State-Separating Proofs: A Reduction Methodology for Real-World Protocols

Chris Brzuska[1,2], Antoine Delignat-Lavaud[3], Konrad Kohbrok[1], and Markulf Kohlweiss[3,4]

[1] Hamburg University of Technology
[2] Aalto University
[3] Microsoft Research Cambridge
[4] University of Edinburgh

Abstract. The security analysis of real-world protocols involves reduction steps that are conceptually simple but have to handle complicated protocol details. Taking inspiration from Universal Composability, Abstract Cryptography, π-calculus and F^*-based analysis we propose a new technique for writing simple reductions to avoid mistakes, have more self-contained concrete security statements, and allow the writer and reader to focus on the interesting steps of the proof.
Our method replaces a monolithic protocol game with a collection of so-called *packages* that are composed by calling each other. Every component scheme is replaced by a package parameterized by a bit b. Packages with a bit 0 behave like a concrete scheme, while packages with a bit 1 behave like an ideal version. The indistinguishability of the two packages captures security properties, such as the concrete pseudo-random function being indistinguishable from a random function.
In a security proof of a real-life protocol, one then needs to flip a package's bit from 0 to 1. To do so, in our methodology, we consider all other packages as the reduction. We facilitate the concise description of such reductions by a number of π-calculus-style algebraic operations on packages that are justified by state separation and interface restrictions. Our proof method handles "simple" steps using algebraic rules and leaves "interesting" steps to existing game-based proof techniques such as, e.g., the code-based analysis suggested by Bellare and Rogaway.
In addition to making simple reductions more precise, we apply our techniques to two composition proofs: a proof of self-composition using a hybrid argument, and the composition of key generating and key consuming components. Consistent with our algebraic style the proofs are generic. For concreteness, we apply them to the KEM-DEM proof of hybrid-encryption by Cramer and Shoup and the proof for composability of Bellare-Rogaway secure key exchange protocols with symmetric-key protocols.

1 Introduction

The proof methodology that we propose is influenced and inspired by important conceptual works from several areas. In particular, we would

like to acknowledge the influences of Canetti's Universal Composability framework (UC) [15], Renner's and Maurer's work on random systems, abstract cryptography and constructive cryptography [34,31,33,32], the π-calculus [35] and miTLS, the $F^\star$-verified analysis of TLS [22,9,10,19]. We discuss those influences in detail in Subsection 1.3, after laying some technical groundwork. The goal of the introduction is to explain how the practice of writing security reductions for real-life protocols changes when adopting the proof methodology proposed in this paper. We start with a simple example.

1.1 Example

Based on a pseudorandom function (PRF), we construct a symmetric encryption scheme that is indistinguishable under chosen plaintext attacks (IND-CPA). The goal of this example is to showcase the usefulness of *associativity* of algorithm composition for the writing of reductions. We will write the IND-CPA game in a modular way that makes the game-hop which replaces the PRF with a random function immediate and thereby emphasizes aspects of the security analysis that are more interesting. Readers closely familiar with miTLS, abstract cryptography or π-calculus might be aware of this modular writing style that exploits associativity of algorithm composition and might prefer to skip ahead to Section 1.2.

As is good cryptographic practice, we proceed as follows:

(1) Specification of security goal: IND-CPA secure symmetric encryption.
(2) Specification of cryptographic assumptions: PRF security.
(3) Construction: We build a symmetric encryption scheme from a PRF.
(4) Reduction: Prove that if the assumption is true, then our construction satisfies the security goal. This involves a reduction, that simulates the IND-CPA game of the security goal (instantiated with the construction) given oracle access to the PRF game of the assumption.

(1) IND-CPA security. In the real-or-ideal formalization of IND-CPA security, the adversary has adaptive access to an encryption oracle ENC to which they can submit one message m at a time. The adversary either receives an encryption of m, or an encryption of a random string of the same length as m. The adversary then needs to distinguish whether they receive encryptions of real messages or of random messages.[5] Note that we

[5] Note that this definition of IND-CPA security is equivalent (by a factor of 2) to the standard left-or-right IND-CPA security definition, where the adversary submits two messages and either receives an encryption of the left message or an encryption of

operate in the concrete security setting as it is more adequate for practice-oriented cryptography and therefore only define advantages rather than security in line with the critique of Rogaway, Bernstein and Lange [39,8]. Our ideas can be transferred analogously to the asymptotic setting.

We denote the interaction of the adversary with the encryption oracle as $\mathcal{A} \circ \mathsf{ENC}$ instead of the common notation $\mathcal{A}^{\mathsf{ENC}}$ (which will reveal its convenience shortly). Moreover, we will write the name of the game instead of the oracle name: $\mathcal{A} \circ \texttt{IND-CPA}^b$. This convention is inessential for the current example; it is useful in more complex settings.

Definition 1 (IND-CPA Security). *Let $\zeta = (\zeta.kgen, \zeta.enc, \zeta.dec)$ be a symmetric encryption scheme. We define the IND-CPA advantage $\epsilon_\zeta(\mathcal{A})$ of an adversary $\mathcal{A}$ as*

$$2 \cdot \left| \Pr\left[1 \leftarrow\!\!\$\, \mathcal{A} \circ \texttt{IND-CPA}^0\right] - \Pr\left[1 \leftarrow\!\!\$\, \mathcal{A} \circ \texttt{IND-CPA}^1\right] \right|.$$

We consider ϵ_ζ as a function of the adversary and write, equivalently,

$$\texttt{IND-CPA}^0 \overset{\epsilon_\zeta}{\approx} \texttt{IND-CPA}^1.$$

The two games $\texttt{IND-CPA}^0$ and $\texttt{IND-CPA}^1$ are specified in the right-most column of Figure 6, Appendix A (We give an equivalent game shortly.).

(2) PRF security. For prf a pseudorandom function (PRF) with key length n, we define the security game of the prf where the adversary's task is to distinguish between (a) the game $\texttt{PRF}^0$ where the adversary has access either to a real prf evaluation oracle EVAL and (b) the game $\texttt{PRF}^1$ where the adversary has access to an evaluation oracle EVAL that implements a random function. For disambiguation based on the secret bit b, we write $\texttt{PRF}^b.\mathsf{EVAL}$ for the respective oracles.

$\texttt{PRF}^0.\mathsf{EVAL}(x)$	$\texttt{PRF}^1.\mathsf{EVAL}(x)$
if $k = \bot$ **then**	**if** $T[x] = \bot$ **then**
$\quad k \leftarrow\!\!\$\, \{0,1\}^n$	$\quad T[x] \leftarrow\!\!\$\, \{0,1\}^n$
$y \leftarrow \mathsf{prf}(k, x)$	$y \leftarrow T[x]$
return y	**return** y

Definition 2 (PRF Security). *Let prf be a pseudorandom function with key length n. For an adversary $\mathcal{A}$, the PRF distinguishing advantage of $\texttt{PRF}^0 \overset{\epsilon_{\texttt{PRF}}}{\approx} \texttt{PRF}^1$ is $\epsilon_{\texttt{PRF}}(\mathcal{A})$.*

(3) Construction. We construct a symmetric encryption scheme $\zeta = (kgen, enc, dec)$ out of the PRF prf, see Figure 1.

the right message. Our choice of definition is not essential here. We merely prefer real-or-ideal games, because ideal functionalities tend to ease composition.

$\zeta.kgen$	$\zeta.enc(k, m)$	$\zeta.dec(k, (r, c))$
$k \leftarrow\!\!\$\ \{0,1\}^n$	$r \leftarrow\!\!\$\ \{0,1\}^n$	
return k	$pad \leftarrow \mathsf{prf}(k, r)$	$pad \leftarrow \mathsf{prf}(k, r)$
	$c \leftarrow m \oplus pad$	$m \leftarrow c \oplus pad$
	return (r, c)	**return** m

Fig. 1: Construction of the IND-CPA secure encryption scheme ζ from the pseudorandom function prf. For simplicity of exposition, we assume that k, r, and pad, m, c are all of equal length n.

(4) Reduction. We reduce the IND-CPA security of the encryption scheme ζ to the PRF security of prf. Towards this goal, we modularize the security game. The package $\texttt{MOD-CPA}^b$ (see Figure on the right) uses an EVAL oracle and provides an encryption oracle ENC. When $\texttt{MOD-CPA}^b$ is composed with $\texttt{PRF}^0$, then for both $b \in \{0,1\}$, the package $\texttt{MOD-CPA}^b \circ \texttt{PRF}^0$ is functionally equivalent to $\texttt{IND-CPA}^b$. The comparison is straightforward via inlining, see Appendix A.

$\texttt{MOD-CPA}^b.\mathsf{ENC}(m)$

if $b = 0$ **then**
  $r \leftarrow\!\!\$\ \{0,1\}^n$
  $pad \leftarrow \mathsf{EVAL}(r)$
  $c \leftarrow m \oplus pad$
if $b = 1$ **then**
  $m' \leftarrow\!\!\$\ \{0,1\}^n$
  $r \leftarrow\!\!\$\ \{0,1\}^n$
  $pad \leftarrow \mathsf{EVAL}(r)$
  $c \leftarrow m' \oplus pad$
return (r, c)

Let $\mathcal{A}$ be an adversary. In the following gamehops, note that the prf advantage appears twice, as the games $\texttt{IND-CPA}^0$ and $\texttt{IND-CPA}^1$ both use $\zeta.enc$ and thus employ the actual PRF and not a random function.

$$
\begin{aligned}
& \mathcal{A} \circ \texttt{IND-CPA}^0 \\
= \;& \mathcal{A} \circ \texttt{MOD-CPA}^0 \circ \texttt{PRF}^0 && \text{(Functional equivalence)} \\
= \;& (\mathcal{A} \circ \texttt{MOD-CPA}^0) \circ \texttt{PRF}^0 && \text{(Associativity)} \\
\overset{\epsilon_1(\mathcal{A})}{\approx}\; & (\mathcal{A} \circ \texttt{MOD-CPA}^0) \circ \texttt{PRF}^1 && \text{(PRF security, } \epsilon_1(\mathcal{A}) = \epsilon_{\texttt{PRF}}(\mathcal{A} \circ \texttt{MOD-CPA}^0)\text{)} \\
= \;& \mathcal{A} \circ \texttt{MOD-CPA}^0 \circ \texttt{PRF}^1 && \text{(Associativity)} \\
\overset{\epsilon_2(\mathcal{A})}{\approx}\; & \mathcal{A} \circ \texttt{MOD-CPA}^1 \circ \texttt{PRF}^1 && \text{(Interesting step)} \\
= \;& (\mathcal{A} \circ \texttt{MOD-CPA}^1) \circ \texttt{PRF}^1 && \text{(Associativity)} \\
\overset{\epsilon_3(\mathcal{A})}{\approx}\; & (\mathcal{A} \circ \texttt{MOD-CPA}^1) \circ \texttt{PRF}^0 && \text{(PRF security, } \epsilon_3(\mathcal{A}) = \epsilon_{\texttt{PRF}}(\mathcal{A} \circ \texttt{MOD-CPA}^1)\text{)} \\
= \;& \mathcal{A} \circ \texttt{MOD-CPA}^1 \circ \texttt{PRF}^0 && \text{(Associativity)} \\
= \;& \mathcal{A} \circ \texttt{IND-CPA}^1 && \text{(Functional equivalence)}
\end{aligned}
$$

The first and last transformation follow by functional equivalence (See Figure 6, Appendix A) and are proven via inlining the implementation of the corresponding oracles. The PRF assumption and associativity of algorithm composition cover all other steps, except for the one labeled *interesting step*, on which we focus now.

Indeed, it is the only part of the proof that might contain anything of conceptual interest. In the interesting step, the game moves from encrypting the adversary's message to encryting a random message. In both cases, the padding is created via a random function. Are the ciphertext distributions that the adversary sees identical in both cases? That is, does $\epsilon_2(\mathcal{A})$ equal 0, or is there some loss?

It turns out that the ciphertext distributions differ whenever there is a collision on the randomness r. In that case, the padding is repeated and therefore, if $b = 0$, xoring the two ciphertexts is equal to the xor of the two messages that the adversary queried while if $b = 1$, then with overwhelming probability, this is not the case. Therefore, in the interesting step, we need to perform a bad event analysis. Let $q_{\mathcal{A}}$ be an upper bound on the number of adversarial queries; by the birthday bound, the probability of the bad event is at most ${q_{\mathcal{A}}}^2/2^{n-1}$ and thus, $\epsilon_2(\mathcal{A}) \leq {q_{\mathcal{A}}}^2/2^{n-1}$.

Our suggested writing style splits reduction proofs into different kinds of steps. Simple steps such as functional equivalence, associativity and using the assumption are carried out separately and algebraically and allow to make the reduction explicit and precise, first as $\texttt{MOD-CPA}^0$ then $\texttt{MOD-CPA}^1$. The second category of steps are interesting steps that can potentially hide subtleties.

1.2 Conceptual Insights

The methodology sketched in the previous example relied on *functional equivalence* and *associativity* of algorithm composition. Moreover, the modularization of the proof relied on *state separation*. That is, in the modularization, we separated the state of the PRF from the state of the encryption scheme that used the PRF. All other technical insights in this paper are guided by the following question:

Which state do cryptographic games share?

Multi-instance games From the viewpoint of state-sharing, one can quickly observe why some multi-instance-to-single-instance reductions are straightforward while others are not: The easy case is the one where the multi-instance game is merely a concatenation of several state-disjoint copies of

the single-instance primitive game, i.e., at most, all copies share a secret bit b. We will see in Subsection 4.1 that indeed, the reduction from such multi-instance games to single-instance games only involves simple steps (i.e., standard transformations that can be specified precisely and do not require any arguing). Avoiding multi-to-single instance reductions is one of the motivations of composition frameworks (See Section 1.3 for a more detailed discussion), so we see it as a sanity check that our modular proof methodology indeed captures those proofs as simple. Note that also in the game-based setting, general multi-instance to single-instance reductions for classes of games have been provided before (see, e.g., Bellare, Boldyreva and Micali [5]).

