by the addition of the map R and its use to simulate the ideal cipher under key k_i in Eval. Since by the abort conditions added in G_1 the ideal cipher is never queried under key k_i and all elements in the list R are distinct, we are correctly simulating the ideal cipher. Hence:

$$\Pr[\mathrm{G}_1 \Rightarrow 1] = \Pr[\mathrm{G}_2 \Rightarrow 1] \ . \tag{5}$$

Game G_3 Game G_3 differs from G_2 by the use of the map V to check if values $E(k_{i-1}, E(\ldots E(k_1, 0)\ldots))$ implicitly computed by the oracle Eval under distinct keys $k_1 \,..\, k_{i-1}$ collide; in case, G_3 aborts.

To compute the abort probability we study a restricted choice of ideal ciphers: That in which the ideal cipher always samples inputs that have not been seen in previous enciphering and deciphering queries. With such a cipher we are guaranteed that the abort condition added in game G_3 does not trigger.

We compute the probability of a random cipher to always sample fresh elements. Assume that the enciphering and deciphering oracles are called at most q times in total. Consider any query to the enciphering or deciphering oracles. If all previous enciphering/deciphering queries yielded unseen elements, then at most $2q$ elements have been seen to this point. The probability to sample any of the at most $2q$ previously seen elements for each of the q oracle queries is then upper bounded by $2q^2/(|\mathcal{H}| - 2q)$.

Since in game G_3 we encipher/decipher at most $Q + nq_e$ times we can write:

$$|\Pr[\mathrm{G}_2 \Rightarrow 1] - \Pr[\mathrm{G}_3 \Rightarrow 1]| \leq \frac{(Q + nq_e)^2}{|\mathcal{H}| - Q - nq_e} \ . \tag{6}$$

The description of games G_4-G_9 is in Figure 13.

Game G_4 Game G_4 is a rewriting of G_3. Hence:

$$\Pr[\mathrm{G}_3 \Rightarrow 1] = \Pr[\mathrm{G}_4 \Rightarrow 1] \ . \tag{7}$$

Most importantly, from now on we assume that before each Eval query under the keys $k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n$ the adversary queries E to fill the table T with the values needed to run line 38. This is possible since the adversary knows in advance all required input to the block cipher (the chains of block cipher evaluations start with input 0).

In this game we repurpose the lists R and L used in game G_3, as well as the index l, which itself becomes a list of indices. The old and new lists are related as follow. The list $R[i, \cdot]$ in G_4 is equivalent to the list $R[\cdot]$ in G_3, the list $L[\cdot]$ with a single input element in G_4 is equivalent to the list $L[\cdot]$ in G_3, and the index $l[i]$ in G_4 is equivalent to the index l in G_3.

We defined two new procedures, Sample.E and Sample.D, that cannot be called by the adversary and are used to sample new elements for the ideal cipher.

Moreover, in game G_4 we introduce the set S^E_{in} (resp. S^D_{in}), which keeps track of the input values to the oracle E (resp. D).

Game G_5 Game G_5 differs from G_4 by the output distribution of the ideal cipher. In the following we compute the difference in the output distributions. For this we analyze a single call to Sample.E or Sample.D in game G_4 such that all previously sampled elements (by both functions) do not belong to $[R]$. If the newly sampled element does not belong to $[R]$, then the simulation is correct for both G_4 and G_5. Note that the space from which an output value is sampled has at least size $|\mathcal{H}| - (Q + nq_e)$. The probability to sample an element in $[R]$ for all the at most $Q + nq_e$ sampling queries is bounded by $(Q + nq_e)\,|[R]|\,/(|\mathcal{H}| - Q - nq_e)$. Since $|[R]| \leq nq_e$, we can compute the difference in probability of the two games as:

$$|\Pr[\mathrm{G}_4 \Rightarrow 1] - \Pr[\mathrm{G}_5 \Rightarrow 1]| \leq \frac{(Q + nq_e)nq_e}{|\mathcal{H}| - Q - nq_e} \ . \tag{8}$$

Games G_4 **to** G_9

00 $T[\cdot], L[\cdot], R[\cdot], V[\cdot] \leftarrow \perp$
01 $l[\cdot] \leftarrow 0$
02 $X, S^E_{\text{in}}, S^D_{\text{in}} \leftarrow \emptyset$
03 revealed $\leftarrow$ *false*
04 For $m \leftarrow i$ to n:
05 For $j \leftarrow 1$ to q_e:
06 $R[m,j] \leftarrow_{\$} \mathcal{H} \setminus [R]$
07 For $j \leftarrow 1$ to q_e: | G_8,G_9
08 $R[n,j] \leftarrow_{\$} \mathcal{H}$ | G_8,G_9
09 $k_i \leftarrow_{\$} \mathcal{H}$
10 $b' \leftarrow_{\$} \mathcal{A}^{\text{Eval,E,D}}$
11 Stop with b'

Oracle $\text{E}(k,h)$

12 If $k = k_i$: Abort
13 If $[R[n,\cdot]] \cap (S^E_{\text{in}} \cup S^D_{\text{in}}) = \emptyset$: | G_7
14 If $h \in [R] \setminus [R[n,\cdot]]$: Abort | G_7
15 If $h \in [R] \setminus [R[n,\cdot]]$: Abort | G_8,G_9
16 $S^E_{\text{in}} \leftarrow S^E_{\text{in}} \cup \{h\}$
17 $h' \leftarrow_{\$} \text{Sample.E}(k,h)$
18 Return h'

Oracle $\text{D}(k,h)$

19 If $k = k_i$: Abort
20 If $[R[n,\cdot]] \cap (S^E_{\text{in}} \cup S^D_{\text{in}}) = \emptyset$: | G_7
21 If $h \in [R] \setminus [R[n,\cdot]]$: Abort | G_7
22 If $h \in [R] \setminus [R[n,\cdot]]$: Abort | G_8,G_9
23 $S^D_{\text{in}} \leftarrow S^D_{\text{in}} \cup \{h\}$
24 $h' \leftarrow_{\$} \text{Sample.D}(k,h)$
25 Return h'

Proc $\text{Sample.E}(k,h)$

26 If $T[k,h] = \perp$:
27 $U \leftarrow [T[k,\cdot]]$ | G_4
28 $U \leftarrow [T[k,\cdot]] \cup [R]$ | G_5-G_7
29 $U \leftarrow [T[k,\cdot]] \cup ([R] \setminus [R[n,\cdot]])$ | G_8,G_9
30 $T[k,h] \leftarrow_{\$} \mathcal{H} \setminus U$
31 Return $T[k,h]$

Oracle $\text{Eval}(k',x)$

32 If $x \in X$: Abort
33 $X \leftarrow X \cup \{x\}$
34 $k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n \leftarrow k'$
35 If $\exists j \in [1\,..\,n] \setminus \{i\}, k_j = k_i$: Abort
36 $h \leftarrow 0$
37 For $j \leftarrow 1$ to $i-1$:
38 $h \leftarrow T[k_j, h]$
39 If $V[h] \notin \{\perp, k_1 \,..\, k_{i-1}\}$: Abort
40 $V[h] \leftarrow k_1 \,..\, k_{i-1}$
41 If $L[h] = \perp$:
42 $l[i] \leftarrow l[i] + 1$
43 $L[h] \leftarrow l[i]$
44 $h \leftarrow R[i, L[h]]$
45 $h' \leftarrow h$
46 If $[R] \cap (S^E_{\text{in}} \cup S^D_{\text{in}}) \neq \emptyset$: | G_6-G_9
47 If *true*: | G_4,G_5
48 For $j \leftarrow i+1$ to n:
49 $h \leftarrow_{\$} \text{Sample.E}(k_j, h)$
50 Else: | G_6-G_9
51 For $j \leftarrow i+1$ to n: | G_6-G_9
52 If $L[h', k_{i+1}, \ldots, k_j] = \perp$: | G_6-G_9
53 $l[j] \leftarrow l[j] + 1$ | G_6-G_9
54 $L[h', k_{i+1}, \ldots, k_j] \leftarrow l[j]$ | G_6-G_9
55 $T[k_j, h] \leftarrow R[j, L[h', k_{i+1}, \ldots, k_j]]$ | G_6,G_7
56 $h \leftarrow R[j, L[h', k_{i+1}, \ldots, k_j]]$ | G_6-G_9
57 $y \leftarrow F(h,x)$
58 $y \leftarrow_{\$} \mathcal{Y}$ | G_9
59 Return y

Proc $\text{Sample.D}(k,h)$

60 If $T^{-1}[k,h] = \perp$:
61 $U \leftarrow [T^{-1}[k,\cdot]]$ | G_4
62 $U \leftarrow [T^{-1}[k,\cdot]] \cup [R]$ | G_5-G_7
63 $U \leftarrow [T^{-1}[k,\cdot]] \cup ([R] \setminus [R[n,\cdot]])$ | G_8,G_9
64 $T^{-1}[k,h] \leftarrow_{\$} \mathcal{H} \setminus U$
65 Return $T^{-1}[k,h]$

Fig. 13. Games G_4–G_9 as used in the proof of Lemma 5. Note that i is implicitly a parameter of all games above. Moreover, if $T[k,h] = h'$ then $T^{-1}[k,h'] = h$, and vice versa. We assume that the adversary has already queried the first $i-1$ ideal cipher queries of each Eval query.

Game G_6 In game G_6, if the condition in line 46 is not fulfilled the game uses the list R to sample the ideal cipher while simulating Eval.

We describe first the roles of the lists R, L, and l. Intuitively, R stores all the random elements used to sample the ideal cipher output for any keys $k_{i+1} \,..\, k_n$ in each evaluation of Eval (note that at least $(n-i) \cdot q_e$ new elements are needed). In the following we fix a specific key position $j > i$ and we explain the role of the lists. The values $R[j,\cdot]$ will be used to sample the cipher under the key in the jth position of k' for the input that has been computed with the previous evaluation of the ideal cipher. We want the randomness to be fresh each time we use different keys or input values, but the same when they are reused. The lists L and l are employed for this role. The list L memorizes the position of the randomness associated with the specific input h' and the keys $k_1 \,..\, k_j$. Note that the input of L has variable length, depending on j. The random values in R are assigned in order from $R[j,1]$ to $R[j,q_e]$. Every time that a new value in R is assigned, the index $l[j]$ is incremented. This procedure is carried out in lines 52–54.

Now we argue that the change is syntactically possible, that is, sampling can be done while keeping the maps T and T^{-1} consistent with each other and without remapping any existing value. Notice that, because of the changes in G_5, neither map gets to a value in R as output of Sample.E or Sample.E.

Moreover, by the condition in line 46 the game resorts to sampling using R only if no elements of $[R]$ were input to E or D. This shows that the two maps T and T^{-1} allow such alternative sampling.

Finally we compute the difference of the two output distribution. Let us consider a single query to Eval that uses the modified sampling. Note that we only need to consider fresh sampling: If some previously sampled values are used again, then we return the same value in both game G_5 and G_6. Thus we only need to consider fresh values r of R. Notice that since r was hidden from the adversary we can assume the sampling of r to be done concurrently with the call to Eval. This creates two differences in sampling. Namely, in G_6 the value r is sampled uniformly in $\mathcal{H} \setminus [R]$ but in G_5 it is sample uniformly in $\mathcal{H} \setminus [T[k, \cdot]]$ for some key k instead. In the worst case the two sets $[T[k, \cdot]]$ and $[R]$ are disjoint. The size of the two sets are at most $Q + nq_e$ and nq_e respectively. We can bound the probability of distinguishing the two games in the worst-case scenario with the bound $(Q + 2 \cdot nq_e)/|\mathcal{H}| - Q - nq_e$. A similar argument to include all $Q + nq_e$ times in which we sample r, which allows us to write:

$$|\Pr[\mathrm{G}_5 \Rightarrow 1] - \Pr[\mathrm{G}_6 \Rightarrow 1]| \leq \frac{(Q + 2nq_e)nq_e}{|\mathcal{H}| - Q - nq_e} \quad . \tag{9}$$

Game G_7 Game G_7 differs from G_6 by two additional abort conditions on the oracles E and D. Precisely, if any of the two oracles is queried on any input that belongs to $[R] \setminus [R[n, \cdot]]$ then, unless the decapsulation oracle was queried with one of the PRF keys in $[R[n, \cdot]]$ as input, the game aborts.

Take any query to E or D. Then we know that the adversary has no information on the values in $[R[n, \cdot]]$. In fact they were never queried as input to the oracles E and D, otherwise the game would have aborted, no information is released by Eval that is not in $[R[n, \cdot]]$, and by our added condition the oracle D was never queried under any element in $[R[n, \cdot]]$. Hence the adversary can do no better than to guess the value. Repeating the argument for all Q queries yields:

$$|\Pr[\mathrm{G}_6 \Rightarrow 1] - \Pr[\mathrm{G}_7 \Rightarrow 1]| \leq \frac{Qnq_e}{|\mathcal{H}|} \quad . \tag{10}$$

Game G_8 Game G_8 differs from G_7 most importantly in line 54: The values in $[R]$ are not linked to the map T anymore, and hence they are uniform and hidden to the adversary (except as PRF keys, use to evaluate the PRF). Other modifications involve the sampling procedures (now the sampling includes output values in $[R[n, \cdot]]$), the oracles E and D (the check if the values in $[R[n, \cdot]]$ have been queried as input of D are dropped) and the generation of the elements in $[R[n, \cdot]]$ admits now repetitions.

We prove that for every adversary $\mathcal{A}$ that distinguishes between games G_7 and G_8 we can build an adversary $\mathcal{B}$ that breaks the multi-instance key inextractability security game (MIKI) for the PRF F described in Figure 11. The code for $\mathcal{B}$ is in Figure 14.

Note that in the reduction the list $\bar{R}$ has the same role of R in games G_7 and G_7 with the exception that the nth column is not defined. In particular, implicitly $[R] \setminus [R[n, \cdot]] = [\bar{R}]$.

We observe that since game G_7 generates distinct PRF keys the simulation of $\mathcal{B}$ is not perfect. However, the security game with distinct keys and that with perfectly uniform keys are identical up to a term of $q_e^2 / |\mathcal{H}|$.

Next we discuss part of the behavior of adversary $\mathcal{B}$. In line 06 we make sure that the elements of $\bar{R}$ and the PRF keys are all distinct. With lines 11 and 16 we are guaranteed that no PRF keys are ever input to E or D. Because of lines 12 and 17 we know that no element in $[\bar{R}]$ is ever input to E or D; for this reasons the procedure Sample.E is not needed to simulate calls to Eval by $\mathcal{A}$. The output sampling in lines 21 and 49 is correct, unless $\mathcal{A}$ breaks key inextractability.

To sum up or conclusion we write that for every adversary $\mathcal{A}$ that distinguishes between games G_7 and G_8 there exists an adversary $\mathcal{B}$ against game MIKI such that:

$$|\Pr[\mathrm{G}_7 \Rightarrow 1] - \Pr[\mathrm{G}_8 \Rightarrow 1]| \leq \mathrm{Adv}^{\mathrm{miki}}_{F,q_e}(\mathcal{B}) + \frac{q_e^2}{|\mathcal{H}|} \quad . \tag{11}$$

Moreover, if adversary $\mathcal{A}$ calls at most q_e (resp. Q) times the oracle Eval (resp. E or D) then $\mathcal{B}$ calls at most q_e (resp. $2Q + nq_e$) times the oracle Eval (resp. Check).

Adversary $\mathcal{B}^{\text{Eval,Check}}$
00 $T[\cdot], L[\cdot], \bar{R}[\cdot], V[\cdot] \leftarrow \perp$
01 $l[\cdot] \leftarrow 0$
02 $X \leftarrow \emptyset$
03 For $m \leftarrow i$ to $n-1$:
04 For $j \leftarrow 1$ to q_e:
05 $\bar{R}[m,j] \leftarrow_{\$} \mathcal{H} \setminus [\bar{R}]$
06 bad $\leftarrow$ Check($\bar{R}[m,j]$); If bad: Abort
07 $k_i \leftarrow_{\$} \mathcal{H}$
08 $b' \leftarrow_{\$} \mathcal{A}^{\text{Eval,E,D}}$
09 Stop with b'

If $\mathcal{A}$ calls E(k,h):
10 If $k = k_i$: Abort
11 bad $\leftarrow$ Check(h); If bad: Abort
12 If $h \in [\bar{R}]$: Abort
13 $h' \leftarrow_{\$}$ Sample.E(k,h)
14 Return h'

If $\mathcal{A}$ calls D(k,h):
15 If $k = k_i$: Abort
16 bad $\leftarrow$ Check(h); If bad: Abort
17 If $h \in [\bar{R}]$: Abort
18 $h' \leftarrow_{\$}$ Sample.D(k,h)
19 Return h'

Proc Sample.E(k,h)
20 If $T[k,h] = \perp$:
21 $T[k,h] \leftarrow_{\$} \mathcal{H} \setminus ([T[k,\cdot]] \cup [\bar{R}])$
22 bad $\leftarrow$ Check($T[k,h]$); If bad: Abort
23 Return $T[k,h]$