Key composition. In turn, game-based composition becomes highly non-trivial when it comes to key composition. Key composition occurs, essentially, anytime when one cryptographic building block generates a string that is used as a key by another building block. Prominent examples here are composition of key encapsulation mechanisms (KEM) with deterministic encryption mechanisms (DEM) (see Cramer and Shoup [17]) as well as composing Bellare-Rogaway secure key exchange protocols with symmetric-key based protocols such as secure channels (see Brzuska, Fischlin, Warinschi, Williams [13,11]). We observe that the difficulty of key composition proofs stems, essentially, from the *sharing of state* between two cryptographic objects with independent security definitions: the key *generating* game and the key *consuming* game. On a high-level, it is clear that first, one proceeds by showing that the keys emerging from the key generating game can be replaced by random keys and then, one can apply security of the key consuming game. Technically, however, the pseudo-code line that generates the key needs to be moved from the key generating game to the key consuming game, which is technically annoying.

We thus propose to move that line (plus some technicalities) into a special `KEY` *package*. In the proof of our `KEY` composition lemma, we first consider the `KEY` package part of the key generating game to move from real keys to random keys. In the second step, we can then consider the `KEY` package as part of the primitive that uses the key and reduce to its security. Our algebraic rules allow to shift the `KEY` package into the appropriate location.

Applications. Using our `KEY` composition lemma, we rewrite the CCA-secure KEM-DEM composition proof by Cramer and Shoup [17] whose original proof was written mostly in prose. Here, the application of our lemma is preceded by a functional equivalence step and then, the lemma

can be directly applied. Our proof is more precise than the proof by Cramer and Shoup, but our proof is not shorter, when one takes into account the functional equivalence step. The main observation here is that only one functional equivalence step is needed, although there are two reductions. For larger real-world protocols, also only one large functional equivalence step is needed, although there are several reductions.

As as second application, we capture the original composition proof for composing Bellare-Rogaway forward-secure key exchange protocols with single-session reducible symmetric-key protocols by Brzuska, Fischlin, Warinschi and Williams (BFWW) [13]. Note that our composition theorem only uses algebraic rules. We hope that thereby, we clarify the original composition proof and make it more precise by giving concise descriptions of all reductions. Concretely, we prove the composition theorem using a multi-instance version of the key composition lemma.

Scope of our method. Our method considers distinguishing games for *single-stage* adversaries [38], that is, we do not consider games where the adversary is split into two algorithms whose communication/state-sharing is restricted. Although suitable extensions might exist (e.g., by extending adversaries into packages that can call each other), this is an actual restriction of our current method. Another seeming – but not an actual – restriction is that we encode all security properties as indistinguishability between two games. Search problems such as strong unforgeability can also be encoded via indistinguishability. While the encoding might seem surprising when not used to it, at a second thought, an appropriate encoding of an unforgeability game also simplifies game-hopping: Imagine that we insert an abort condition whenever a message is accepted by verification that was not signed by the signer. This step actually corresponds to idealizing the verification of the signature scheme so that it only accepts messages that were actually signed before. Finally, our method does not consider recursions, i.e. oracles that call themselves, or oracles that contain infinite loops.

We now discuss related ideas and then introduce our proof methodology.

1.3 Related Techniques

Our perspective is on improving the (hand-written) security proofs of real-world protocols [26,29,20,16]. Such proofs typically contain large reductions relating a complex monolithic game to each cryptographic assumption through an intricate simulation of the protocol.

To manage the complexity we were inspired by modular techniques from both the field of cryptography and programming.

Cryptography. Modular a.k.a. composable proofs in the pen-and-paper world as pioneered by Backes, Pfitzmann, Waidner, and Canetti have a 17-year long history full of rich ideas [1,15,30,37,23,24,43], such as the idea of an environment that cannot distinguish a real protocol from an ideal variant with strong security guarantees. This might be obvious in retrospect, but is foundational for any compositional approach.

Likewise, Maurer's and Renner's work on random systems, abstract cryptography and constructive cryptography [34,31,33,32] inspired and encouraged our view that a more abstract and algebraic approach to cryptographic proofs is possible and desirable. Indeed several of our concepts have close constructive cryptography analogues:

A core idea of the proof writing technique presented in this paper is derived from the associativity of function and algorithm composition. This has been termed composition-order independence in Maurer's frameworks [32].

The idea of associativity and parallel composition relates to the idea of cryptographic algebras. An ambitious expression of the idea is found in Section 6.2 of [33]. Abstract cryptography has an associativity law and neutral element for serial composition and an interchange law for parallel composition. This line of work [33,32] also introduces a distinguishing advantage between composed systems and makes use of transformations that move part of the system being considered into and out of the distinguisher. Our focus is not on definitions but on writing proofs and we thus employ these techniques in game-based security proofs. As such we are also influenced by work on game-based composition by Brzuska, Fischlin, Warinschi, and Williams [13].

Programming. Algebraic reasoning is also at the core of process calculi (or process algebras) such as the π-calculus by Milner, Parrow and Walker [35]. The primary focus of such calculi is on the modelling of concurrency using non-determinism, but research inspired by probabilistic process algebras has applied these techniques to computational cryptography by Mitchell, Ramanathan, Scedrov and Teague [36] and Barthe, Crespo, Lakhnech and Schmidt [3].

The associativity of monadic composition, a generalisation of function composition to effectful programs, is an important structuring principle of functional programming languages such as Haskell, $\mathrm{F}^\sharp$ and F^* [27,41,40]. Our concept of packages for which we defined the associative composition

operator ∘ is heavily inspired by modules in programming languages such as $F^\sharp$, OCaml, and ML, see, e.g., Tofte [42]. Our oracles can be considered stateful functions that operate on the local state of a package. It is the same state-separation that enables composition in cryptographic frameworks.

Existing techniques for overcoming the crisis of rigour in provable security as formalised by Bellare and Rogaway [7] and mechanised in Easycrypt [4] have focused on the most intricate aspects of proofs. While Easycrypt has a module system as found in functional programming languages [2], it has not been used to simplify reasoning about large reductions in real-world protocols.

We make use of packages and algebraic reasoning to give concrete reductions with concise descriptions. For example, jumping ahead, in Section 2.1 (see especially Figure 2), we first use the interchange rule to move the key consuming package into the reduction to the security assumption of the key generating package, and then, we do the converse. I.e., we use the interchange rule to move the key generating package into the reduction to the security assumption of the key consuming package.

The closest to our idea of packaged reductions is the modular structure of miTLS, an $F^\star$-verified implementation of TLS [22,9,10,19]. The insight of Fournet, Kohlweiss and Strub [22] was that code-based game rewriting could be done on actual code, one module at a time, with the rest of the program becoming the reduction for distinguishing the *ideal* from the *real* version of the module. In this paper we move in the other direction back to the pen-and-paper world.

2 Proof Methodology

The PRF and IND-CPA games considered in the introduction are very simple, because each game only has a single oracle. However, many games have several oracles, e.g., the security game for indistinguishability under chosen ciphertext attacks (IND-CCA) provides an encryption oracle and a decryption oracle to the adversary. In cryptographic literature, we use the words *game* and *oracle* rather loosely. In this paper, we want to give those terms precise meaning. A game is an interactive, stateful algorithm and the oracles a.k.a. queries specify how the game updates its state and which output is returned to the adversary. In fact, sometimes, oracle calls are made within the game. For example, consider the IND-CCA encryption game in the random oracle model: Here, the game provides an encryption oracle, a decryption oracle and a random oracle to the adversary. When

the adversary makes a query to the encryption oracle, then the encryption oracle makes an internal query to the random oracle. To summarize, a game provides several oracles that can also internally call each other and that operate on a joint state.

We now introduce the general definition of *packages* that subsumes adversaries, games and reductions. Let us focus on reductions for a moment, what is the difference between a reduction and a game? On the one hand, reductions and games are quite similar, because they both provide oracles to the adversary. However, the game's oracles only make queries to internal oracles of the game. In turn, the reduction also makes queries to the external game. That is, a reduction has *two* sides, so to speak, an *output* side where the reductions answers queries from the adversary, and an *input* side where the reduction makes queries to the game. Therefore, packages too have two sides. A game will then just be a special case of a package where the input side is empty.

We begin by introducing oracles formally.

Definition 3 (Oracle). *An oracle* O *is an interactive stateful algorithm that operates on some state* Σ*, a set of parameters* Π *and has a name* $\mathsf{O.name}$*. If another algorithm interacts with an oracle, we say the algorithm "calls" or "queries" the oracle. When calling the oracle, the caller can provide the oracle with an input. Upon termination, the oracle will return some value.*

We will usually write O to denote both the oracle and its name $\mathsf{O.name}$, unless context requires further clarification.

A package consists of a set of oracles and an internal state, on which all (and only) the oracles in the set operate. When an oracle O is contained in a package's set of oracles, we say that the package *contains* O. If a package $\mathtt{M}$ contains oracle O, we sometimes address the oracle with $\mathtt{M}.\mathsf{O}$. Additionally, every package has an *input* interface and an *output* interface, where each interface is a set of oracle names. The *output* interface is a subset of the set of names of oracles contained in the package and corresponds to the oracles that can be called from the outside of the package, e.g., by an adversary. When an oracle name is part of the output interface, then we say that the package *provides* the oracle with that name. In turn, the *input* interface contains a name list of oracles that are external to the package itself and that the package can makes calls to.

Definition 4 (Package). *A package* $\mathtt{M}$ *consists of a set of oracles* Ω*, some internal state* Σ *and a set of parameters* Π*, as well as an input*

interface $\mathsf{in}(\mathtt{M})$ *and an output interface* $\mathsf{out}(\mathtt{M})$*, which are sets of oracle names. For all oracles* $\mathsf{O} \in \Omega$*, the following holds*

- O *can access parameters* Π *and only operate directly on state in* Σ*.*
- O *can only be called by oracles* $\mathsf{O}' \notin \Omega$*, if* $\mathsf{O}.\mathsf{name} \in \mathsf{out}(\mathtt{M})$*.*
- O *can make calls to oracles* O'*iff* $\mathsf{O}'.\mathsf{name} \in \mathsf{in}(\mathtt{M}) \land \mathsf{O}' \notin \Omega$*.*

Note, that we do not allow "internal" calls, where for some package $\mathtt{M}$ with oracles Ω, an oracle $\mathsf{O} \in \Omega$ calls another oracle $\mathsf{O}' \in \Omega$. This is without loss of generality, as we can inline the code of O' in the places where O would do the calls.

Parameters and state. State and parameters of a package can be accessed by all oracles contained in the package. We annotate the set of parameters Π of a package $\mathtt{M}$ in superscript: $\mathtt{M}^{\Pi}$. Note that $\mathtt{M}^{\Pi}$ and $\mathtt{M}^{\Pi'}$ are different packages if $\Pi \neq \Pi'$ and that different packages have disjoint state and oracles. The state Σ of a package contains all variables that are used by any of the oracles, and the variables are initialized generically: Sets are initialized as $\emptyset$, arrays are initially empty. Variables of any other kind are initialized as $\bot$. Value $\bot$ indicates an uninitialized value or error.

Oracles. An oracle is bound to the state and the parameters it operates on. I.e., two different packages $\mathtt{M}, \mathtt{M}'$ can contain oracles O, O' with the same functional description, yet operating on different states and parameters.

We can model the PRF game from the introduction as a package $\mathtt{PRF}^b$, $b \in \{0, 1\}$, that contains the oracle EVAL. As the EVAL oracle can be queried from outside the package (e.g., by an adversary), we have $\mathsf{out}(\mathtt{PRF}^b) = \{\mathsf{EVAL}\}$ and as EVAL makes no queries to other oracles, we have $\mathsf{in}(\mathtt{PRF}^b) = \emptyset$. Being a common object in cryptography, we call a package with an empty input interface a *game*:

Definition 5 (Game). *Let* $\mathtt{M}$ *be a package;* $\mathtt{M}$ *is a closed package or game, if* $\mathsf{in}(\mathtt{M}) = \emptyset$*.*

Package composition. Let us revisit the IND-CPA game from the introduction. The package $\mathtt{MOD\text{-}CPA}^b$ contains an ENC oracle and has interfaces $\mathsf{in}(\mathtt{MOD\text{-}CPA}^b) = \{\mathsf{EVAL}\}$ and $\mathsf{out}(\mathtt{MOD\text{-}CPA}^b) = \{\mathsf{ENC}\}$. We obtain a *game* or *closed package* by composing $\mathtt{MOD\text{-}CPA}^b$ with $\mathtt{PRF}^0$. We write the aforementioned composition as $\mathtt{MOD\text{-}CPA}^b \circ \mathtt{PRF}^0$, using the operator $\circ$ which is more convenient for us than the superscript notation (i.e., $\mathtt{MOD\text{-}CPA}^{b^{\mathtt{PRF}^0}}$). We say that $\mathtt{M}$ *matches* the output interface of $\mathtt{N}$ iff $\mathsf{in}(\mathtt{M}) \subseteq \mathsf{out}(\mathtt{N})$.

Definition 6 (Circle Operator). *Let* $\mathtt{M}$ *and* $\mathtt{M}'$ *be two packages with states* Σ, Σ'*, oracle sets* Ω, Ω'*, and* $\mathsf{in}(\mathtt{M}) \subseteq \mathsf{out}(\mathtt{M}')$*. Then we call* $\mathtt{P} := \mathtt{M} \circ \mathtt{M}'$ *the package composition of* $\mathtt{M}$ *and* $\mathtt{M}'$ *with state* $\Sigma_{\mathtt{P}} = \Sigma \uplus \Sigma'$*,* $\mathsf{in}(\mathtt{P}) = \mathsf{in}(\mathtt{M}')$ *and* $\mathsf{out}(\mathtt{P}) = \mathsf{out}(\mathtt{M})$*. We define* $\Omega_{\mathtt{P}} := \Omega$*, where for all calls from oracles* $\mathsf{O} \in \Omega$ *to oracles* $\mathsf{O}' \in \Omega'$ *we replace the call with the oracle code from* O'*.*

Inlining. Our pseudo-code language only has precise but not formal semantics (in the sense of formal programming languages) (yet). We consider a language without recursion or infinite loops. Note that when inlining, formally, we need to prefix all variable names with the name of their original package to avoid variable collisions. For example a variable v of a package $\mathtt{M}$, will be renamed to $\mathtt{M}.v$. If no collisions are present, we skip the renaming to ease readability.