If $\mathcal{A}$ calls Eval(k',x):
24 If $x \in X$: Abort
25 $X \leftarrow X \cup \{x\}$
26 $k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n \leftarrow k'$
27 If $\exists j \in [1\,..\,n] \setminus \{i\}, k_j = k_i$: Abort
28 $h \leftarrow 0$
29 For $j \leftarrow 1$ to $i-1$:
30 $h \leftarrow T[k_j, h]$
31 If $V[h] \notin \{\perp, k_1 \,..\, k_{i-1}\}$:
32 $V[h] \leftarrow k_1 \,..\, k_{i-1}$
33 If $L[h] = \perp$:
34 $l[i] \leftarrow l[i] + 1$
35 $L[h] \leftarrow l[i]$
36 $h \leftarrow \bar{R}[i, L[h]]$
37 $h' \leftarrow h$
38 For $j \leftarrow i+1$ to $n-1$:
39 If $L[h', k_{i+1}, \ldots, k_j] = \perp$:
40 $l[j] \leftarrow l[j] + 1$
41 $L[h', k_{i+1}, \ldots, k_j] \leftarrow l[j]$
42 $h \leftarrow \bar{R}[j, L[h', k_{i+1}, \ldots, k_j]]$
43 If $L[h', k_{i+1}, \ldots, k_n] = \perp$:
44 $l[n] \leftarrow l[n] + 1$
45 $L[h', k_{i+1}, \ldots, k_n] \leftarrow l[n]$
46 $y \leftarrow$ Eval$(L[h', k_{i+1}, \ldots, k_n], x)$
47 Return y

Proc Sample.D(k,h)
48 If $T^{-1}[k,h] = \perp$:
49 $T^{-1}[k,h] \leftarrow_{\$} \mathcal{H} \setminus ([T^{-1}[k,\cdot]] \cup [\bar{R}])$
50 bad $\leftarrow$ Check($T^{-1}[k,h]$); If bad: Abort
51 Return $T^{-1}[k,h]$

Fig. 14. Adversary $\mathcal{B}$ against multi-instance key inextractability of F. Implicitly, if $T[k,h] = h'$ then we have $T^{-1}[k,h'] = h$, and vice versa.

Game G_9 Game G_9 differs from G_8 by using uniform elements as output of Eval instead of real evaluations of F.

We prove that for every adversary $\mathcal{A}$ that distinguishes between games G_8 and G_9 we can build an adversary $\mathcal{C}$ that breaks the multi-instance pseudorandomness (MIPR) of the PRF F described in Figure 10. The code for $\mathcal{C}$ is in Figure 15.

Since the PRF keys are now uniform and hidden, the reduction is straightforward.

To sum up or conclusion we write that for every adversary $\mathcal{A}$ that distinguishes between games G_8 and G_9 there exists an adversary $\mathcal{C}$ that distinguishes between games MIPR^0 and MIPR^1 such that:

$$|\Pr[G_8 \Rightarrow 1] - \Pr[G_9 \Rightarrow 1]| \leq \text{Adv}^{\text{mipr}}_{F,q_e}(\mathcal{C}) \ . \tag{12}$$

Moreover, if adversary $\mathcal{A}$ calls at most q_e (resp. Q) times the oracle Eval then $\mathcal{C}$ calls at most q_e times the oracle Eval.

The bound in our statement follows from combining equations (4) to (12) and some generous bounds. □

4.2 Split-key PRFs in the random oracle model

Next, we consider constructions of skPRFs where the key-mixing step employs standard model primitives. However, to achieve security we idealize the PRF that is employed afterwards. Here we identify a sufficient condition on the key-mixing function such that the overall construction achieves split-key pseudorandomness. We begin by giving the aforementioned property for the key-mixing function.

Adversary $\mathcal{C}^{\mathrm{Eval}}$
00 $T[\cdot], L[\cdot], \bar{R}[\cdot], V[\cdot] \leftarrow \perp$
01 $l[\cdot] \leftarrow 0$
02 $X \leftarrow \emptyset$
03 For $m \leftarrow i$ to n:
04 For $j \leftarrow 1$ to $q_e - 1$:
05 $\bar{R}[m, j] \leftarrow_{\$} \mathcal{H} \setminus [\bar{R}]$
06 $k_i \leftarrow_{\$} \mathcal{H}$
07 $b' \leftarrow_{\$} \mathcal{A}^{\mathrm{Eval,E,D}}$
08 Stop with b'

If $\mathcal{A}$ calls $\mathrm{E}(k, h)$:
09 If $k = k_i$: Abort
10 If $h \in [\bar{R}]$: Abort
11 $h' \leftarrow_{\$}$ Sample.E(k, h)
12 Return h'

Proc Sample.E(k, h)
13 If $T[k, h] = \perp$:
14 $U \leftarrow [T[k, \cdot]] \cup [\bar{R}]$
15 $T[k, h] \leftarrow_{\$} \mathcal{H} \setminus U$
16 Return $T[k, h]$

If $\mathcal{A}$ calls Eval(k', x):
17 If $x \in X$: Abort
18 $X \leftarrow X \cup \{x\}$
19 $k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n \leftarrow k'$
20 If $\exists j \in [1\,..\,n] \setminus \{i\}, k_j = k_i$: Abort
21 $h \leftarrow 0$
22 For $j \leftarrow 1$ to $i - 1$:
23 $h \leftarrow T[k_j, h]$
24 If $V[h] \notin \{\perp, k_1 \,..\, k_{i-1}\}$:
25 $V[h] \leftarrow k_1 \,..\, k_{i-1}$
26 If $L[h] = \perp$:
27 $l[i] \leftarrow l[i] + 1$
28 $L[h] \leftarrow l[i]$
29 $h \leftarrow \bar{R}[i, L[h]]$
30 $h' \leftarrow h$
31 For $j \leftarrow i + 1$ to $n - 1$:
32 If $L[h', k_{i+1}, \ldots, k_j] = \perp$:
33 $l[j] \leftarrow l[j] + 1$
34 $L[h', k_{i+1}, \ldots, k_j] \leftarrow l[j]$
35 $h \leftarrow \bar{R}[j, L[h', k_{i+1}, \ldots, k_j]]$
36 If $L[h', k_{i+1}, \ldots, k_n] = \perp$:
37 $l[n] \leftarrow l[n] + 1$
38 $L[h', k_{i+1}, \ldots, k_n] \leftarrow l[n]$
39 $y \leftarrow \mathrm{Eval}(L[h', k_{i+1}, \ldots, k_n], x)$
40 Return y

If $\mathcal{A}$ calls $\mathrm{D}(k, h)$:
41 If $k = k_i$: Abort
42 If $h \in [\bar{R}]$: Abort
43 $h' \leftarrow_{\$}$ Sample.D(k, h)
44 Return h'

Proc Sample.D(k, h)
45 If $T^{-1}[k, h] = \perp$:
46 $U \leftarrow [T^{-1}[k, \cdot]] \cup [\bar{R}]$
47 $T^{-1}[k, h] \leftarrow_{\$} \mathcal{H} \setminus U$
48 Return $T^{-1}[k, h]$

Fig. 15. Adversary $\mathcal{C}$ against multi-instance pseudorandomness of F. Implicitly, if $T[k, h] = h'$ then we have $T^{-1}[k, h'] = h$, and vice versa.

Almost uniformity of a key-mixing function. For all $i \in [1\,..\,n]$ let $\mathcal{K}_i$ be a finite key space and $\mathcal{K}$ any key space. Consider a function

$$g \colon \mathcal{K}_1 \times \ldots \times \mathcal{K}_n \to \mathcal{K} \ .$$

We say that g is *ϵ-almost uniform w.r.t. the ith key* if for all $k \in \mathcal{K}$ and all $k_j \in \mathcal{K}_j$ for $j \in [1\,..\,n] \setminus \{i\}$ we have:

$$\Pr_{k_i \leftarrow_{\$} \mathcal{K}_i}[g(k_1 \,..\, k_n) = k] \leq \epsilon \ .$$

We say that g is *ϵ-almost uniform* if it is *ϵ-almost uniform w.r.t. the ith key* for all $i \in [1\,..\,n]$.

We give three standard model instantiations of key-mixing functions that enjoy almost uniformity.

Example 1. Let $\mathcal{K}_1 = \ldots = \mathcal{K}_n = \mathcal{K} = \{0,1\}^k$ for some $k \in \mathbb{N}$ and define

$$g_{\oplus}(k_1 \,..\, k_n) \coloneqq \bigoplus_{j=1}^{n} k_j \ .$$

Then $g_{\oplus}$ is $1/\left|\mathcal{K}\right|$-almost uniform.

The proof follows from observing that for any $i \in [1\,..\,n]$ and any fixed $k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n$, the function $g_{\oplus}(k_1 \,..\, k_{i-1} \cdot k_{i+1} \,..\, k_n)$ is a permutation.

Example 2. Let $\mathcal{K}, \mathcal{H}$ be finite and $\pi \colon \mathcal{K} \times \mathcal{H} \to \mathcal{H}$ such that for all $k \in \mathcal{K}$ we have that $\pi(k, \cdot)$ is a permutation on $\mathcal{H}$. Let

$$g(k_1 \,..\, k_n) \coloneqq \pi(k_n, \ldots \pi(k_1, 0) \ldots) \ ,$$

for some $0 \in \mathcal{K}$.

If for all $k \in \mathcal{K}$, $\pi(k, \cdot)$ is a pseudorandom permutation (i.e., π is a blockcipher) then for all i and all $k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n$ there exists an adversary $\mathcal{A}$ against the pseudorandomness of π such that g is $\mathrm{Adv}^{\mathrm{prp}}_{\pi}(\mathcal{A}) + 1/\left|\mathcal{K}\right|$-almost uniform. Here $\mathrm{Adv}^{\mathrm{prp}}_{\pi}(\mathcal{A})$ is the advantage of $\mathcal{A}$ in distinguishing π under a uniform key from a uniform permutation.

We sketch a proof of Example 2. First, observe that, since k_j for all $j \neq i$ is known by $\mathcal{A}$, all permutations $\pi(k_j, \cdot)$ can be disregarded. Secondly, we replace the permutation $\pi(k_i, \cdot)$ with a uniform permutation, losing the term $\mathrm{Adv}^{\mathrm{prp}}_{\pi}(\mathcal{A})$. The claim follows.

Example 3. Let $\mathcal{K}_1, \ldots, \mathcal{K}_n, \mathcal{K}$ be finite. Let

$$g(k_1 \,..\, k_n) \coloneqq k_1 \,\|\, \ldots \,\|\, k_n \ ,$$

then g is $1/|\mathcal{K}|$-almost uniform.

The proof uses the same argument as in Example 1.

We now show that we can generically construct a pseudorandom skPRF from any almost-uniform key-mixing function in the random oracle model.

Lemma 6. *Let $g\colon \mathcal{K}_* \to \mathcal{K}'$ be a function. Let $H\colon \mathcal{K}' \times \mathcal{X} \to \mathcal{Y}$ be a (hash) function. Let*

$$H \diamond g\colon \mathcal{K}_* \times \mathcal{X} \to \mathcal{Y}, \ \ (H \diamond g)(k_1, \ldots, k_n, x) \coloneqq H(g(k_1 \,..\, k_n), x) \ .$$

If H is modeled as a random oracle then for any adversary $\mathcal{A}$ such that g is ϵ-almost uniform and $\mathcal{A}$ makes at most q_H H queries and q_e Eval queries and all i we have

$$\mathrm{Adv}^{\mathrm{pr}}_{i}(\mathcal{A}) \leq q_H \cdot \epsilon \ .$$

Proof sketch. Note that any adversary against the pseudorandomness of $H \diamond g$ is given access to Eval and H, the latter implementing a random oracle. Now, intuitively, $\mathcal{A}$ is unlikely to predict the output of the g invocation within an Eval query as g is almost uniform. Hence, $\mathcal{A}$ will not query H on the same input as done within Eval. Thus, even in the real game, the output of Eval is likely to be uniform.

We give a refined analysis next.

Proof (Lemma 6). We bound the distance between the probabilities of $\mathcal{A}$ outputting 1 in game PR^0_i and

Games PR^b_i	**Oracle** $\mathrm{Eval}(k', x)$	**Oracle** $\mathrm{H}(k'', x)$
00 $X \leftarrow \emptyset$	07 If $x \in X$: Abort	17 $S_{\mathrm{H}} \leftarrow S_{\mathrm{H}} \cup \{(k'', x)\}$
01 $S_E, S_{\mathrm{H}} \leftarrow \emptyset$	08 $X \leftarrow X \cup \{x\}$	18 If $H[k'', x] = \bot$:
02 $k_i \leftarrow_{\$} \mathcal{K}_i$	09 $k_1 \,..\, k_{i-1} k_{i+1} \,..\, k_n \leftarrow k'$	19 $\quad H[k'', x] \leftarrow_{\$} \mathcal{K}'$
03 $b' \leftarrow_{\$} \mathcal{A}^{\mathrm{Eval,H}}$	10 $k'' \leftarrow g(k_1 \,..\, k_i \,..\, k_n)$	20 Return $H[k'', x]$
04 If $S_{\mathrm{H}} \cap S_E \neq \emptyset$:	11 $S_E \leftarrow S_E \cup \{(k'', x)\}$	
05 $\quad$ bad $\leftarrow$ true	12 If $H[k'', x] = \bot$:	
06 Stop with b'	13 $\quad H[k'', x] \leftarrow_{\$} \mathcal{K}'$	
	14 $y \leftarrow H[k'', x]$	
	15 $y^0 \leftarrow y$; $y^1 \leftarrow_{\$} \mathcal{Y}$	
	16 Return y^b	

Fig. 16. Game PR^b_i for $i \in [1 \,..\, n]$ instantiated with $H \diamond g$.

PR^1_i. The PR^b_i game is given in Figure 16. For game PR^b_i we performed merely syntactical changes: $\mathcal{A}$ is given access to H via oracle H. Note that we record the queries to H made by $\mathcal{A}$. Two sets S_E, S_H are initialized as empty and updated in lines 01, 11, 17 and used to define an event in line 05.

Observe that for all i the PR^0_i and PR^1_i games are identical if bad does not happen: As $S_H \cap S_E$ remains empty, adversary $\mathcal{A}$ did not query H on an input that H was evaluated on during an Eval query (see line 14). Hence, $y \leftarrow \mathrm{H}(k'', x)$ is uniform and thus, $y^0 \leftarrow y$ and $y^1 \leftarrow \mathcal{Y}$ are identically distributed.

We bound $\Pr[\mathrm{bad}]$ in PR^1_i. To this end, let (k''_j, x_j) for $j \in [1 \,..\, q_H]$ denote the H queries made by $\mathcal{A}$. We have

$$\Pr[\mathrm{bad}] = \Pr[S_{\mathrm{H}} \cap S_E \neq \emptyset] \leq \sum_{j=1}^{q_H} \Pr[(k''_j, x_j) \in S_E] \ .$$

Recall from line 07 that for every $x \in \mathcal{X}$ there is at most one query $\mathrm{Eval}(\cdot, x)$ by $\mathcal{A}$. Hence, for each (k''_j, x_j) in S_H there is at most one element of the form $(\cdot, x_j)$ in S_E. Assume it exists[7] and let k''_{x_j} be such that $(k''_{x_j}, x_j) \in S_E$ denotes that element. Then

$$\sum_{j=1}^{q_H} \Pr[(k''_j, x_j) \in S_E] \leq \sum_{j=1}^{q_H} \Pr[k''_j = k''_{x_j}] = \sum_{j=1}^{q_H} \Pr_{k_{i,x_j} \leftarrow \mathcal{K}_i}[k''_j = g(k'_1 \,..\, k_{i-1} k_{i,x_j} k_{i+1} \,..\, k'_n)]$$

for $k'_1, \ldots, k'_{i-1}, k'_{i+1}, \ldots, k'_n$ chosen by $\mathcal{A}$ and uniform k_{i,x_j} that satisfies $g(k'_1 \,..\, k_{i-1} k_{i,x_j} k_{i+1} \,..\, k'_n) = k''_{x_j}$. Eventually, we can employ the ϵ-almost uniformity of g to conclude that

$$\sum_{j=1}^{q_H} \Pr_{k_{i,x_j} \leftarrow \mathcal{K}_i}[k''_j = g(k'_1 \,..\, k_{i-1} k_{i,x_j} k_{i+1} \,..\, k'_n)] \leq \sum_{j=1}^{q_H} \epsilon \leq q_H \cdot \epsilon \ .$$

□

Next, we show that, generally, the construction from Lemma 6 does not yield a split-key pseudorandom function in the standard model.