State of composed packages. When composing two packages such that $\mathtt{P} = \mathtt{M} \circ \mathtt{M}'$, the state of packages $\mathtt{M}$ and $\mathtt{M}'$ is merged and can not be addressed individually anymore. Instead, we now only have the state of $\mathtt{P}$, which can be modified through calls to oracles in $\mathsf{in}(\mathtt{P})$. Note, however, that internally to package $\mathtt{P}$, one can still conceive of the different parts of the state of $\mathtt{P}$ and see that those parts only influence each other by what would have been an oracle call before composition. We refer to this property as *state separation*.

Uniqueness. Note, that we can't use the same package twice when describing a package composition, i.e., it is not possible to have compositions such as $(\mathtt{M} \circ \mathtt{M'} \circ \mathtt{M})$. In particular, the state of such an expression would not be well-defined, and the concept of state separation would be very unclear if repetitions were allowed. This would be similar to copies of pointers to the same state (a.k.a. aliases).

Lemma 1 (Associativity). *Package composition is associative, i.e., for three package* $\mathcal{A}$*,* M *and* N *with* $\mathsf{in}(\mathtt{M}) \subseteq \mathsf{out}(\mathtt{N})$ *and* $\mathsf{in}(\mathcal{A}) \subseteq \mathsf{out}(\mathtt{M})$*, we have that* $(\mathcal{A} \circ \mathtt{M}) \circ \mathtt{N} = \mathcal{A} \circ (\mathtt{M} \circ \mathtt{N})$*.*

Thus, the package $\mathtt{P} := \mathcal{A} \circ \mathtt{M} \circ \mathtt{N}$ is well-defined and its interfaces are $\mathsf{in}(\mathtt{P}) = \mathsf{in}(\mathtt{N})$ and $\mathsf{out}(\mathtt{P}) = \mathsf{out}(\mathcal{A})$.

Proof sketch. Associativity holds because one can first inline procedures of $\mathtt{M}$ in $\mathcal{A}$, (this corresponds to $\mathcal{A} \circ \mathtt{M}$), and then inline procedures of $\mathtt{N}$ in $(\mathcal{A} \circ \mathtt{M})$, or one can inline procedures of $\mathtt{N}$ in $\mathtt{M}$ (this corresponds to $\mathtt{M} \circ \mathtt{N}$), and then inline the resulting procedures in $\mathcal{A}$. In both cases we obtain

exactly the same program text. When we fix formal semantics, then this argument becomes formal.

Adversaries. After defining the traditional cryptographic game in terms of packages (or the composition of such), we now turn to adversaries. Traditionally, an adversary $\mathcal{A}$ is an algorithm that interacts with a set of oracles with the goal of achieving a certain winning condition. In this paper, we only consider distinguishing adversary that return a bit 0 or 1. We model the adversary as a package whose input interface is equal to the set of names of the oracles of the game that the adversary is meant to interact with. In turn, the output interface of the adversary only contains the oracle RUN and returns the guess of the adversary.

Definition 7 (Adversary). *We call a package $\mathcal{A}$ an adversary against a game $\mathtt{G}$, if $\mathsf{in}(\mathcal{A}) = \mathsf{out}(\mathtt{G})$ and $\mathsf{out}(\mathcal{A}) = \{\mathsf{RUN}\}$.*

Definition 8 (Distinguishing Advantage). *For two games $\mathtt{Game}^0$ and $\mathtt{Game}^1$ with the same output interface $\mathsf{out}(\mathtt{Game}^0) = \mathsf{out}(\mathtt{Game}^1)$ and an adversary $\mathcal{A}$ with $\mathsf{in}(\mathcal{A}) = \mathsf{out}(\mathtt{Game}^0)$, the distinguishing advantage $\epsilon_{\mathtt{Game}}(\mathcal{A})$ of $\mathtt{Game}^0 \overset{\epsilon_{\mathtt{Game}}}{\approx} \mathtt{Game}^1$ is equal to $2 \cdot \left|\Pr\left[1 \leftarrow_{\$} \mathcal{A} \circ \mathtt{Game}^0\right] - \Pr\left[1 \leftarrow_{\$} \mathcal{A} \circ \mathtt{Game}^1\right]\right|$.*

Lemma 2 (Triangle Inequality). *Let $\mathtt{Game}^0$, $\mathtt{Game}^1$ and $\mathtt{Game}^2$ be games such that $\mathsf{out}(\mathtt{Game}^0) = \mathsf{out}(\mathtt{Game}^1) = \mathsf{out}(\mathtt{Game}^2)$. If $\mathtt{Game}^0 \overset{\epsilon_1}{\approx} \mathtt{Game}^1$, $\mathtt{Game}^1 \overset{\epsilon_2}{\approx} \mathtt{Game}^2$, and $\mathtt{Game}^0 \overset{\epsilon_3}{\approx} \mathtt{Game}^2$, then $\epsilon_3 \leq \epsilon_1 + \epsilon_2$.*

The triangle inequality helps to sum up game-hops. Many game-hops will exploit simple associativity, as the following lemma illustrates.

Lemma 3 (Composition). *Let $\mathtt{Game}^0$ and $\mathtt{Game}^1$ be games with the same output interface $\mathsf{out}(\mathtt{Game}^0) = \mathsf{out}(\mathtt{Game}^1)$ and let $\mathtt{M}$ be a package such that $\mathsf{in}(M) \subseteq \mathsf{out}(\mathtt{Game}^0)$. Let $\mathcal{A}$ be an adversary that matches the output interface of $\mathtt{M}$, then for $b \in \{0,1\}$ the adversary $\mathcal{D} := \mathcal{A} \circ \mathtt{M}$ satisfies,*

$$\Pr\left[0 \leftarrow_{\$} \mathcal{A} \circ (\mathtt{M} \circ \mathtt{Game}^b)\right] = \Pr\left[0 \leftarrow_{\$} \mathcal{D} \circ \mathtt{Game}^b\right].$$

As a corollary, we obtain

$$\mathcal{A} \circ \mathtt{M} \circ \mathtt{Game}^0 \overset{\epsilon(\mathcal{A})}{\approx} \mathcal{A} \circ \mathtt{M} \circ \mathtt{Game}^1 \quad (\textit{for } \epsilon(\mathcal{A}) = \epsilon_{\mathtt{Game}}(\mathcal{A} \circ \mathtt{M})).$$

Proof. The proof follows by the definition of $\mathcal{D} = \mathcal{A} \circ \mathtt{M}$ and associativity of package composition, i.e., $\mathcal{A} \circ (\mathtt{M} \circ \mathtt{Game}^b) = (\mathcal{A} \circ \mathtt{M}) \circ \mathtt{Game}^b = \mathcal{D} \circ \mathtt{Game}^b$.

The comma operator which we define now is essentially a disjoint union operator that takes two packages and builds a new package that implements both of them in parallel. It is important to note that only the output interfaces of $\mathtt{M}$ and $\mathtt{N}$ need to be disjoint, while they can potentially share input oracles. This feature allows for parallel composition of several packages that use the same input interface.

Definition 9 (Comma-Bracket Operator $[.,.]$). *The operator*

$$[\mathtt{M}, \mathtt{N}]$$

takes as input packages $\mathtt{M}$ *and* $\mathtt{N}$ *such that* $\mathsf{out}(\mathtt{M}) \cap \mathsf{out}(\mathtt{N}) = \emptyset$*. For the resulting package* $\mathtt{P} := [\mathtt{N}, \mathtt{N}]$*, we get* $\mathsf{in}(\mathtt{P}) = \mathsf{in}(\mathtt{M}) \cup \mathsf{in}(\mathtt{N})$ *and* $\mathsf{out}(\mathtt{P}) = \mathsf{out}(\mathtt{M}) \uplus \mathsf{out}(\mathtt{N})$*. The package* $[\mathtt{N}, \mathtt{M}]$*, when receiving a query to an oracle* O*, runs* $\mathtt{M}$ *on the query if* $\mathsf{O.name} \in \mathsf{out}(\mathtt{M})$*, or it runs* $\mathtt{N}$ *if* $\mathsf{O.name} \in \mathsf{out}(\mathtt{N})$*.*

Lemma 4. *The comma-bracket operator is commutative and associative.*

We omit the proof. Associativity of the comma-bracket operator allows us to write $[\mathtt{M}_1, \ldots, \mathtt{M}_n]$ for the parallel composition of multiple packages.

Lemma 5 (Interchange Rule). *If package* $\mathtt{M}$ *matches* $\mathsf{out}(\mathtt{O})$*, package* $\mathtt{N}$ *matches* $\mathsf{out}(\mathtt{P})$*, and* $\mathsf{out}(\mathtt{O}) \cap \mathsf{out}(\mathtt{P}) = \emptyset$*, then*

$$[\mathtt{M}, \mathtt{N}] \circ [\mathtt{O}, \mathtt{P}] = [\mathtt{M} \circ \mathtt{O}, \mathtt{N} \circ \mathtt{P}] \ .$$

Proof sketch. We again sketch the proof using inlining semantics. Consider the package $[\mathtt{M}, \mathtt{N}]$. We inline the package $[\mathtt{O}, \mathtt{P}]$. However, as $\mathtt{M}$ only makes queries to $\mathtt{O}$ and as $\mathtt{N}$ only makes queries to $\mathtt{P}$, the inlining is a disjoint operation and we obtain $[\mathtt{M} \circ \mathtt{O}, \mathtt{N} \circ \mathtt{P}]$.

Identity packages. Some proofs and definitions make one or more oracles of a package unavailable to the adversary, which we capture as follows.

Definition 10 (Identity Packages). *Let* S *be a set of oracle names. An identity package* $\mathtt{ID}_S$ *has interfaces* $\mathsf{in}(\mathtt{ID}_S) = \mathsf{out}(\mathtt{ID}_S) = S$*. It contains oracles* O*, where* $\mathsf{O.name} \in S$*. Each oracle* O *of* $\mathtt{ID}_S$*, on receiving an input* x*, calls the equally named oracle* O *from the input interface on* x *and returns the answer.*

Lemma 6 (Identity Rule). *Let* $\mathtt{M}$ *and* $\mathtt{N}$ *be packages and* $\mathsf{in}(M) \subseteq S \subseteq \mathsf{out}(N)$*. Then the following equivalence is called the identity rule:*

$$\mathtt{M} \circ \mathtt{N} = \mathtt{M} \circ \mathtt{ID}_S \circ \mathtt{N}$$

As a corollary of the rule we have $\mathtt{M} = \mathtt{M} \circ \mathtt{ID}_{\mathsf{in}(M)}$ *and* $\mathtt{N} = \mathtt{ID}_{\mathsf{out}(N)} \circ \mathtt{N}$*.*

Proof sketch. Since we have $\mathsf{in}(\mathtt{M}) \subseteq S = \mathsf{out}(\mathtt{ID}_S) = \mathsf{in}(\mathtt{ID}_S) = S \subseteq \mathsf{out}(\mathtt{N})$, $\mathtt{M}$ matches $\mathsf{out}(\mathtt{ID}_S)$ and $\mathtt{ID}_S$ matches $\mathsf{out}(\mathtt{N})$. Also, since $\mathtt{ID}_S$ simply forwards any queries, functionality does not change. The corollaries follow from $\mathsf{in}(\mathtt{M}) \subseteq \mathsf{in}(\mathtt{M})$ and $\mathsf{out}(\mathtt{N}) \subseteq \mathsf{out}(\mathtt{N})$ always holding regardless of further composition.

2.1 KEY Package Composition

Many cryptographic constructions emerge as compositions of two cryptographic building block: The first building block generates the (symmetric) key(s) and the second building block uses the (symmetric) key(s). Composition of key encapsulation mechanisms (KEM) with a deterministic encryption mechanism (DEM) is built this way. Likewise, complex protocols such as TLS first execute a key exchange protocol to use the session keys for a secure channel. In composition proofs, the key generating building block and the key-consuming building block thus share some state, namely the (symmetric) key(s).

To ease the sharing of state, we now introduce a key package $\mathtt{KEY}^\lambda$ which can store a single[6] key k in its internal state. The key package $\mathtt{KEY}^\lambda$ handles the key via its oracles $\mathsf{GEN}, \mathsf{SET}$ and GET. It is parameterized by a key length λ. GEN (and SET) can be used to initialize the key with a random value of length λ (or a specific value). GET requires the key to be initialized via GEN or SET before it can be called. We now define the $\mathtt{KEY}^\lambda$ package and then show how to use it for composing a key-generating package with a key-consuming package.

Definition 11 ($\mathtt{KEY}$ Package). *Let $\lambda \in \mathbb{N}$. We define $\mathsf{in}(\mathtt{KEY}^\lambda) = \emptyset$ and $\mathsf{out}(\mathtt{KEY}^\lambda) = \{\mathsf{GEN}, \mathsf{SET}, \mathsf{GET}\}$, where the oracles are described as follows:*

$\mathsf{GEN}()$	$\mathsf{SET}(k')$	$\mathsf{GET}()$
assert $k = \bot$	**assert** $k = \bot$	**assert** $k \neq \bot$
$k \leftarrow_\$ \{0,1\}^\lambda$	$k \leftarrow k'$	**return** k

Definition 12 (Key Generating Package). *Let $\lambda \in \mathbb{N}$. We call $\mathtt{A}^b$, $b \in \{0,1\}$ a* key generating package *if $\mathsf{in}(\mathtt{A}^0) = \{\mathsf{SET}\}$, $\mathsf{in}(\mathtt{A}^1) = \{\mathsf{GEN}\}$ and $\mathsf{out}(\mathtt{A}^0) = \mathsf{out}(\mathtt{A}^1)$. We define*

$$\left[\mathtt{A}^0, \mathtt{ID}_{\{\mathsf{GET}\}}\right] \circ \mathtt{KEY}^\lambda \overset{\epsilon_{\mathtt{A}}}{\approx} \left[\mathtt{A}^1, \mathtt{ID}_{\{\mathsf{GET}\}}\right] \circ \mathtt{KEY}^\lambda$$

Definition 13 (Key Consuming Package). *Let $\lambda \in \mathbb{N}$. We call $\mathtt{B}^d$, $d \in \{0,1\}$, a* key consuming package *if $\mathsf{in}(\mathtt{B}^d) = \{\mathsf{GET}\}$ and $\mathsf{out}(\mathtt{B}^0) =$*

[6] We discuss the multi-instance variant in Section 4.