Lemma 7. *Let g be with syntax as in Lemma 6 and let F be with syntax as H in Lemma 6. There exists an instantiation of g and F such that g is almost uniform and F is pseudorandom but*

$$F \diamond g \colon \mathcal{K}_* \times \mathcal{X} \to \mathcal{Y} \ , \quad (F \diamond g)(k_1, \ldots, k_n, x) \coloneqq F(g(k_1 \,..\, k_n), x)$$

is not a pseudorandom skPRF.

Proof. We saw in Example 1 that $g_\oplus$ is almost uniform. Further, we saw in Lemma 3 that, when using $F \diamond g_\oplus$ as a core function, there exists a pseudorandom function F such that the combined KEM is not CCA secure. If $F \diamond g_\oplus$ (with such F) were split-key pseudorandom, then this would contradict Theorem 1. □

5 KEM combiners in the standard model

Our approach was hitherto to mix the keys $k_1, \ldots, k_n$ to obtain a key for a PRF, which was then evaluated on the ciphertext vector. The drawback of this is that to show security we had to turn to idealized primitives. In the following we embark on a different approach, with the goal to obtain a standard model construction.

5.1 The PRF-then-XOR split-key PRF

Here we abstain from mixing the keys together, but use each key k_i in a PRF evaluation. The security of the model is offset by its price in terms of efficiency: When employed in a parallel combiner, the skPRF requires n PRF calls, whereas for our constructions secure in idealized models in Section 4.2 a single call to a PRF suffices. We give our construction next.

As before we want to allow possibly different session-key spaces of the ingredient KEMs. Thus, as the keys k_i in Construction 2 come from an encapsulation of K_i, we allow the construction to use distinct PRFs. Yet, one may choose $F_i = F_j$ for all i, j, if supported by the ingredient KEM's syntax.

Construction 2. *For all $i \in [1 \,..\, n]$ let $F_i \colon \mathcal{K}_i \times \mathcal{X} \to \mathcal{Y}$ be a function and let $\mathcal{K} = \mathcal{K}_1 \times \ldots \times \mathcal{K}_n$. We define the PRF-then-XOR composition of $F_1, \ldots, F_n$:*

$$[F_1 \,..\, F_n] \colon \mathcal{K} \times \mathcal{X} \to \mathcal{Y} \ , \quad [F_1 \,..\, F_n](k_1, \ldots, k_n, x) \coloneqq \bigoplus_{i=1}^{n} F_i(k_i, x) \ .$$

[7] If such an element does not exist the following bounds would only become tighter.

Lemma 8. *For all $i \in [1\,..\,n]$ let F_i be as in Construction 2. If all F_i are pseudorandom then $[F_1\,..\,F_n]$ is split-key pseudorandom.*

More precisely, for all $n, F_1, \ldots, F_n$, for all indices i and all adversaries $\mathcal{A}$ there exist an adversary $\mathcal{B}$ such that

$$\mathrm{Adv}^{\mathrm{pr}}_{[F_1\,..\,F_n],i}(\mathcal{A}) \leq \mathrm{Adv}^{\mathrm{pr}}_{F_i}(\mathcal{B}) \ .$$

Suppose that $\mathcal{A}$ poses at most q queries to its evaluation oracle. Then adversary $\mathcal{B}$ poses at most q queries to its own encapsulation oracle. The running times of $\mathcal{B}$ is roughly the same as of $\mathcal{A}$.

Proof. We fix an index $i \in [1\,..\,n]$ and we build an adversary $\mathcal{B}$ against the PRF F_i from an adversary $\mathcal{A}$ against the skPRF $[F_1\,..\,F_n]$.

Adversary $\mathcal{B}$ works as follows. It starts by running adversary $\mathcal{A}$. Each time that $\mathcal{A}$ queries the oracle Eval on input (k', x) it queries its own evaluation oracle on input x, obtaining the output $y \in \mathcal{Y}$. Then it computes the key $k := y \oplus \bigoplus_{j \neq i} F_j(k_j, x)$, and returns the key to $\mathcal{A}$. Finally, $\mathcal{B}$ returns the output of $\mathcal{A}$.

We observe that if $\mathcal{B}$ is playing against game PR^0 then it receives a real evaluation of F_i from the oracle Eval. Hence $\mathcal{B}$ returns to $\mathcal{A}$ a real key and $\mathcal{A}$ is playing against game PR^0_i. If $\mathcal{B}$ is playing against game PR^1 instead, then $\mathcal{B}$ receives independent, uniformly distributed values from the oracle Eval (note that, by the restrictions of game PR^1_i, adversary $\mathcal{A}$ queries its oracle on distinct input each time). If we add any constant value to $y \leftarrow_{\$} \mathcal{Y}$ the result remains uniformly distributed. Hence, on each query to Eval adversary $\mathcal{B}$ returns to $\mathcal{A}$ independent uniformly distributed keys and $\mathcal{A}$ is playing against game PR^1_i. □

5.2 Improving the PRF-then-XOR combiner

In Section 5.1 we studied the PRF-then-XOR split-key PRF $[F_1\,..\,F_n](k_1\,..\,k_n, x) = \bigoplus F_i(k_i, x)$ that outputs the XOR sum of (pseudorandom) function evaluations, the latter being on a common input but with individual keys. As it holds for all split-key PRFs, also this construction induces a KEM combiner that preserves CCA security. Fine.

In the following we study the core function $[\![F_1\,..\,F_n]\!]$ that is closely related to the above: There are still n individually keyed PRF evaluations, but each evaluation has an associated input that is shorter than in the PRF-then-XOR construction. The advantage is that processing shorter inputs generally takes less time, i.e., the construction is more efficient. The reason for stating it as a core function and not as a split-key PRF is that, as we will show, the construction is secure when used as the former but insecure when used as the latter.[8]

In the following we define the notation that will be used throughout the section. Let $\mathcal{K}$ be a finite session-key space. Consider some KEMs $\mathsf{K}_1, \ldots, \mathsf{K}_n$. For all $i \in [1\,..\,n]$, let $\mathcal{K}_i$ and $\mathcal{C}_i$ be the session-key space and the ciphertext space, respectively, of K_i, and let $\mathcal{C}^i = \mathcal{C}_1 \times \ldots \times \mathcal{C}_{i-1} \times \mathcal{C}_{i+1} \times \ldots \times \mathcal{C}_n$ and $\mathcal{K}^i = \mathcal{K}_1 \times \ldots \times \mathcal{K}_{i-1} \times \mathcal{K}_{i+1} \times \ldots \times \mathcal{K}_n$. Further, for all i let $F_i \colon \mathcal{K}_i \times \mathcal{C}^i \to \mathcal{K}$ be a (pseudorandom) function. Finally, let $\mathcal{C} = \mathcal{C}_1 \times \ldots \times \mathcal{C}_n$, $\mathcal{K}_* = \mathcal{K}_1 \times \ldots \times \mathcal{K}_n$, and define

$$[\![F_1\,..\,F_n]\!] \colon \mathcal{K}_* \times \mathcal{C} \to \mathcal{K}; \quad (k_1, \ldots, k_n, c_1\,..\,c_n) \mapsto \bigoplus_{i=1}^{n} F_i(k_i, c_1\,..\,c_{i-1}\,c_{i+1}\,..\,c_n) \ .$$

That is, the input to the ith PRF is the ciphertext vector with its ith component removed.

We conclude by proving two statements on the choice of $[\![F_1\,..\,F_n]\!]$ as a core function in the parallel combiner: First, $[\![F_1\,..\,F_n]\!]$ does not possess split-key pseudorandomness but, secondly, still yields a KEM combiner retaining CCA security. For the latter we crucially rely on the fact that ciphertexts of the CCA secure ingredient KEM are non-malleable as described before.

Lemma 9. *For $n \geq 2$ there exists an adversary $\mathcal{A}$ that makes 2 queries to* Eval *and has an advantage of $1 - 1/|\mathcal{K}|$ in breaking the split-key pseudorandomness of $[\![F_1\,..\,F_n]\!]$.*

Proof. Let $i \in [1\,..\,n]$ be any value. We arbitrarily fix some ciphertexts $c_j \in \mathcal{C}_j$ for $j \in [1\,..\,n] \setminus \{i\}$, $c_i, \bar{c}_i \in \mathcal{C}_i$ such that $c_i \neq \bar{c}_i$, and a list of keys $k' = k_1\,..\,k_{i-1}k_{i+1}\,..\,k_n \in \mathcal{K}^i$. In the following we also

[8] This makes evident that being split-key pseudorandom is a sufficient but not a necessary condition for a core function to induce a CCA combiner.

define $c^j = c_1 \,..\, c_{j-1} c_{j+1} \,..\, c_n \in \mathcal{C}^j$ for $j \in [1\,..\,n]$ and $\bar{c}^j$ as c^j with the ciphertext component c_i replaced by $\bar{c}_i$. In particular, $c^i = \bar{c}^i$. Moreover we define $c = c_1 \,..\, c_i \,..\, c_n \in \mathcal{C}$ and $\bar{c} = c_1 \,..\, \bar{c}_i \,..\, c_n \in \mathcal{C}$.

We build an adversary $\mathcal{A}$ to distinguish the games PR_i^b, $b \in \{0,1\}$, as follows. The adversary queries the oracle Eval on input (k', c) and $(k', \bar{c})$ obtaining respectively k and $\bar{k}$ as output. Then the adversary returns 1 if and only if $k \oplus \bigoplus_{j \neq i} F_j(k_j, c^j) = \bar{k} \oplus \bigoplus_{j \neq i} F_j(k_j, \bar{c}^j)$. Note that all values in the expression above are known to the adversary.

We compute the success probability of $\mathcal{A}$. We know that the keys $k, \bar{k}$ are either random, and thus the probability that the adversary outputs 1 is $1/|\mathcal{K}|$, or real. In the latter case, calling k_i the key implicitly used by the games PR_i^b, $b \in \{0,1\}$, we know that:

$$k = \bigoplus_{j \in [1\,..\,n]} F_j(k_j, c^j)\,; \quad \bar{k} = \bigoplus_{j \in [1\,..\,n]} F_j(k_j, \bar{c}^j) \quad .$$

From the equality $F_i(k_i, c^i) = F_i(k_i, \bar{c}^i)$ follows that $\mathcal{A}$ always outputs 1. □

Theorem 3. *Using $[\![F_1 \,..\, F_n]\!]$ as the core function in the parallel KEM combiner yields a CCA-secure KEM if at least one of the ingredient KEMs is CCA secure and $F_1, \ldots, F_n$ are pseudorandom.*

More precisely, for all $n, \mathsf{K}_1, \ldots, \mathsf{K}_n$, if $\mathsf{K} = \mathsf{K}_1 \parallel \ldots \parallel \mathsf{K}_n$ with core function $[\![F_1 \,..\, F_n]\!]$ then for all indices i and all adversaries $\mathcal{A}$ there exist adversaries $\mathcal{B}$ and $\mathcal{C}$ such that

$$\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}}(\mathcal{A}) \leq 2 \cdot \left(\mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}_i}(\mathcal{B}) + \mathrm{Adv}^{\mathrm{pr}}_{F_i}(\mathcal{C}) \right) \quad .$$

Moreover, if adversary $\mathcal{A}$ calls at most q_d times the oracle Dec*, then $\mathcal{B}$ calls at most q_d times the oracle* Dec*, and $\mathcal{C}$ calls at most $q_d + 1$ times the oracle* Eval*. The running times of $\mathcal{B}$ and $\mathcal{C}$ are roughly the same as that of $\mathcal{A}$.*

The proof of this statement follows very closely that of Theorem 1, with the PRF security games of F_i replacing the games PR_i^b of the skPRF. For completeness we provide a formal proof below.

Proof (Theorem 3). Let $\mathcal{A}$ denote an adversary attacking the CCA security of KEM K that issues at most q_d queries to the decapsulation oracle. We proceed with detailed descriptions of the games (see Figure 17) used in our proof.

Games G_0 **to** G_4		**Oracle** $\mathrm{Dec}(c)$	
00 $C^*, C_i^* \leftarrow \emptyset$; $L[\cdot] \leftarrow \perp$		16 If $c \in C^*$: Abort	
01 For $j \leftarrow 1$ to n:		17 If $L[c] \neq \perp$: Return $L[c]$	
02 $\quad (pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$		18 $c_1 \,..\, c_n \leftarrow c$	
03 $pk \leftarrow (pk_1, \ldots, pk_n)$		19 For $j \in [1\,..\,n] \setminus \{i\}$:	
04 $st \leftarrow_{\$} \mathcal{A}_1^{\mathrm{Dec}}(pk)$		20 $\quad k_j \leftarrow \mathsf{K.dec}_j(sk_j, c_j)$	
05 For $j \leftarrow 1$ to n:		21 $\quad$ If $k_j = \perp$: Return $\perp$	
06 $\quad (k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j(pk_j)$		22 $\quad y_j \leftarrow F_j(k_j, c_1 \,..\, c_{j-1} c_{j+1} \,..\, c_n)$	
07 $k_i^* \leftarrow_{\$} \mathcal{K}_i$	$\mid \mathrm{G}_1\text{-}\mathrm{G}_3$	23 If $c_i \in C_i$:	
08 For $j \leftarrow 1$ to n:		24 $\quad k_i \leftarrow k_i^*$	
09 $\quad y_j \leftarrow F_j(k_j^*, c_1^* \,..\, c_{j-1}^* c_{j+1}^* \,..\, c_n^*)$		25 Else:	
10 $y_i \leftarrow_{\$} \mathcal{K}$	$\mid \mathrm{G}_2\text{-}\mathrm{G}_4$	26 $\quad k_i \leftarrow \mathsf{K.dec}_i(sk_i, c_i)$	
11 $k^* \leftarrow y_1 \oplus \ldots \oplus y_n$		27 $\quad$ If $k_i = \perp$: Return $\perp$	
12 $C^* \leftarrow C^* \cup \{c^*\}$; $C_i^* \leftarrow C_i^* \cup \{c_i^*\}$		28 $y_i \leftarrow F_i(k_i, c_1 \,..\, c_{i-1} c_{i+1} \,..\, c_n)$	
13 $c^* \leftarrow c_1^* \,..\, c_n^*$		29 If $c_i \in \mathcal{C}_i$: $y_i \leftarrow_{\$} \mathcal{K}$	$\mid \mathrm{G}_2$
14 $b' \leftarrow_{\$} \mathcal{A}_2^{\mathrm{Dec}}(st, c^*, k^*)$		30 $L[c] \leftarrow y_1 \oplus \ldots \oplus y_n$	
15 Stop with b'		31 Return $L[c]$	

Fig. 17. Games G_0–G_4 as used in the proof of Theorem 3. Note that i is implicitly a parameter of all games above.

Game G_0 The KIND^0 game instantiated with the KEM K as given in Figure 4. Beyond that we made merely syntactical changes: In line 00 a set C_i^* and an array L are initialized as empty. In line 17 we check if the adversary has already queried the oracle for the same input and we return the same output. Lines 23 and 24 are added such that instead of using sk_i to decapsulate c_i^* the key k_i^* is used. Note that if line 24 is executed then key k_i^* is already defined, since $C_i^* \neq \emptyset$.

Claim 7. $\Pr[\mathrm{KIND}^0 \Rightarrow 1] = \Pr[\mathrm{G}_0 \Rightarrow 1]$.

This follows immediately from the correctness of K_i and the fact that the decapsulation algorithm is deterministic.

Game G_1 Line 07 is added to replace the key k_i^* with a uniform key from $\mathcal{K}_i$.

Claim 8. There exists an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against the session-key indistinguishability of K_i (see Figure 18) that issues at most q_d decapsulation queries such that

$$|\Pr[\mathrm{G}_0 \Rightarrow 1] - \Pr[\mathrm{G}_1 \Rightarrow 1]| \leq \mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}_i}(\mathcal{B}) \ ,$$

and the running time of $\mathcal{B}$ is roughly the running time of $\mathcal{A}$.

Proof. We construct $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ as given in Figure 18: Adversary $\mathcal{B}_1$ gets pk_i as input, and runs $(pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$ for all $j \in [1\,..\,n] \setminus \{i\}$ to instantiate the other KEMs (see lines 01–03). To answer the decapsulation queries of $\mathcal{A}_1$, $\mathcal{B}_1$ decapsulates all c_i for $j \neq i$ using sk_j (lines 18–20) and queries its own decapsulation oracle to decapsulate c_i (lines 24–26).

Adversary $\mathcal{B}_2$, run on the challenge (c_i^*, k_i^*), executes $(k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j$ for $j \neq i$ on its own (lines 07, 08). Then it computes the challenge session key k^* by, for all $j \in [1\,..\,n]$, evaluating the function F_j under the key k_j (lines 09–11) and runs $\mathcal{A}_2$ on $(c_1^*, \ldots, c_n^*, k^*)$ (line 14). Decryption queries are answered as in phase one unless $\mathcal{B}_2$ has to decapsulate c_i^* where it uses k_i^* instead (lines 22, 23). At the end $\mathcal{B}_2$ relays $\mathcal{A}_2$'s output and halts (line 15).