$\mathsf{out}(\mathtt{B}^1)$. *We define*

$$\left[\mathtt{ID}_{\{\mathsf{GEN}\}}, \mathtt{B}^0\right] \circ \mathtt{KEY}^\lambda \overset{\epsilon_{\mathtt{B}}}{\approx} \left[\mathtt{ID}_{\{\mathsf{GEN}\}}, \mathtt{B}^1\right] \circ \mathtt{KEY}^\lambda$$

Definition 14 (Compatible Packages). *We call a key-consuming package* $\mathtt{B}^d$, $d \in \{0,1\}$, *and a key-generating package* $\mathtt{A}^b$, $b \in \{0,1\}$ compatible *if they use* $\mathtt{KEY}^\lambda$ *packages with the same* $\lambda \in \mathbb{N}$, *and for all* $b, d \in \{0,1\}$,

$$\mathsf{out}(\mathtt{A}^b) \cap \mathsf{out}(\mathtt{B}^d) = \emptyset.$$

Lemma 7 (Single Key Lemma). *Let* $\mathtt{A}^b$, $b \in \{0,1\}$ *be a key generating package and* $\mathtt{B}^d$, $d \in \{0,1\}$ *a key consuming package such that the two are compatible with* $\lambda \in \mathbb{N}$. *Then we have that*

$$\left[\mathtt{A}^0, \mathtt{B}^0\right] \circ \mathtt{KEY}^\lambda \overset{\epsilon}{\approx} \left[\mathtt{A}^0, \mathtt{B}^1\right] \circ \mathtt{KEY}^\lambda, \tag{1}$$

where for all adversaries $\mathcal{A}$, $\epsilon(\mathcal{A})$ *is less or equal to*

$$\epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^0\right]\right) + \epsilon_{\mathtt{B}}\left(\mathcal{A} \circ \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B})}\right]\right) + \epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^1\right]\right).$$

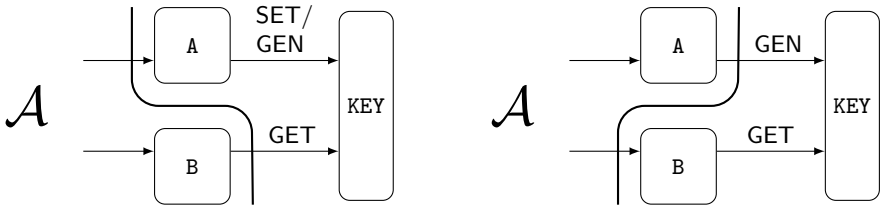

Fig. 2: Reduction to the key providing cryptographic building block (left) and reduction to the key consuming cryptographic building block (right).

Proof. The proof proceeds by (1) idealizing the key providing game (left side of Fig. 2, SET switches to GEN), (2) idealizing the key consuming game (right side of Fig. 2) and (3) de-idealizing the key providing game again (left side of Fig. 2, GEN switches to SET).

Technically, we use the algebraic rules introduced in Section 2 to transform the construction such that we can apply Def. (12) and Def. (13).

Idealizing the key providing game. The first intermediate goal is to bring the package into a shape where we can use Def. 12 to change $\mathtt{A}^0$ into $\mathtt{A}^1$, which happens in the first 4 lines. Below, for all adversaries $\mathcal{A}$, we have $\epsilon_1(\mathcal{A}) = \epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^0\right]\right)$.

$$
\begin{aligned}
&\left[\mathtt{A}^0, \mathtt{B}^0\right] \circ \mathtt{KEY}^\lambda && \\
&= \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^0\right] \circ \left[\mathtt{A}^0, \mathtt{ID}_{\{\mathsf{GET}\}}\right] \circ \mathtt{KEY}^\lambda && \text{identity \& interchange} \\
&\overset{\epsilon_1}{\approx} \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^0\right] \circ \left[\mathtt{A}^1, \mathtt{ID}_{\{\mathsf{GET}\}}\right] \circ \mathtt{KEY}^\lambda && \text{Def. 12} \\
&= \left[\mathtt{A}^1, \mathtt{B}^0\right] \circ \mathtt{KEY}^\lambda && \text{interchange \& identity}
\end{aligned}
$$

Idealizing the key consuming game. Now, as a second step, we want to use Def. 13 to move from $\mathtt{B}^0$ to $\mathtt{B}^1$. Towards this goal, we need to make $\mathtt{ID}_{\{\mathsf{GEN}\}}$ appear. Note that we can use $\mathtt{ID}_{\{\mathsf{GEN}\}}$ because $\{\mathsf{GEN}\}$ is equal to the input interface of $\mathtt{A}^1$. This was not possible before idealizing to $\mathtt{A}^1$, since $\mathsf{in}(\mathtt{A}^0) = \{\mathsf{SET}\}$. Below, for all adversaries $\mathcal{A}$, we have $\epsilon_2(\mathcal{A}) = \epsilon_{\mathtt{B}}\left(\mathcal{A} \circ \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B})}\right]\right)$.

$$
\begin{aligned}
&\left[\mathtt{A}^1, \mathtt{B}^0\right] \circ \mathtt{KEY}^\lambda && \\
&= \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B})}\right] \circ \left[\mathtt{ID}_{\{\mathsf{GEN}\}}, \mathtt{B}^0\right] \circ \mathtt{KEY}^\lambda && \text{identity \& interchange} \\
&\overset{\epsilon_2}{\approx} \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B})}\right] \circ \left[\mathtt{ID}_{\{\mathsf{GEN}\}}, \mathtt{B}^1\right] \circ \mathtt{KEY}^\lambda && \text{Def. 13} \\
&= \left[\mathtt{A}^1, \mathtt{B}^1\right] \circ \mathtt{KEY}^\lambda && \text{interchange \& identity}
\end{aligned}
$$

Deidealizing the key providing game. Finally, we need to move back from $\mathtt{A}^1$ to $\mathtt{A}^0$, which will be the inverse steps of the operations we did in the beginning of the proof. Below, for all adversaries $\mathcal{A}$, we have $\epsilon_3(\mathcal{A}) = \epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^1\right]\right)$.

$$
\begin{aligned}
&\left[\mathtt{A}^1, \mathtt{B}^1\right] \circ \mathtt{KEY}^\lambda && \\
&= \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^1\right] \circ \left[\mathtt{A}^1, \mathtt{ID}_{\{\mathsf{GET}\}}\right] \circ \mathtt{KEY}^\lambda && \text{identity \& interchange} \\
&\overset{\epsilon_3}{\approx} \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^1\right] \circ \left[\mathtt{A}^0, \mathtt{ID}_{\{\mathsf{GET}\}}\right] \circ \mathtt{KEY}^\lambda && \text{Def. 12} \\
&= \left[\mathtt{A}^0, \mathtt{B}^1\right] \circ \mathtt{KEY}^\lambda && \text{interchange \& identity}
\end{aligned}
$$

To obtain the desired upper bound on $\epsilon(\mathcal{A})$, we use the triangle inequality and sum up the advantages which concludes the proof of Lemma 2.

$$\begin{aligned}&\epsilon_1(\mathcal{A}) + \epsilon_2(\mathcal{A}) + \epsilon_3(\mathcal{A})\\ =&\epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^0\right]\right) + \epsilon_{\mathtt{B}}\left(\mathcal{A} \circ \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B})}\right]\right) + \epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^1\right]\right)\end{aligned}$$

3 KEM-DEMs

We now use the KEY package composition introduced in the previous section to give a new formulation of the KEM-DEM proof by Cramer and Shoup [17, §7] which shows that composing a CCA-secure key encapsulation mechanism (KEM) and a CCA-secure data encapsulation mechanism (DEM) yields a CCA secure public-key encryption (PKE).

We denote PKE schemes by $\zeta = (kgen, enc, dec)$, using standard notation and semantics. We denote DEM schemes by $\theta = (kgen, enc, dec)$, where we recall that *enc* is a deterministic algorithm. We will prepend algorithm names by ζ and θ for disambiguation. We denote KEM schemes by $\eta = (kgen, encap, decap)$, where *kgen* produces a key pair (pk, sk), $encap(pk)$ generates a symmetric key k of length $\eta.\lambda$ and a key encapsulation c, while $decap(sk, c)$ given sk and an encapsulation c returns a key k. For all three schemes, we consider perfect correctness. Throughout this section, we consider a single symmetric-key length λ that corresponds to the length of the symmetric key used by the DEM scheme as well as the length of the symmetric key produced by the encapsulation mechanism $\eta.encap$. We now turn to the security notions which are the standard IND-CPA security notions for all three primitives.

Definition 15 (PKE-CCA). *Let ζ be a PKE-scheme. For $b \in \{0,1\}$, $\mathsf{in}(\mathtt{PKE\text{-}CCA}^{b,\zeta}) = \emptyset$ and $\mathsf{out}(\mathtt{PKE\text{-}CCA}^{b,\zeta}) = \{\mathsf{PKGEN}, \mathsf{PKENC}, \mathsf{PKDEC}\}$, where the oracles are defined as follows.*

$\mathsf{PKGEN}()$	$\mathsf{PKENC}(m)$	$\mathsf{PKDEC}(c')$
assert $sk = \perp$	**assert** $pk \neq \perp$	**assert** $sk \neq \perp$
$pk, sk \leftarrow\!\!\$\ \zeta.kgen()$	**assert** $c = \perp$	**assert** $c' \neq c$
return pk	**if** $b = 0$ **then**	$m \leftarrow \zeta.dec(sk, c')$
	$c \leftarrow\!\!\$\ \{0,1\}^{\lvert m\rvert}$	**return** m
	else	
	$c \leftarrow\!\!\$\ \zeta.enc(pk, m)$	
	return c	

We denote the PKE-CCA advantage as $\mathtt{PKE\text{-}CCA}^{0,\zeta} \overset{\epsilon^{\zeta}_{\mathtt{PKE\text{-}CCA}}}{\approx} \mathtt{PKE\text{-}CCA}^{1,\zeta}$.

We model the KEM as a key producing and the DEM as a key consuming package. In the description of our definitions, we will use the $\texttt{KEY}^{\lambda}$ package as specified in Definition 11.

Definition 16 (KEM-CCA). *Let η be a KEM. For $d \in \{0,1\}$, we define input interface $\mathsf{in}(\texttt{KEM-CCA}^{d,\eta}) = \{\mathsf{SET}, \mathsf{GEN}\}$ and output interface $\mathsf{out}(\texttt{KEM-CCA}^{d,\eta}) = \{\mathsf{KEMGEN}, \mathsf{ENCAP}, \mathsf{DECAP}\}$, where the oracles of $\texttt{KEM-CCA}^{d,\eta}$ are defined as follows:*

$\mathsf{KEMGEN}()$	$\mathsf{ENCAP}()$	$\mathsf{DECAP}(c')$
assert $sk = \bot$	**assert** $pk \neq \bot$	**assert** $sk \neq \bot$
$pk, sk \leftarrow\!\!\$\ \eta.kgen()$	**assert** $c = \bot$	**assert** $c' \neq c$
return pk	$k, c \leftarrow\!\!\$\ \eta.encap(pk)$	$k \leftarrow \eta.decap(sk, c')$
	if $b = 0$ **then**	**return** k
	$\mathsf{SET}(k)$	
	else	
	$\mathsf{GEN}()$	
	return c	

We denote the KEM-CCA advantage as

$$\left[\texttt{KEM-CCA}^{0,\eta}, \texttt{ID}_{\{\mathsf{GET}\}}\right] \circ \texttt{KEY}^{\eta.\lambda} \overset{\epsilon^{\eta}_{\texttt{KEM-CCA}}}{\approx} \left[\texttt{KEM-CCA}^{1,\eta}, \texttt{ID}_{\{\mathsf{GET}\}}\right] \circ \texttt{KEY}^{\eta.\lambda}.$$

Definition 17 (DEM-CCA). *Let θ be a DEM. For $b \in \{0,1\}$, we define input interface $\mathsf{in}(\texttt{DEM-CCA}^{b,\theta}) = \{\mathsf{GET}\}$ and output interface $\mathsf{out}(\texttt{DEM-CCA}^{b,\theta}) = \{\mathsf{ENC}, \mathsf{DEC}\}$, where the oracles of $\texttt{DEM-CCA}^{b,\theta}$ are defined as follows:*

$\mathsf{ENC}(m)$	$\mathsf{DEC}(c')$
assert $c = \bot$	**assert** $c \neq c'$
$k \leftarrow \mathsf{GET}()$	$k \leftarrow \mathsf{GET}()$
if $d = 0$ **then**	$m \leftarrow \theta.dec(k, c')$
$c \leftarrow\!\!\$\ \{0,1\}^{\lvert m\rvert}$	**return** m
else	
$c \leftarrow\!\!\$\ \theta.enc(k, m)$	
return c	

We denote the DEM-CCA advantage as

$$\left[\texttt{DEM-CCA}^{0,\theta}, \texttt{ID}_{\{\mathsf{GEN}\}}\right] \circ \texttt{KEY}^{\theta.\lambda} \overset{\epsilon^{\theta}_{\texttt{DEM-CCA}}}{\approx} \left[\texttt{DEM-CCA}^{1,\theta}, \texttt{ID}_{\{\mathsf{GEN}\}}\right] \circ \texttt{KEY}^{\theta.\lambda}.$$

3.1 Composition and Proof

We prove that the PKE scheme obtained by composing a KEM-CCA secure KEM and a DEM-CCA secure DEM is PKE-CCA secure.