Adversary $\mathcal{B}_1^{\mathrm{Dec}}(pk_i)$
00 $C^*, C_i^* \leftarrow \emptyset$
01 For $j \in [1\,..\,n] \setminus \{i\}$:
02 $\quad (pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$
03 $pk \leftarrow (pk_1, \ldots, pk_n)$
04 $st \leftarrow_{\$} \mathcal{A}_1^{\mathrm{Dec}}(pk)$
05 $st' \leftarrow (st, pk, sk_1, \ldots, sk_{i-1}, sk_{i+1}, \ldots, sk_n)$
06 Return st'

Adversary $\mathcal{B}_2^{\mathrm{Dec}}(st', c_i^*, k_i^*)$
07 For $j \in [1\,..\,n] \setminus \{i\}$:
08 $\quad (k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j(pk_j)$
09 For $j \leftarrow 1$ to n:
10 $\quad y_j \leftarrow F_j(k_j^*, c_1^* \,..\, c_{j-1}^* c_{j+1}^* \,..\, c_n^*)$
11 $k^* \leftarrow y_1 \oplus \ldots \oplus y_n$
12 $C^* \leftarrow C^* \cup \{c^*\}$; $C_i^* \leftarrow C_i^* \cup \{c_i^*\}$
13 $c^* \leftarrow c_1^* \,..\, c_n^*$
14 $b' \leftarrow_{\$} \mathcal{A}_2^{\mathrm{Dec}}(st, c^*, k^*)$
15 Stop with b'

If $\mathcal{A}$ calls $\mathrm{Dec}(c)$:
16 If $c \in C^*$: Abort
17 $c_1 \,..\, c_n \leftarrow c$
18 For $j \in [1\,..\,n] \setminus \{i\}$:
19 $\quad k_j \leftarrow \mathsf{K.dec}_j(sk_j, c_j)$
20 $\quad$ If $k_j = \perp$: Return $\perp$
21 $\quad y_j \leftarrow F_j(k_j, c_1 \,..\, c_{j-1} c_{j+1} \,..\, c_n)$
22 If $c_i \in C_i^*$:
23 $\quad k_i \leftarrow k_i^*$
24 Else:
25 $\quad k_i \leftarrow \mathrm{Dec}(c_i)$
26 $\quad$ If $k_i = \perp$: Return $\perp$
27 $y_i \leftarrow F_i(k_i, c_1 \,..\, c_{i-1} c_{i+1} \,..\, c_n)$
28 $k^* \leftarrow y_1 \oplus \ldots \oplus y_n$
29 Return k

Fig. 18. Adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ against session-key indistinguishability of K_i from adversary $(\mathcal{A}_1, \mathcal{A}_2)$ against session-key indistinguishability of K.

Analysis Games G_0 and G_1 only differ on the key k_i^* used to compute k^* for $\mathcal{A}_2$, and, consequently, when answering $\mathcal{A}_2$'s decapsulation queries involving c_i^*. If $\mathcal{B}$ is run by the game KIND^0, that is, key k_i^* is a real key output of $\mathsf{K.enc}_i$, then $\mathcal{B}$ perfectly emulates game G_0. Otherwise, if $\mathcal{B}$ is run by the game KIND^1, and thus the key k_i^* is uniform, then $\mathcal{B}$ emulates G_1. Hence

$$\Pr[\mathrm{G}_0 \Rightarrow 1] = \Pr[\mathrm{KIND}^0 \Rightarrow 1]$$

and

$$\Pr[\mathrm{G}_1 \Rightarrow 1] = \Pr[\mathrm{KIND}^1 \Rightarrow 1] \ .$$

Lastly we observe that $\mathcal{B}$ issues at most as many decapsulation queries as $\mathcal{A}$. Our claim follows. □

Game G_2 We add line 10 and line 29. Thus if F_i is evaluated as a result of the challenge generation or a decapsulation query whose ith component is c_i^*, then the output y_i is overwritten with a uniform value from $\mathcal{K}$.

Claim 9. There exists an adversary $\mathcal{C}$ against the pseudorandomness of F_i that issues at most $q_d + 1$ evaluation queries such that

$$|\Pr[\mathrm{G}_1 \Rightarrow 1] - \Pr[\mathrm{G}_2 \Rightarrow 1]| \leq \mathrm{Adv}^{\mathrm{pr}}_{F_i}(\mathcal{C}) \ ,$$

and the running time of $\mathcal{C}$ is roughly the running time of $\mathcal{A}$.

Proof. We construct an adversary $\mathcal{C}$ that breaks the pseudorandomness of F_i if $\mathcal{A}$ distinguishes between games G_1 and G_2.

Adversary $\mathcal{C}$ runs $\mathsf{K.gen}_j$ for all $j \in [1 \,..\, n]$ to instantiate all KEMs (see lines 01–03). Then for each KEM K_j it generates a pair key-ciphertext (k_j^*, c_j^*) (lines 05 and 06). To generate the challenge the adversary evaluates normally each function F_j under the key k_j (lines 07 and 08) and then calls its own evaluation oracle to implicitly compute the function F_i (line 09). The results are then summed up to generate the challenge session key k^* (line 10). To answer the decapsulation queries of $\mathcal{A}$ on input $c_1 \,..\, c_n$, the adversary keeps track of previous decapsulation queries and returns the same result for two queries with the same input (line 16). $\mathcal{C}$ uses the secret keys it generated to decapsulate all ciphertext components c_j for $j \neq i$ (lines 18–20). The same procedure is used to decapsulate c_i if $c_i \neq c_i^*$; otherwise it queries its own decapsulation oracle (lines 21–26).

Adversary $\mathcal{C}^{\mathrm{Eval}}$	If $\mathcal{A}$ calls $\mathrm{Dec}(c)$:
00 $C^*, C_i^* \leftarrow \emptyset$; $L[\cdot] \leftarrow \perp$	15 If $c \in C^*$: Abort
01 For $j \leftarrow 1$ to n:	16 If $L[c] \neq \perp$: Return $L[c]$
02 $\quad (pk_j, sk_j) \leftarrow_{\$} \mathsf{K.gen}_j$	17 $c_1 \,..\, c_n \leftarrow c$
03 $pk \leftarrow (pk_1, \ldots, pk_n)$	18 For $j \in [1 \,..\, n] \setminus \{i\}$:
04 $st \leftarrow_{\$} \mathcal{A}_1^{\mathrm{Dec}}(pk)$	19 $\quad k_j \leftarrow \mathsf{K.dec}_j(sk_j, c_j)$
05 For $j \leftarrow 1$ to n:	20 $\quad$ If $k_j = \perp$: Return $\perp$
06 $\quad (k_j^*, c_j^*) \leftarrow_{\$} \mathsf{K.enc}_j(pk_j)$	21 If $c_i \in C_i^*$:
07 For $j \in [1 \,..\, n] \setminus \{i\}$:	22 $\quad y_i \leftarrow \mathrm{Eval}(c_1 \,..\, c_{i-1} c_{i+1} \,..\, c_n)$
08 $\quad y_j \leftarrow F_j(k_j^*, c_1^* \,..\, c_{j-1}^* c_{j+1}^* \,..\, c_n^*)$	23 Else:
09 $y_i \leftarrow \mathrm{Eval}(c_1^* \,..\, c_{i-1}^* c_{i+1}^* \,..\, c_n^*)$	24 $\quad k_i \leftarrow \mathsf{K.dec}_i(sk_i, c_i)$
10 $k^* \leftarrow y_1 \oplus \ldots \oplus y_n$	25 $\quad$ If $k_i = \perp$: Return $\perp$
11 $C^* \leftarrow C^* \cup \{c^*\}$; $C_i^* \leftarrow C_i^* \cup \{c_i^*\}$	26 $\quad y_i \leftarrow F_i(k_i, c_1 \,..\, c_{i-1} c_{i+1} \,..\, c_n)$
12 $c^* \leftarrow c_1^* \,..\, c_n^*$	27 $L[c] \leftarrow y_1 \oplus \ldots \oplus y_n$
13 $b' \leftarrow_{\$} \mathcal{A}_2^{\mathrm{Dec}}(st, c^*, k^*)$	28 Return $L[c]$
14 Stop with b'	

Fig. 19. Adversary $\mathcal{C}$ against pseudorandomness of F_i. At the start of the experiment the list L returns $\perp$ on any input.

Analysis First we note that by the conditions in lines 15 and 16 in Figure 19 all calls to Eval by $\mathcal{C}^b$ have different input and thus we can always use Eval to simulate F_i.