Construction 1 (KEM-DEM Construction) *Let η be a KEM and θ be a DEM. We define the PKE scheme ζ as follows:*

$\underline{\zeta.kgen()}$
return $\eta.gen()$

$\underline{\zeta.enc(pk, m)}$
$k, c_1 \leftarrow\!\!\$\ \eta.encap(pk)$
$c_2 \leftarrow \theta.enc(k, m)$
return $c_1||c_2$

$\underline{\zeta.dec(sk, c)}$
$c_1||c_2 \leftarrow c$
$k \leftarrow \eta.decap(sk, c_1)$
$m \leftarrow \theta.dec(k, c_2)$
return m

Theorem 1 (PKE Security of the KEM-DEM Construction). *Let ζ be the PKE scheme in Construction 1. For adversaries $\mathcal{A}$, we have that*

$$\begin{aligned}\epsilon^{\zeta}_{\texttt{PKE-CCA}}(\mathcal{A}) \leq\ & \epsilon^{\eta}_{\texttt{KEM-CCA}}\left(\mathcal{A} \circ \texttt{MOD-CCA} \circ \left[\texttt{ID}_{\mathsf{out}(\texttt{KEM-CCA}^{0,\eta})}, \texttt{DEM-CCA}^{0,\theta}\right]\right) \\ & + \epsilon^{\theta}_{\texttt{DEM-CCA}}\left(\mathcal{A} \circ \texttt{MOD-CCA} \circ \left[\texttt{KEM-CCA}^{1,\eta}, \texttt{ID}_{\mathsf{out}(\texttt{DEM-CCA}^{0,\theta})}\right]\right) \\ & + \epsilon^{\eta}_{\texttt{KEM-CCA}}\left(\mathcal{A} \circ \texttt{MOD-CCA} \circ \left[\texttt{ID}_{\mathsf{out}(\texttt{KEM-CCA}^{0,\eta})}, \texttt{DEM-CCA}^{1,\theta}\right]\right)\ ,\end{aligned}$$

where $\mathsf{out}(\texttt{MOD-CCA}) = \mathsf{out}(\texttt{PKE-CCA}^{\zeta,0})$*, the oracles of* MOD-CCA *are defined in Fig. 3, and* $\mathsf{in}(\texttt{MOD-CCA}) = \mathsf{out}(\texttt{DEM-CCA}^{0,\theta}) \cup \mathsf{out}(\texttt{KEM-CCA}^{0,\eta})$.

$\underline{\mathsf{PKGEN}()}$
assert $pk = \bot$
$pk \leftarrow \mathsf{KEMGEN}()$
return pk

$\underline{\mathsf{PKENC}(m)}$
assert $pk \neq \bot$
assert $c = \bot$
$c_1 \leftarrow \mathsf{ENCAP}()$
$c_2 \leftarrow \mathsf{ENC}(m)$
$c \leftarrow c_1||c_2$
return (c)

$\underline{\mathsf{PKDEC}(c')}$
assert $pk \neq \bot$
assert $c \neq c'$
$c'_1||c'_2 \leftarrow c'$
if $c'_1 = c_1$ **then**
  $m \leftarrow \mathsf{DEC}(c'_2)$
else
  $k' \leftarrow \mathsf{DECAP}(c'_1)$
  $m \leftarrow \theta.dec(k', c'_2)$
return m

Fig. 3: MOD-CCA construction.

In Appendix D, we prove via code comparison that for $b \in \{0, 1\}$,

$$\texttt{PKE-CCA}^{b,\zeta} = \texttt{MOD-CCA} \circ \left[\texttt{KEM-CCA}^{0,\eta}, \texttt{DEM-CCA}^{b,\theta}\right] \circ \texttt{KEY}^{\lambda}.$$

Thus, for all adversaries $\mathcal{A}$, we can now apply Lemma 7 to the adversary $\mathcal{B} = \mathcal{A} \circ \texttt{MOD-CCA}$, as $\texttt{KEM-CCA}^{d,\eta}$, $d \in \{0,1\}$ is indeed a key generating package, $\texttt{DEM-CCA}^{b,\theta}$, $b \in \{0,1\}$ is indeed a key consuming package, and the two are compatible. For all adversaries $\mathcal{B}$, we denote

$$\mathcal{B} \circ \left[\texttt{KEM-CCA}^{\eta,0}, \texttt{DEM-CCA}^{\theta,0}\right] \circ \texttt{KEY}^{\lambda} \overset{\epsilon(\mathcal{B})}{\approx} \mathcal{B} \circ \left[\texttt{KEM-CCA}^{\eta,0}, \texttt{DEM-CCA}^{\theta,1}\right] \circ \texttt{KEY}^{\lambda}.$$

Thus, via Lemma 7, we obtain that for all adversaries $\mathcal{B}$, the value $\epsilon(\mathcal{B})$ is less or equal to

$$\begin{aligned}
\epsilon^{\zeta}_{\texttt{PKE-CCA}}(\mathcal{A}) \leq\ & \epsilon^{\eta}_{\texttt{KEM-CCA}} \left(\mathcal{B} \circ \left[\texttt{ID}_{\mathsf{out}(\texttt{KEM-CCA}^{0,\eta})}, \texttt{DEM-CCA}^{0,\theta}\right]\right) \\
& + \epsilon^{\theta}_{\texttt{DEM-CCA}} \left(\mathcal{B} \circ \left[\texttt{KEM-CCA}^{1,\eta}, \texttt{ID}_{\mathsf{out}(\texttt{DEM-CCA}^{0,\theta})}\right]\right) \\
& + \epsilon^{\eta}_{\texttt{KEM-CCA}} \left(\mathcal{B} \circ \left[\texttt{ID}_{\mathsf{out}(\texttt{KEM-CCA}^{0,\eta})}, \texttt{DEM-CCA}^{1,\theta}\right]\right) \ ,
\end{aligned}$$

Plugging in $\mathcal{B} = \mathcal{A} \circ \texttt{MOD-CCA}$ concludes the proof.

4 Multi-Instance Packages and Composition

Cryptographic primitives are usually not used in isolation, but rather are executed, concurrently, by many different users with different keys. Now consider a game $\texttt{B}$ which is used to model the security of a cryptographic primitive or protocol, e.g., a secure channel or an authenticated encryption scheme.

A simple way to have the adversary interact with *multiple* instances is to use the comma-bracket operator and n packages $\texttt{B}$. To distinguish the different instances, we introduce an additional *index* parameter i with $0 < i \leq n$. For convenience, we here use the array notation instead of the superscript notation (which we use for other parameters).

$$\mathcal{A} \circ [\texttt{B}[1], \texttt{B}[2], \texttt{B}[3], \ldots, \texttt{B}[n]] = \mathcal{A} \circ \texttt{B}[1..n].$$

Recall, that a requirement for using the comma-bracket operator is that the output interfaces of $\texttt{B}[i]$ for $1 \leq i \leq n$ have no intersection. We accomplish this by including the index in the oracle names. We define the packages with multiple instances formally as follows:

Definition 18 (Package Instances). *Let* $\texttt{B}$ *be a package with a set of oracles* Ω *operating on state* Σ*. For all* $j, i \in \mathbb{N}$*,* $j \neq i$*, we denote by* $\Sigma[i]$ *and* $\Sigma[j]$ *disjoint copies of* Σ*. For* $\mathsf{O} \in \Omega$*, we denote by* $\overline{\mathsf{O}[i]}$ *the oracle*

that operates as O*, except that it uses the variables in state* $\Sigma[i]$*. We denote by* $\mathsf{O}[i]$ *the oracle that behaves as* $\overline{\mathsf{O}[i]}$ *except that, whenever* $\overline{\mathsf{O}[i]}$ *calls an oracle* O'*, then* $\mathsf{O}[\mathsf{i}]$ *calls an oracle* $\mathsf{O}'[\mathsf{i}]$*. We denote by* $\mathtt{B}[i]$ *and* $\overline{\mathtt{B}[i]}$ *an* instance *of* $\mathtt{B}$ *with state* $\Sigma[i]$*, where* $\mathsf{out}(\overline{\mathtt{B}[i]}) := \{\overline{\mathsf{O}[i]} | \mathsf{O} \in \mathsf{out}(\mathtt{B})\}$ *and* $\mathsf{out}(\mathtt{B}[i]) := \{\mathsf{O}[i] | \mathsf{O} \in \mathsf{out}(\mathtt{B})\}$*, as well as* $\mathsf{in}(\mathtt{B}[i]) := \{\mathsf{O}[i] | \mathsf{O} \in \mathsf{in}(\mathtt{B})\}$ *and* $\mathsf{in}(\overline{\mathtt{B}[i]}) := \mathsf{in}(\mathtt{B})$*. For* $n \in \mathbb{N}$*, we define* $\overline{\mathtt{B}[1..n]} := [\overline{\mathtt{B}[1]}, .., \overline{\mathtt{B}[n]}]$ *and* $\mathtt{B}[1..n] := [\mathtt{B}[1], .., \mathtt{B}[n]]$*.*

4.1 Multi-Instance Lemma

We introduce a multi-instance lemma that allows us to turn arbitrary games using symmetric keys into multi-instance games.

Lemma 8 (Multi-Instance Lemma). *Let* $\mathtt{M}^b$ *be a game with distinguishing advantage* $\epsilon_{\mathtt{M}}$*. Then for any number* n *of instances, adversaries* $\mathcal{A}$*, and reduction* $\mathcal{R}$ *that samples* $i \leftarrow_{\$} \{1, \ldots, n\}$ *and runs* $[\mathtt{M}^0[1..i-1], \overline{\mathtt{ID}_{\mathsf{out}(\mathtt{M})}[i]}, \mathtt{M}^1[i+1..n]]$ *we have*

$$\epsilon_{\mathtt{M}[1..n]}(\mathcal{A}) \leq n \cdot \epsilon_{\mathtt{M}}(\mathcal{A} \circ \mathcal{R})$$

See Appendix B we provide a systematic recipe for hybrid arguments and instantiate it for the proof of this lemma.

From multi-instance to unbounded instance. We see multi-instance games primarily as a proof technique. Moving from multi-instance games to unbounded instance games requires an additional proof step.

Consider an unbounded instance game $\mathtt{U}$, e.g., with interface $\mathsf{in}(\mathtt{U}) = \{\mathsf{Gen}, \mathsf{O}'\}$ where oracle O' takes a handle returned by the instance generating oracle Gen as input. We introduce a module $\mathtt{A}$ with $\mathsf{out}(\mathtt{A}) = \{\mathsf{Gen}, \mathsf{O}'\}$ and $\mathsf{in}(\mathtt{A}) = \{\mathtt{B}[i..n].\mathsf{O}\}$. In the proof, we can bound the number of queries made by $\mathcal{A}$ to GEN by n and use a functional equivalence proof step to show that $\mathcal{A} \circ \mathtt{U}^b = \mathcal{A} \circ \mathtt{A} \circ \mathtt{B}^b[1..n]$.

The following lemma states that the multi-instance operator $[1..n]$ commutes with the operators $\circ$, $[.,.]$ and $\mathtt{ID}$.

Lemma 9 (Multi-Instance Interchange). *Let* $\mathtt{M}$ *and* $\mathtt{N}$ *be packages such that* $\mathtt{M}$ *matches the output interface of* $\mathtt{N}$*. Let* $\mathtt{P}$ *be a packages such that* $\mathsf{out}(\mathtt{M})$ *and* $\mathsf{out}(\mathtt{P})$ *are disjoint. Let* S *be set of queries. Then, for any*

number n of instances, the following hold:

$$\begin{aligned}(\mathtt{M} \circ \mathtt{N})[1..n] &= \mathtt{M}[1..n] \circ \mathtt{N}[1..n] \\ [\mathtt{M}, \mathtt{P}][1..n] &= [\mathtt{M}[1..n], \mathtt{P}[1..n]] \\ \mathtt{ID}_{S[1..n]} &= \mathtt{ID}_S[1..n] \\ \overline{\mathtt{M}[i]} &= \overline{\mathtt{ID}_{\mathsf{out}(M)}[i]} \circ \mathtt{M}\end{aligned}$$

Proof. Firstly, note that the package $\mathtt{M}[1..n] \circ \mathtt{N}[1..n]$ is well-defined, since $\mathtt{M}[1..n]$ matches the input interface of $\mathtt{N}[1..n]$ due to Definition **??**. Moreover, note that $[\mathtt{M}[1..n], \mathtt{P}[1..n]]$ is well-defined due to the disjointness condition observed in Equation **??**. In the following, each of the three columns corresponds to a proof of one equation. We omit the proof of the fourth equation.

$$\begin{aligned}&M[1..n] \circ N[1..n] \\ &= [M[1], \ldots, M[n]] \circ [N[1], \ldots, N[n]] \\ &= [M[1] \circ N[1], \ldots, M[n] \circ N[n]] \\ &= (M \circ N)[1..n]\end{aligned}$$

$$\begin{aligned}&[M[1..n], P[1..n]] \\ &= [[M[1], P[1]]..[M[n], P[n]] \\ &= [M, P][1..n]\end{aligned}$$

$$\begin{aligned}&\mathsf{in}(\mathtt{ID}_S[1..n]) \\ &= S[1] \cup \cdots \cup S[n] \\ &= S[1..n] \\ &= \mathsf{in}(\mathtt{ID}_{S[1..n]})\end{aligned}$$

4.2 Multi-Instance Key Lemma

We now combine key composition and multi-instance lemmas. For this purpose, we use a multi-instance version of the following single-instance package `CKEY`. In contrast to the simpler `KEY` package, `CKEY` allows for corrupted keys (whence the name `CKEY`) and, consequently, needs to allow the symmetric-key protocol to check whether keys are honest.

Definition 19 (`CKEY` Package). *The* `CKEY` *package is parameterized with a key length parameter $\lambda \in \mathbb{N}$ and has input interface $\mathsf{in}(\mathtt{CKEY}) = \emptyset$ and $\mathsf{out}(\mathtt{CKEY}) = \{\mathsf{GEN}, \mathsf{SET}, \mathsf{CSET}, \mathsf{GET}, \mathsf{HON}\}$.*

$\mathsf{GEN}()$	$\mathsf{SET}(k')$	$\mathsf{CSET}(k')$	$\mathsf{GET}()$	$\mathsf{HON}()$
assert $k = \bot$	**assert** $k = \bot$	**assert** $k = \bot$	**assert** $k \neq \bot$	**assert** $h \neq \bot$
$k \leftarrow\!\!\$\ \{0,1\}^\lambda$	$k \leftarrow k'$	$k \leftarrow k'$		
$h \leftarrow 1$	$h \leftarrow 1$	$h \leftarrow 0$	**return** k	**return** h

Fig. 4: Oracles of the $\mathtt{CKEY}^\lambda$ package.

The corruptible key providing package is multi-instance and can set corrupty keys via the CSET oracle. The corruptible key consuming package will be turned into a multi-instance package later and can access the honesty status of keys via the HON oracle.

Definition 20 (Corruptible Key Generating Package). *We call* $\mathtt{A}^b$, $b \in \{0,1\}$ *a* key generating package *if* $\mathsf{in}(\mathtt{A}^0) = \{\mathsf{SET}, \mathsf{CSET}\}$ *and* $\mathsf{in}(\mathtt{A}^1) = \{\mathsf{GEN}, \mathsf{CSET}\}$. *For* $n, \lambda \in \mathbb{N}$, *we define*

$$\left[\mathtt{A}^0, \mathtt{ID}_{\{\mathsf{GET},\mathsf{HON}\}}\right] \circ \mathtt{KEY}^\lambda[1..n] \overset{\epsilon_{\mathtt{A}}}{\approx} \left[\mathtt{A}^1, \mathtt{ID}_{\{\mathsf{GET},\mathsf{HON}\}}\right] \circ \mathtt{KEY}^\lambda[1..n]$$

Definition 21 (Corruptible Key Consuming Package). *We call* $\mathtt{B}^d$, $d \in \{0,1\}$ *a* key consuming package *if* $\mathsf{in}(\mathtt{B}^d) = \{\mathsf{GET}, \mathsf{HON}\}$. *For* $\lambda \in \mathbb{N}$, *we define*

$$\left[\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{B}^0\right] \circ \mathtt{KEY}^\lambda \overset{\epsilon_{\mathtt{B}}}{\approx} \left[\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{B}^1\right] \circ \mathtt{KEY}^\lambda$$

Key generating and consuming packages $\mathtt{A}$ and $\mathtt{B}$ are compatible if for all $n \in \mathbb{N}$, $\mathsf{out}(\mathtt{A}) \cap \mathsf{out}(\mathtt{B}[1..n])$ is empty and their key length parameters λ match.

Lemma 10 (Multi-Instance Key Lemma). *Let* $\mathtt{A}^b$, $b \in \{0,1\}$ *be a corruptible key generating package and* $\mathtt{B}^d$, $d \in \{0,1\}$ *a corruptible key consuming package such that the two are compatible with* $n, \lambda \in \mathbb{N}$.

$$\left[\mathtt{A}^0, \mathtt{B}^0[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n] \overset{\epsilon}{\approx} \left[\mathtt{A}^0, \mathtt{B}^1[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n]$$

where for all adversaries $\mathcal{A}$,

$$\begin{aligned}\epsilon(\mathcal{A}) \leq \quad & \epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^0[1..n]\right]\right) \\ & + n \cdot \epsilon_{\mathtt{B}}\left(\mathcal{A} \circ \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B}[1..n])}\right] \circ \mathcal{R}\right) \\ & + \epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^1[1..n]\right]\right),\end{aligned}$$

where reduction $\mathcal{R}$ *samples* $i \leftarrow\!\!\$\, \{1, \ldots, n\}$ *and implements the package* $[\mathtt{M}^0[1..i-1], \overline{\mathtt{ID}_{\mathsf{out}(\mathtt{M})}[i]}, \mathtt{M}^1[i+1..n]]$, *where* $\mathtt{M}^0 = [\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{B}^0] \circ \mathtt{CKEY}^\lambda$ *and* $\mathtt{M}^1 = [\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{B}^1] \circ \mathtt{CKEY}^\lambda$.

Proof. On a high-level, the proof of this lemma consists of two parts: We need to invoke the multi-instance lemma on the key-consuming package, and then, we need to carry out a proof that is analogous to the proof of the single-instance key composition lemma. In particular, we proceed again by idealizing the key providing game, then idealizing the key consuming game and finally deidealizing the key providing game.

Multi-instance lemma. We start by invoking the multi-instance lemma (Lemma 8) with $\mathtt{M}^0 = [\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{B}^0] \circ \mathtt{CKEY}^\lambda$ and $\mathtt{M}^1 = [\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{B}^1] \circ \mathtt{CKEY}^\lambda$. By applying Lemma 8, we obtain that for all adversaries $\mathcal{B}$, we have

$$\epsilon_{\mathtt{M}[1..n]}(\mathcal{B}) \leq n \cdot \epsilon_{\mathtt{B}}(\mathcal{B} \circ \mathcal{R}), \tag{2}$$

where reduction $\mathcal{R}$ samples $i \leftarrow\!\!\$\, \{1, \ldots, n\}$ and implements the package $[\mathtt{M}^0[1..i-1], \overline{\mathtt{ID}_{\mathsf{out}(\mathtt{M})}[i]}, \mathtt{M}^1[i+1..n]]$.

Idealizing the key providing game. For the second part of the proof, the steps that idealize the corruptible key providing game are analogous to the single-instance key composition proof, and we obtain

$$\left[\mathtt{A}^0, \mathtt{B}^0[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n] \overset{\epsilon_1(\mathcal{A})}{\approx} \left[\mathtt{A}^1, \mathtt{B}^0[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n],$$

where $\epsilon_1(\mathcal{A}) = \epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^0[1..n]\right]\right)$.

Idealizing the multi-instance version of $\mathtt{B}$. We discuss ϵ_2 after presenting the transformations. The step from the penultimate to the last line works by performing the first 3 steps of the proof in inverse order.

$$\begin{aligned}
&\left[\mathtt{A}^1, \mathtt{B}^0[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n] \\
&= \left[\mathtt{A}^1 \circ \mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}[1..n]}, \mathtt{ID}_{\mathsf{out}(\mathtt{B}[1..n])} \circ \mathtt{B}^0[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n] && \text{identity} \\
&= \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B}[1..n])}\right] \circ \left[\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}[1..n], \mathtt{B}^0[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n] && \text{interch.} \\
&= \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B}[1..n])}\right] \circ \left(\left[\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{B}^0\right] \circ \mathtt{CKEY}^\lambda\right)[1..n] && \text{interch.} \\
&\overset{\epsilon_2}{\approx} \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B}[1..n])}\right] \circ \left(\left[\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{B}^1\right] \circ \mathtt{CKEY}^\lambda\right)[1..n] && \text{Def. 13} \\
&= \left[\mathtt{A}^1, \mathtt{B}^1[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n]
\end{aligned}$$

We have $\epsilon_2(\mathcal{A}) = \epsilon_{\mathtt{M}[1..n]}\left(\mathcal{A} \circ \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B}[1..n])}\right]\right)$. Moreover, plugging in Inequality 2, we obtain

$$\epsilon_2(\mathcal{A}) \leq n \cdot \epsilon_{\mathtt{B}}\left(\mathcal{A} \circ \left[\mathtt{A}^1, \mathtt{ID}_{\mathsf{out}(\mathtt{B}[1..n])}\right] \circ \mathcal{R}\right).$$

Idealizing the key providing game. The steps that deidealize the corruptible key providing game are analogous to the single-instance key composition proof, and we obtain

$$\left[\mathtt{A}^1, \mathtt{B}^1[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n] \overset{\epsilon_3}{\approx} \left[\mathtt{A}^0, \mathtt{B}^1[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n],$$

where for all adversaries $\mathcal{A}$, we have $\epsilon_3(\mathcal{A}) = \epsilon_{\mathtt{A}}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{A})}, \mathtt{B}^0[1..n]\right]\right)$.

5 Composition of Bellare-Rogaway Key Exchange

We here give a short definition of eCK-forward secure key exchange protocols, see Cremers and Feltz [18] as well as the discussion in the end of this section.

Definition 22 (Key Exchange Protocol). *A key exchange protocol π consists of a key generation algorithm $\pi.kgen$ and a protocol algorithm $\pi.run$. $\pi.kgen$ returns a pair of keys:*

$$(sk, pk) \leftarrow_{\$} \pi.kgen$$

$\Pi.run$ takes as input a state and an incoming message and returns a state and an outgoing message:

$$(state, m) \leftarrow_{\$} \pi.run(state, m)$$

Each party holds several sessions and the algorithm $\pi.run$ is executed locally on the *session* state. We use indices i for sessions and indices u, v as for parties. For the ith session of party u, we denote the state by $\Pi[u, i].state$. The state contains at least the following variables. For a variable v, we denote by $\Pi[u, i].a$ the variable a stored in $\Pi[u, i].state$.

- (pk, sk): the party's own public-key and corresponding private key
- *peer*: the public-key of the intended peer for the session
- *role*: determines whether the session runs as an initiator or responder
- α: protocol state that is either *running* or *accepted*.
- k: the symmetric session key derived by the session

Upon initialization, the state is initialized with pair (pk, sk), the public-key *peer* of the intended peer of a session, a value $role \in \{I, R\}$, $\alpha = running$ and $k = \perp$. The first three variables cannot be changed. The variables α and k can be set only once. We require that

$$\Pi[u, i].\alpha = accepted \quad \Longrightarrow \quad \Pi[u, i].k \neq \perp.$$

The game that we will define soon will run $(\perp, m') \leftarrow_{\$} \pi.run(state, \perp)$ on the initial state *state* and an empty message $\perp$. For initiator roles, this first *run* yields $m' \neq \perp$, and for responder roles, it yields $m' = \perp$.

Protocol correctness. For all pairs of sessions which are initialized with (pk_I, sk_I), pk_R, $role = I$, $\alpha = running$ and $k = \perp$ for one session, and (pk_R, sk_R), pk_I, $role = R$, $\alpha = running$ and $k = \perp$ for the other session, the following holds: When the messages produced by $\pi.run$ are faithfully transmitted to the other session, then eventually, both sessions have $\alpha = accepted$ and hold the same key $k \neq \perp$.

Partnering. As a partnering mechanism, we use sound partnering functions, one of the partnering mechanisms suggested by Bellare and Rogaway's [6]. Discussing the specifics, advantages and disadvantages of partnering mechanisms is beyond the scope of this work, we provide a short discussion as well as complete definitions and soundness requirements for partner functions in Appendix C. For the sake of the definition presented in this section, the reader may think of the partnering function $f(u,i)$ as indicating the (first) session (v,j) which derived the same key as (u,i), has a different role than (u,i), and is the intended partner of (u,i). It is a symmetric function, thus partners of sessions, if they exist, are unique.

Definition 23 (AKE Advantage). *For a key exchange protocol $\pi = (kgen, run)$ and a symmetric, monotonic and sound partnering function f, and a number of instances $n \in \mathbb{N}$, we denote the* AKE *advantage by*

$$[\mathtt{ID}_{\{\mathsf{GET},\mathsf{HON}\}}, \mathtt{AKE}^{0,\pi,f}] \circ \mathtt{CKEY}[1..n] \overset{\epsilon_{\mathtt{AKE}}^{\pi,f,n}}{\approx} [\mathtt{ID}_{\{\mathsf{GET},\mathsf{HON}\}}, \mathtt{AKE}^{1,\pi,f}] \circ \mathtt{CKEY}[1..n],$$

where $\mathsf{in}(\mathtt{AKE}^{0,\pi,f}) = \{\mathsf{SET}, \mathsf{CSET}\}$ *and* $\mathsf{in}(\mathtt{AKE}^{1,\pi,f}) = \{\mathsf{GEN}, \mathsf{CSET}\}$ *and for $b \in \{0,1\}$, the output interface* $\mathsf{out}(\mathtt{AKE}^{b,\pi,f})$ *is equal to the set of oracle names* $\{\mathsf{NEWPARTY}, \mathsf{NEWSESSION}, \mathsf{SEND}, \mathsf{CORRUPT}\}$*, where the oracles are defined in Fig. 5.*

Theorem 2 (BR-Secure Key Exchange is Composable). *Let π be a key exchange protocol with partnering function f such that for $n, \lambda \in \mathbb{N}$, their* AKE *advantage is $\epsilon_{\mathtt{AKE}}^{\pi,f,n}$. Let* SYM *be a corruptible key consuming package that is compatible with* AKE*. Then it holds that*

$$\left[\mathtt{AKE}^{0,\pi,f}, \mathtt{SYM}^0[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n] \overset{\epsilon_{\mathtt{BR}}}{\approx} \left[\mathtt{AKE}^{0,\pi}, \mathtt{SYM}^1[1..n]\right] \circ \mathtt{CKEY}^\lambda[1..n],$$

where

$$\begin{aligned}\epsilon_{\mathtt{BR}}(\mathcal{A}) \leq \quad & \epsilon_{\mathtt{AKE}}^{\pi,f,n}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{AKE})}, \mathtt{SYM}^0[1..n]\right]\right) \\ & + n \cdot \epsilon_{\mathtt{SYM}}\left(\mathcal{A} \circ \left[\mathtt{AKE}^{1,\pi,f}, \mathtt{ID}_{\mathsf{out}(\mathtt{SYM}[1..n])}\right] \circ \mathcal{R}\right) \\ & + \epsilon_{\mathtt{AKE}}^{\pi,f,n}\left(\mathcal{A} \circ \left[\mathtt{ID}_{\mathsf{out}(\mathtt{AKE})}, \mathtt{SYM}^1[1..n]\right]\right),\end{aligned}$$

where reduction $\mathcal{R}$ samples $i \leftarrow_\$ \{1,\ldots,n\}$ and implements the package $[\mathtt{M}^0[1..i-1], \overline{\mathtt{ID}_{\mathsf{out}(\mathtt{M})}[i]}, \mathtt{M}^1[i+1..n]]$, where $\mathtt{M}^0 = [\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{SYM}^0] \circ \mathtt{CKEY}^\lambda$ and $\mathtt{M}^1 = [\mathtt{ID}_{\{\mathsf{GEN},\mathsf{CSET}\}}, \mathtt{SYM}^1] \circ \mathtt{CKEY}^\lambda$.