Observe that when $\mathcal{C}$ plays against PR^0 we are implicitly setting k_i^* as the key internally generated by PR^0. Hence $\mathcal{C}$ correctly simulates game G^1 to $\mathcal{A}$. Otherwise, when $\mathcal{C}$ plays against PR^1 the oracle Eval consistently outputs random elements in $\mathcal{K}$. Since y_i in line 22 is uniformly generated, so is $y_1 \oplus \ldots \oplus y_n$. Thus $\mathcal{C}$ correctly simulates game G_2 to $\mathcal{A}$. Therefore

$$\Pr[\mathrm{G}_1 \Rightarrow 1] = \Pr[\mathrm{PR}^0 \Rightarrow 1]$$

and

$$\Pr[\mathrm{G}_2 \Rightarrow 1] = \Pr[\mathrm{PR}^1 \Rightarrow 1]| \ .$$

Thus

$$|\Pr[\mathrm{G}_1 \Rightarrow 1] - \Pr[\mathrm{G}_2 \Rightarrow 1]| \leq \mathrm{Adv}^{\mathrm{pr}}_{F_i}(\mathcal{C}) \ .$$

We count the number of Eval queries by $\mathcal{C}$. From the definition of $\mathcal{C}$ we see that the oracle Eval is called once to generate the challenge. Further, for each Dec query by $\mathcal{A}$, $\mathcal{C}$ queries Eval at most once. □

Game G_3 We remove lines 29 to undo the modifications of the Dec oracle introduced in game G_2. Thus if the adversary asks for a decapsulation query on a ciphertext whose ith component is c_i^*, then the value y_i is computed by evaluating the function F_i instead of returning a uniform input.

Claim 10. There exists an adversary $\mathcal{C}'$ against the pseudorandomness security of F_i that issues at most q_d evaluation queries such that

$$|\Pr[\mathrm{G}_2 \Rightarrow 1] - \Pr[\mathrm{G}_3 \Rightarrow 1]| \leq \mathrm{Adv}^{\mathrm{pr}}_{F_i}(\mathcal{C}') \ ,$$

and the running time of $\mathcal{C}'$ is roughly the running time of $\mathcal{A}$.

Proof. Adversary $\mathcal{C}'$ is essentially the same as adversary $\mathcal{C}$ in Figure 19, with the exception that we replace line 10 with the generation of a uniform session key ($k^* \leftarrow_{\$} \mathcal{K}$). The proof analysis is the same as in Claim 9. Notice that since the challenge session key is uniform, this time $\mathcal{C}'$ calls Eval just q_d times. □

Note that, currently, the only difference from game G_1 is the addition of line 10, i.e., the challenge session key k^* is uniform.

Game G_4 Line 07 is removed to undo the modification introduced in game G_1. That is, we replace the uniform key k_i^* with a real key output by $\mathsf{K}.\mathsf{enc}_i(pk_i)$.

Claim 11. There exists an adversary $\mathcal{B}' = (\mathcal{B}'_1, \mathcal{B}'_2)$ against the session-key indistinguishability of K_i that issues at most q_d decapsulation queries such that

$$|\Pr[\mathrm{G}_3 \Rightarrow 1] - \Pr[\mathrm{G}_4 \Rightarrow 1]| \leq \mathrm{Adv}^{\mathrm{kind}}_{\mathsf{K}_i}(\mathcal{B}') \ ,$$

and the running time of $\mathcal{B}'$ is roughly the running time of $\mathcal{A}$.

Proof. Adversary $\mathcal{B}'$ is the same as adversary $\mathcal{B}$ in Figure 18, with the exception that we replace line 11 with the generation of a uniform session key ($k^* \leftarrow_{\$} \mathcal{K}$). The proof analysis is the same as in Claim 8. □

Claim 12. $\Pr[\mathrm{G}_4 \Rightarrow 1] = \Pr[\mathrm{KIND}^1 \Rightarrow 1]$.

This follows immediately from the correctness of K_i, the fact that the decapsulation algorithm is deterministic, and observing that since y_i is uniformly generated then so is k^*.

The proof of the main statement follows from collecting the statements from Claims 7 to 12. □

Note that the gain of evaluating any F_i on $n-1$ rather than n ciphertexts disappears for large n. However, in practice, where only a few KEMs shall be combined it might pose as a significant improvement.

Acknowledgments

We are grateful to the anonymous PKC reviewers for their valuable comments. Federico Giacon was supported by ERC Project ERCC (FP7/615074). Felix Heuer was supported by Mercator Research Center Ruhr project "LPN-Krypt: Das LPN-Problem in der Kryptographie". Bertram Poettering conducted part of this work at Ruhr University Bochum, supported by ERC Project ERCC (FP7/615074).

References

1. Ananth, P., Jain, A., Naor, M., Sahai, A., Yogev, E.: Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 491–520. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016)
2. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In: 2015 IEEE Symposium on Security and Privacy. pp. 553–570. IEEE Computer Society Press, San Jose, CA, USA (May 17–21, 2015)
3. Brzuska, C., Farshim, P., Mittelbach, A.: Random-oracle uninstantiability from indistinguishability obfuscation. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part II. LNCS, vol. 9015, pp. 428–455. Springer, Heidelberg, Germany, Warsaw, Poland (Mar 23–25, 2015)
4. Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J. Comput. 33(1), 167–226 (2003), https://doi.org/10.1137/S0097539702403773
5. Diffie, W., Hellman, M.E.: Special feature exhaustive cryptanalysis of the NBS Data Encryption Standard. Computer 10(6), 74–84 (Jun 1977), http://dx.doi.org/10.1109/C-M.1977.217750
6. Dodis, Y., Katz, J.: Chosen-ciphertext security of multiple encryption. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 188–209. Springer, Heidelberg, Germany, Cambridge, MA, USA (Feb 10–12, 2005)
7. Even, S., Goldreich, O.: On the power of cascade ciphers. ACM Trans. Comput. Syst. 3(2), 108–116 (1985), http://doi.acm.org/10.1145/214438.214442
8. Fischlin, M., Herzberg, A., Noon, H.B., Shulman, H.: Obfuscation combiners. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 521–550. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016)
9. Fischlin, M., Lehmann, A.: Security-amplifying combiners for collision-resistant hash functions. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 224–243. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2007)
10. Fischlin, M., Lehmann, A.: Multi-property preserving combiners for hash functions. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 375–392. Springer, Heidelberg, Germany, San Francisco, CA, USA (Mar 19–21, 2008)
11. Fischlin, M., Lehmann, A., Pietrzak, K.: Robust multi-property combiners for hash functions revisited. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 655–666. Springer, Heidelberg, Germany, Reykjavik, Iceland (Jul 7–11, 2008)
12. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of Cryptology 26(1), 80–101 (Jan 2013)
13. Harnik, D., Kilian, J., Naor, M., Reingold, O., Rosen, A.: On robust combiners for oblivious transfer and other primitives. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 96–113. Springer, Heidelberg, Germany, Aarhus, Denmark (May 22–26, 2005)
14. Herzberg, A.: On tolerant cryptographic constructions. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 172–190. Springer, Heidelberg, Germany, San Francisco, CA, USA (Feb 14–18, 2005)
15. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 341–371. Springer, Heidelberg, Germany, Baltimore, MD, USA (Nov 12–15, 2017)
16. Hohenberger, S., Lewko, A.B., Waters, B.: Detecting dangerous queries: A new approach for chosen ciphertext security. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 663–681. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012)
17. Manulis, M., Poettering, B., Stebila, D.: Plaintext awareness in identity-based key encapsulation. Int. J. Inf. Sec. 13(1), 25–49 (2014), https://doi.org/10.1007/s10207-013-0218-5
18. Merkle, R.C., Hellman, M.E.: On the security of multiple encryption. Commun. ACM 24(7), 465–467 (Jul 1981), http://doi.acm.org/10.1145/358699.358718
19. NIST: Post-Quantum Cryptography Standardization project. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography (2017)
20. Shannon, C.: Communication theory of secrecy systems. Bell System Technical Journal, Vol 28, pp. 656–715 (Oct 1949)
21. Zhang, C., Cash, D., Wang, X., Yu, X., Chow, S.S.M.: Combiners for chosen-ciphertext security. In: Dinh, T.N., Thai, M.T. (eds.) Computing and Combinatorics — 22nd International Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9797, pp. 257–268. Springer (2016), https://doi.org/10.1007/978-3-319-42634-1_21
22. Zhang, R., Hanaoka, G., Shikata, J., Imai, H.: On the security of multiple encryption or CCA-security+CCA-security=CCA-security? In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 360–374. Springer, Heidelberg, Germany, Singapore (Mar 1–4, 2004)