NEWSESSION(u, i, r, v)

assert $PK[u] \neq \bot$, $PK[v] \neq \bot$, $\Pi[u,i] = \bot$
$\Pi[u,i] \leftarrow ($
  $(pk, sk) \leftarrow (PK[u], SK[u])$,
  $peer \leftarrow v$,
  $role \leftarrow r$,
  $\alpha \leftarrow running$,
  $k \leftarrow \bot)$
$(\Pi[u,i], m) \leftarrow\!\!\$\ \pi.run(\Pi[u,i], \bot)$
return m

SEND$(u, i, m))$

assert $\Pi[u,i].\alpha = running$
$(\Pi[u,i], m') \leftarrow\!\!\$\ \pi.run(\Pi(u,i), m)$
if $\Pi[u,i].\alpha \neq accepted$ **then**
  return $(m', \bot)$.
if $\Pi[f(u,i)].\alpha = accepted$ **then**
  return $(m', \Pi[f(u,i)].id)$
$\Pi[u,i].id \leftarrow cntr$
if $H[\Pi[u,i].peer] = 1 \vee f(u,i) \neq \bot$ **then**
  if $b = 1$ **then**
    GEN$[cntr]()$
  else
    SET$[cntr](\Pi[u,i].k)$
else
  CSET$[cntr](\Pi[u,i].k)$
$cntr \leftarrow cntr + 1$
return $(m', \Pi[u,i].id)$

NEWPARTY(u)

assert $PK[u] = \bot$
$(SK[u], PK[u]) \leftarrow\!\!\$\ \pi.kgen$
$H[u] \leftarrow 1$
return $PK[u]$

CORRUPT(u)

$H[u] \leftarrow 0$
return $SK[u]$

Fig. 5: Oracles of the AKE package.

Proof. We observe that Theorem 2 is a direct application of the multi-instance key composition lemma (Lemma 10). Firstly, AKE is a corruptible key generating package as we have that $\mathsf{in}(\mathtt{AKE}^{0,\pi,f}) = \{\mathsf{SET}, \mathsf{CSET}\}$ and $\mathsf{in}(\mathtt{AKE}^{1,\pi,f}) = \{\mathsf{GEN}, \mathsf{CSET}\}$. Also, by definition, SYM is a corruptible key consuming package that is compatible with $\mathtt{AKE}^{b,\pi,f}$.

Discussion of definitional choices. Forward secrecy usually requires a notion of time that cryptographic games are not automatically endowed with and that we have no tools to handle in hand-written proofs. Thus, the corresponding security games rely on a notion of time that is difficult to handle. In the miTLS work and also in our notation of eCK security, instead, it is decided *upon acceptance* whether a session shall be idealized or not. The advantage is that one can check *in the moment of acceptance* whether the preconditions for freshness are satisfied, and this check does not require a notion of time. In our encoding the `CKEY` package then stores either a real or a random key, and when the partner of the session accepts, the partner session inherits these idealization or non-idealization properties. A downside of this encoding is that it is only suitable for protocols with explicit entity authentication (See, e.g., Fischlin, Günther, Schmidt and Warinschi [21]), as in those, the first accepting session is already idealized. In particular, our model does not capture two-flow protocols such as HMQV [28].

Using partner functions instead of session identifiers or key partnering has the advantage that the *at most* condition of Match security defined by Brzuska, Fischlin, Smart, Warinschi and Williams [12] holds syntactically. Thus, one does not need to make probabilistic statements that are external to the games. Note that we made another simplication to the model: Currently, the `CKEY` module and thus `SYM` does not receive information about the timing of acceptance. This can be integrated at the cost of a more complex `CKEY` module.

Acknowledgements. We are deeply indebted to Cas Cremers and Cédric Fournet for extensive feedback on an early draft of our article. We are greatful to Simon Peyton Jones for pointing out the associativity of monadic composition as a generalization of function composition to effectful programs. We thank Giorgia Azzurra Marson and Hoeteck Wee for feedback on the presentation of our toy example in the introduction. We thank Martijn Stam for suggesting to use KEM-DEM composition as one of our application cases. We are greatful to Håkon Jacobsen for feedback on our key exchange definition. We thank Ueli Maurer for an inspiring and helpful discussion on abstraction. Chris Brzuska is greatful to NXP for the support of his chair of IT Security Analysis at TU Hamburg. Much of the research was done while the first author was at Microsoft Research Cambridge and during internships and research visits supported by Microsoft and the EU COST framework. In particular, this work was

supported by an STSM Grant from COST Action IC1306 “Cryptography for Secure Digital Interaction”.

References

1. M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *TCC*, 2004.
2. G. Barthe, J. M. Crespo, Y. Lakhnech, and B. Schmidt. Mind the gap: Modular machine-checked proofs of one-round key exchange protocols. In *EUROCRYPT*, 2015.
3. G. Barthe, M. Daubignard, B. M. Kapron, and Y. Lakhnech. Computational indistinguishability logic. In *ACM CCS*, pages 375–386, 2010.
4. G. Barthe, B. Grégoire, S. Heraud, and S. Z. Béguelin. Computer-aided security proofs for the working cryptographer. In *CRYPTO*, 2011.
5. M. Bellare, A. Boldyreva, and S. Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *EUROCRYPT 2000*. Springer, 2000.
6. M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *STOC*, 1995.
7. M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *EUROCRYPT*, 2006.
8. D. J. Bernstein and T. Lange. Non-uniform cracks in the concrete: The power of free precomputation. In *ASIACRYPT*, 2013.
9. K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P.-Y. Strub. Implementing TLS with verified cryptographic security. In *Security and Privacy*, 2013.
10. K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and S. Zanella Béguelin. Proving the TLS handshake secure (as it is). In *CRYPTO*, 2014.
11. C. Brzuska. *On the foundations of key exchange*. PhD thesis, Darmstadt University of Technology, Germany, 2013.
12. C. Brzuska, M. Fischlin, N. P. Smart, B. Warinschi, and S. C. Williams. Less is more: relaxed yet composable security notions for key exchange. *Int. J. Inf. Sec.*, 12(4), 2013.
13. C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams. Composability of Bellare-Rogaway key exchange protocols. In *ACM CCS*, 2011.
14. C. Brzuska and H. Jacobsen. A modular security analysis of EAP and IEEE 802.11. In *PKC*, 2017.
15. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
16. K. Cohn-Gordon, C. J. F. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *EuroS&P 2017*, 2017.
17. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM J. Comput.*, 2003.
18. C. J. F. Cremers and M. Feltz. Beyond eck: perfect forward secrecy under actor compromise and ephemeral-key reveal. *Des. Codes Cryptography*, 74(1), 2015.
19. A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Z. Béguelin, K. Bhargavan, J. Pan, and J. K. Zinzindohoue. Implementing and proving the TLS 1.3 record layer. In *Security and Privacy*, 2017.
20. B. Dowling, M. Fischlin, F. Günther, and D. Stebila. A cryptographic analysis of the TLS 1.3 handshake protocol candidates. In *ACM CCS*, 2015.

21. M. Fischlin, F. Günther, B. Schmidt, and B. Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *Security and Privacy*, 2016.
22. C. Fournet, M. Kohlweiss, and P.-Y. Strub. Modular code-based cryptographic verification. In *ACM CCS*, 2011.
23. D. Hofheinz and V. Shoup. GNUC: A new universal composability framework. Cryptology ePrint Archive, Report 2011/303, 2011. `http://eprint.iacr.org/2011/303`.
24. D. Hofheinz and V. Shoup. GNUC: A new universal composability framework. *Journal of Cryptology*, 28(3), 2015.
25. H. Jacobsen. *A Modular Security Analysis of EAP and IEEE 802.11*. PhD thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2017.
26. T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *CRYPTO 2012*, 2012.
27. S. P. Jones. Haskell 98 language and libraries: the revised report, 2003.
28. H. Krawczyk. HMQV: A high-performance secure diffie-hellman protocol. In *CRYPTO*. Springer, 2005.
29. H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In *CRYPTO 2013*, 2013.
30. R. Kuesters and M. Tuengerthal. The IITM model: a simple and expressive model for universal composability. Cryptology ePrint Archive, Report 2013/025, 2013. `http://eprint.iacr.org/2013/025`.
31. U. Maurer. Constructive cryptography - a primer (invited paper). In *Financial Cryptography*, 2010.
32. U. Maurer. Constructive cryptography - A new paradigm for security definitions and proofs. In *TOSCA*, 2011.
33. U. Maurer and R. Renner. Abstract cryptography. In *ITCS*, 2011.
34. U. M. Maurer. Indistinguishability of random systems. In *EUROCRYPT*, 2002.
35. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1), 1992.
36. J. C. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3), 2006.
37. J. Müller-Quade and D. Unruh. Long-term security and universal composability. In *TCC*, 2007.
38. T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In *EUROCRYPT*, 2011.
39. P. Rogaway. Formalizing human ignorance. In *VIETCRYPT*, 2006.
40. N. Swamy, C. Hriţcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P.-Y. Strub, M. Kohlweiss, J.-K. Zinzindohoue, and S. Zanella-Béguelin. Dependent types and multi-monadic effects in F*. In *POPL*, 2016.
41. D. Syme, A. Granicz, and A. Cisternino. *Expert $F^{\#}$ 3.0*. Springer, 2012.
42. M. Tofte. Essentials of standard ML modules. In *Advanced Functional Programming, Second International School, Olympia*, 1996.
43. D. Wikström. Simplified universal composability framework. In *TCC*, 2016.

$\texttt{MOD-CPA}^b.\mathsf{ENC}(m)$	$\texttt{MOD-CPA}^b.\mathsf{ENC}(m)$	$\texttt{MOD-CPA}^b.\mathsf{ENC}(m)$	$\texttt{IND-CPA}^b.\mathsf{ENC}(m)$
		if $k = \bot$ **then**	**if** $k = \bot$ **then**
		$k \leftarrow\!\$\ \{0,1\}^n$	$k \leftarrow\!\$\ \{0,1\}^n$
if $b = 0$ **then**	**if** $b = 0$ **then**	**if** $b = 0$ **then**	**if** $b = 0$ **then**
$r \leftarrow\!\$\ \{0,1\}^n$	$r \leftarrow\!\$\ \{0,1\}^n$	$r \leftarrow\!\$\ \{0,1\}^n$	$(r, c) \leftarrow\!\$\ \zeta.enc(k, m)$
$pad \leftarrow \mathsf{EVAL}(r)$	**if** $k = \bot$ **then**		
	$k \leftarrow\!\$\ \{0,1\}^n$		
	$pad \leftarrow \mathsf{prf}(k, r)$	$pad \leftarrow \mathsf{prf}(k, r)$	
$c \leftarrow m \oplus pad$	$c \leftarrow m \oplus pad$	$c \leftarrow m \oplus pad$	
if $b = 1$ **then**	**if** $b = 1$ **then**	**if** $b = 1$ **then**	**if** $b = 1$ **then**
$m' \leftarrow\!\$\ \{0,1\}^n$	$m' \leftarrow\!\$\ \{0,1\}^n$	$m' \leftarrow\!\$\ \{0,1\}^n$	$m' \leftarrow\!\$\ \{0,1\}^n$
$r \leftarrow\!\$\ \{0,1\}^n$	$r \leftarrow\!\$\ \{0,1\}^n$	$r \leftarrow\!\$\ \{0,1\}^n$	$(r, c) \leftarrow\!\$\ \zeta.enc(k, m')$
$pad \leftarrow \mathsf{EVAL}(r)$	**if** $k = \bot$ **then**		
	$k \leftarrow\!\$\ \{0,1\}^n$		
	$pad \leftarrow \mathsf{prf}(k, r)$	$pad \leftarrow \mathsf{prf}(k, r)$	
$c \leftarrow m' \oplus pad$	$c \leftarrow m' \oplus pad$	$c \leftarrow m' \oplus pad$	
return (r, c)	**return** (r, c)	**return** (r, c)	**return** (r, c)

Fig. 6: The left-most column shows the modular game MOD-CPA that uses an oracle EVAL. From the left-most to the second-left column, we inline the code of $\texttt{PRF}^0.\mathsf{EVAL}$. From the second-left to the second-right column, we use Bellare-Rogaway-like code-comparison to see that the key generation can be moved up, as it is the same in both branches of the program. We get from the second-right column to the right-most column by considering the code of our concrete construction ζ.

A IND-CPA and Functional Equivalence

The right-most column of Figure 6 complements the definition of IND-CPA security (Definition 1). We need to show that for $b \in \{0,1\}$, the modular $\texttt{MOD-CPA}^b$ game, composed with the $\texttt{PRF}^0$ game, is functionally equivalent to $\texttt{IND-CPA}^b$. See Figure 6, including the caption, for a proof. For ease of readability, recall that the EVAL oracle in $\texttt{PRF}^0$ runs the code depicted on the right.

$\mathsf{EVAL}(x)$
if $k = \bot$ **then**
$k \leftarrow\!\$\ \{0,1\}^n$
$y \leftarrow \mathsf{prf}(k, x)$
return y

B Hybrid Argument Recipe

Hybrid arguments can be used in various contexts and are the standard technique to reduce multi-instance games to single-instance games. We here write down a general hybrid argument recipe.

Lemma 11 (Hybrid Argument Recipe Lemma). *Let* $\mathtt{Game}^0$, $\mathtt{Game}^1$, $\mathtt{Multi}^0$ *and* $\mathtt{Multi}^1$ *be four packages with* $\mathsf{in}(\mathtt{Game}^0) = \mathsf{in}(\mathtt{Game}^1) = \emptyset$ *and* $\mathsf{out}(\mathtt{Game}^0) = \mathsf{out}(\mathtt{Game}^1)$ *as well as* $\mathsf{in}(\mathtt{Multi}^0) = \mathsf{in}(\mathtt{Multi}^1) = \emptyset$ *and* $\mathsf{out}(\mathtt{Multi}^0) = \mathsf{out}(\mathtt{Multi}^1)$*. Let* $\mathcal{A}$ *be an adversary. Let* n *be a natural number. Let* $\mathtt{H}_0, \ldots, \mathtt{H}_n$ *be games with* $\mathsf{out}(\mathtt{H}_i) = \mathsf{out}(\mathtt{Multi}^1)$*, let* $\mathcal{R}_i$ *be reduction packages with* $\mathsf{out}(\mathcal{R}_i) = \mathsf{out}(\mathtt{Multi}^1)$ *and* $\mathsf{in}(\mathcal{R}_i) = \mathsf{out}(\mathtt{Game}^1)$*, and let* $\mathcal{R}$ *be a package, which samples* $i \leftarrow\!\!\$\, \{0, \ldots, n-1\}$ *and then behaves like* $\mathcal{R}_i$*. Then we need to prove the following:*

Claim 1: *It holds that*

$$\mathtt{Multi}^0 = \mathtt{H}_0 \tag{3}$$

$$\textit{and } \mathtt{Multi}^1 = \mathtt{H}_n \tag{4}$$

Claim 2: *For all* $i \in \{0, \ldots, n-1\}$ *the following holds*

$$\mathcal{R}_i \circ \mathtt{Game}^0 = \mathtt{H}_i \tag{5}$$

$$\textit{and } \mathcal{R}_i \circ \mathtt{Game}^1 = \mathtt{H}_{i+1} \tag{6}$$

If Claim 1 and Claim 2 hold, then the package $\mathcal{R}$ *satisfies*

$$\epsilon_{\mathtt{Multi}}(\mathcal{A}) \leq n \cdot \epsilon_{\mathtt{Game}}(\mathcal{A} \circ \mathcal{R}) \ .$$

Proof. Let $\mathcal{A} : \mathsf{out}(\mathtt{Multi}^0) \to \mathsf{out}(\mathcal{A})$ be an adversary and let $\epsilon_{i,i'}(\mathcal{A}) = |\Pr[0 \leftarrow \mathcal{A} \circ \mathtt{H}_i] - \Pr[1 \leftarrow\!\!\$\, \mathcal{A} \circ \mathtt{H}_{i'}]|$ be the distinguishing advantage between hybrids $\mathtt{H}_i$ and $\mathtt{H}_{i'}$ for $\mathcal{A}$.

1. By definition we have

   $$\mathcal{A} \circ \mathtt{H}_0 \overset{\epsilon_{0,1}(\mathcal{A})}{\approx} \mathcal{A} \circ \mathtt{H}_1 \approx \cdots \approx \mathcal{A} \circ \mathtt{H}_{n-1} \overset{\epsilon_{n-1,n}(\mathcal{A})}{\approx} \mathcal{A} \circ \mathtt{H}_n \ .$$

   From the equation in Claim 1, it follows that $\epsilon_{0,n} = \epsilon_{\mathtt{Multi}}$, i.e.,

   $$\mathcal{A} \circ \mathtt{H}_0 \overset{\epsilon_{\mathtt{Multi}}(\mathcal{A})}{\approx} \mathcal{A} \circ \mathtt{H}_n$$

   By the triangle inequality of $\overset{\epsilon}{\approx}$ we have that

   $$\epsilon_{\mathtt{Multi}}(\mathcal{A}) \leq \epsilon_{0,1}(\mathcal{A}) + \cdots + \epsilon_{n-1,n}(\mathcal{A}) = \sum_{\ell=0}^{n-1} \epsilon_{i,i+1}(\mathcal{A})$$

2. Now, we recall the definition of $\epsilon_{i,i+1}$ and plug in Eq. 5 and 6 from Claim 2:

$$\begin{aligned}\epsilon_{\mathtt{Multi}}(\mathcal{A}) &\leq \sum_{i=0}^{n-1} \epsilon_{i,i+1}(\mathcal{A}) \\ &= \sum_{i=0}^{n-1} \left|\Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathtt{H}_i\right] - \Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathtt{H}_{i+1}\right]\right| \\ &= \sum_{i=0}^{n-1} \left|\Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathcal{R}_i \circ \mathtt{Game}^0\right] - \Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathcal{R}_i \circ \mathtt{Game}^1\right]\right|\end{aligned}$$

Due to the construction of $\mathcal{R}$ we get that $\epsilon_{\mathtt{Multi}}(\mathcal{A})$ is smaller or equal to

$$\sum_{i=0}^{n-1} \left|\Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathcal{R} \circ \mathtt{Game}^0 \mid \ell' = \ell\right] - \Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathcal{R} \circ \mathtt{Game}^1 \;\middle|\; i' = i\right]\right| \tag{7}$$

As the sum iterates over all $i \in \{0, \ldots, n-1\}$, we obtain

$$\sum_{i=0}^{n-1} \Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathcal{R} \circ \mathtt{Game}^b \mid i' = i\right] = \frac{\Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathcal{R} \circ \mathtt{Game}^b\right]}{\frac{1}{n}}. \tag{8}$$

Plugging Eq. 8 into Eq. 7 gives us

$$\begin{aligned}\epsilon_{\mathtt{Multi}}(\mathcal{A}) \leq& n \cdot \left(\Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathcal{R} \circ \mathtt{Game}^0\right] - \Pr\left[1 \leftarrow\!\!\$\ \mathcal{A} \circ \mathcal{R} \circ \mathtt{Game}^1\right]\right) \\ =& n \cdot \epsilon_{\mathtt{Game}}(\mathcal{A} \circ \mathcal{R})\ .\end{aligned}$$

We now use the above recipe to provide a proof of Lemma 8.

Proof. We instantiate the hybrid argument recipe lemma as follows

$$\begin{aligned}\mathtt{Multi}^b &:= \mathtt{M}^b[1..n], \\ \mathtt{Game}^b &:= \mathtt{M}^b.\end{aligned}$$

We define the hybrids $\mathtt{H}_i$ for $0 \leq i \leq n$ as follows

$$\mathtt{H}_i := [\mathtt{M}^0[1..i], \mathtt{M}^1[i+1..n]]$$

Observe, that indeed, $\mathtt{H}_0 = \mathtt{Multi}^0$ and $\mathtt{H}_n = \mathtt{Multi}^1$, so Claim 1 holds. We now specify the reduction package $\mathcal{R}_i[1..n]$ for $1 \leq i \leq n$ with

$\mathsf{in}(\mathcal{R}_i[1..n]) = \mathsf{out}(\mathtt{M})$ and $\mathsf{out}(\mathcal{R}_i[1..n]) = \mathsf{out}(\mathtt{M}[1..n])$. It behaves just as hybrid $\mathtt{H}_i[1..n]$, except for instance i, where $\mathcal{R}_i[1..n]$ forwards the queries to the oracles provided through its input interface (i.e. $\mathtt{M}$). Formally,

$$\mathcal{R}_i[1..n] := [\mathtt{M}^0[1..i-1], \overline{\mathtt{ID}_{\mathsf{out}(\mathtt{M})}[i]}, \mathtt{M}^1[i+1..n]]$$

We now need to show that the reductions $\mathcal{R}_i[1..n]$ satisfy Claim 2. We show Eq. 5, then Eq. 6 follows analogously.

$$\begin{aligned}
\mathcal{R}_i[1..n] \circ \mathtt{M}^0 &= [\mathtt{M}^0[1..i-1], \overline{\mathtt{ID}_{\mathsf{out}(\mathtt{M})}[i]}, \mathtt{M}^1[i+1..n]] \circ \mathtt{M}^0 \\
&= [\mathtt{M}^0[1..i-1], \mathtt{M}^0[i], \mathtt{M}^1[i+1..n]] && \text{multi-ins. interch.} \\
&= [\mathtt{M}^0[1..i], \mathtt{M}^1[i+1..n]] \\
&= \mathtt{H}^i
\end{aligned}$$

Therefore, by Lemma 11 $\mathcal{R}$ satisfies

$$\epsilon_{\mathtt{M}[1..n]}(\mathcal{A}) \leq n \cdot \epsilon_{\mathtt{M}}(\mathcal{A} \circ \mathcal{R}),$$

which concludes the proof of Lemma 8.

C Partner Mechanisms in Key Exchange

Partnering is needed in key exchange protocols to specify which partners derive the same key so that security notions for key exchange can exclude trivial winning strategies, such as revealing the key of a partner session. The original BFWW work showed that for composition, the reduction needs to know the partnering between sessions. In our model, we give this information directly to the adversary (via the indices) and thus also to the reduction. There are many ways to define partnering in key exchange, and partnering in key exchange is an interesting area of research that is not yet fully clarified. For simplicity, we here follow Bellare and Rogaway's formulation of public partnering functions that map sessions merely based on public transcripts [6]. Although partner functions have not been very popular over many years, Brzuska and Jacobsen [14,25] recently re-discovered partnering functions, because properties of partnering functions such as uniqueness can be required to hold syntactically, while they only hold probabilistically for concepts such as session identifiers and key equality. These syntactic properties simplify our composition theorem as we discuss in the end of Section 5. The following definition

is a prose variant of the definition of transcript given by Brzuska and Jacobsen [14,25].

Partner functions are used within key exchange security games and yet, at the same time, the definition of partner functions requires part of the game as already defined. The way out of the circularity is as follows: (1) The partner function can be defined syntactically on transcripts, and the transcript are well-defined also without a partner function. (2) No probabilistic properties on the partner function are required, so that we can consider all powerful adversaries in the consideration of the partner function.

Definition 24 (Transcript). *The public* transcript T *of a key exchange game consists of all* NEWPARTY*,* NEWSESSION *and* SEND *queries by the adversary as well as their answers, except for the answers of* SEND *where only the first component of each answer becomes part of the transcript.*

Definition 25 (Partner Functions). *A* symmetric and monotonic partner function *is a function f, parametrized by a transcript T, that maps pairs (U, i) of sessions to other pairs (V, j) of sessions*

1. $f_T(U,i) = (V,j) \implies f_T(V,j) = (U,i)$, *(symmetric)*
2. $f_T(U,i) = (V,j) \implies f_{T'}(U,i) = (V,j)$ *for all* $T \subseteq T'$. *(monotonic)*

Partnering soundness. For a security analysis based on partner functions to be meaningful, the partner function needs to satisfy certain soundness properties. Briefly, soundness demands that partners should: (1) end up with the same session key, (2) agree upon who they are talking to, (3) have compatible roles, and (4) be unique. However, since we are limiting our attention to symmetric partner functions in this paper, the last requirement follows directly so we omit it.

Definition 26 (Partner Function Soundness). *A partner function is* sound *if the following holds for all transcripts T. If sessions $f_{T'}(U,i) = (V,J)$ then:*

1. $\pi[U,i].\alpha = \pi[V,j].\alpha = \mathit{accepted} \implies \pi[U,i].k = \pi[V,j].k \neq \perp$,
2. $\pi[U,i].\mathit{peer} = \mathit{pk}[V]$, *and* $\pi[V,j].\mathit{peer} = \mathit{pk}[U]$.
3. *(*$\pi[U,i].\mathit{role} = I$, *and* $\pi[V,j].\mathit{role} = R$*) or (*$\pi[U,i].\mathit{role} = R$, *and* $\pi[V,j].\mathit{role} = I$*)*

D Functional Equivalence for MOD-CCA in the Proof of Theorem 1

MOD-CCA			PKE-CCA$^{b,\zeta}$
PKENC(m)	PKENC(m)	PKENC(m)	PKENC(m)
assert $pk \neq \bot$	**assert** $pk \neq \bot$	**assert** $pk \neq \bot$	**assert** $pk \neq \bot$
assert $c = \bot$	**assert** $c = \bot$	**assert** $c = \bot$	**assert** $c = \bot$
$c_1 \leftarrow$ ENCAP()	$k, c_1 \leftarrow \eta.encap(pk)$	$k, c_1 \leftarrow \eta.encap(pk)$	$k, c_1 \leftarrow \eta.encap(pk)$
	~~**if** $0 = 0$ **then**~~		
	SET(k)	~~SET(k)~~	
	~~**else**~~		
	~~GEN()~~		
$c_2 \leftarrow$ ENC(m)	$c_2 \leftarrow$ ENC(m)	~~$k \leftarrow$ GET()~~	
		if $b = 0$ **then**	**if** $b = 0$ **then**
		$c_2 \leftarrow\$ \{0,1\}^{\lvert m \rvert}$	$c_2 \leftarrow\$ \{0,1\}^{\lvert m \rvert}$
		else	**else**
		$c_2 \leftarrow\$ \theta.enc(k, m)$	$c_2 \leftarrow\$ \theta.enc(k, m)$
$c \leftarrow c_1 \lvert\rvert c_2$	$c \leftarrow c_1 \lvert\rvert c_2$	$c \leftarrow c_1 \lvert\rvert c_2$	$c \leftarrow c_1 \lvert\rvert c_2$
return c	**return** c	**return** c	**return** c
PKDEC(c')	PKDEC(c')	PKDEC(c')	PKDEC(c')
assert $pk \neq \bot$	**assert** $pk \neq \bot$	**assert** $pk \neq \bot$	**assert** $pk \neq \bot$
assert $c \neq c'$	**assert** $c \neq c'$	**assert** $c \neq c'$	**assert** $c \neq c'$
$c'_1 \lvert\rvert c'_2 \leftarrow c'$	$c'_1 \lvert\rvert c'_2 \leftarrow c'$	$c'_1 \lvert\rvert c'_2 \leftarrow c'$	$c'_1 \lvert\rvert c'_2 \leftarrow c'$
if $c'_1 = c_1$ **then**	**if** $c'_1 = c_1$ **then**	**if** $c'_1 = c_1$ **then**	
$m \leftarrow$ DEC(c'_2)	$m \leftarrow$ DEC(c'_2)	~~$k \leftarrow$ GET()~~	
		$m \leftarrow \theta.dec(k, c'_2)$	
else	**else**	**else**	
$k' \leftarrow$ DECAP(c'_1)	$k' \leftarrow \eta.decap(sk, c'_1)$	$k' \leftarrow \eta.decap(sk, c'_1)$	$k \leftarrow \eta.decap(sk, c'_1)$
$m \leftarrow \theta.dec(k', c'_2)$	$m \leftarrow \theta.dec(k')$	$m \leftarrow \theta.dec(k')$	$m \leftarrow \theta.dec(k, c'_2)$
return m	**return** m	**return** m	**return** m

Fig. 7: Col. 1-to-2: ENCAP and DECAP of KEM-CCA$^{0,\eta}$ are inlined, highlighted in gray. We cross out code that is not executed. Col. 2-to-3: ENC and DEC of DEM-CCA$^{b,\theta}$ are inlined. Calls to SET and GET do not modify k. Col. 3-to-4: we compare MOD-CCA to KEM-CCA$^{b,\theta}$. They differ only when $c'_1 = c_1$ in the PKDEC oracle. PKENC can only be called once and thus, MOD-CCA.PKDEC decrypts c'_1 with the symmetric key k that was previously encapsulated in the MOD-CCA.PKENC oracle. By correctness of the KEM, $k = \eta.decap(sk, c'_1)$ and $\eta.dec$ uses the same k in both cases.