Programmable encryption and key-dependent messages

Dominique Unruh
University of Tartu

July 27, 2012

This is a preliminary version. The technical content is complete, but discussion and examples will be extended. Feedback is welcome.

Abstract

We present the notion of PROG-KDM security for public-key encryption schemes. This security notion captures both KDM security and revealing of secret keys (key corruptions) in a single definition. This is achieved by requiring the existence of a simulator that can program ciphertexts when a secret key is revealed, i.e., the simulator can delay the decision what plaintext is contained in what ciphertext to the moment where the ciphertext is opened. The definition is formulated in the random oracle model.

We show that PROG-KDM security can be achieved by showing that a natural and practical construction in the ideal cipher model is PROG-KDM secure (hybrid encryption using authenticated CBC encryption).

Contents

1 Introduction

1.1 The problem

Encryption schemes constitute the oldest and arguably the most important cryptographic primitive. The first rigorous security definition was given by Shannon [Sha49] in the information-theoretical case. For public-key encryption, the first definition is probably that of semantic security [GM84] which is equivalent to IND-CPA security [GM84, BDPR98]. However, IND-CPA turned out to be insufficient for many applications, because it only covers passive attacks. As soon as an adversary can ask for decryptions of ciphertexts of his choosing, IND-CPA does not give any guarantees any more. To cover this case, the notion of IND-CCA2 security was introduced [RS92, BDPR98]. Roughly, IND-CCA2 security guarantees that the adversary learns nothing about the plaintext of a ciphertext, even if he can ask for decryptions of arbitrary other ciphertexts.

IND-CCA2 security is probably the most popular security notion for general purpose public-key encryption schemes. One reason is that IND-CCA2 secure encryption gives us "worry-free encryption", that is, an IND-CCA2 secure encryption should be usable in any setting where we would expect some idealized encryption scheme to work. For example, in an ideal world, we would not expect data to be leaked from a ciphertext if we can decrypt a different ciphertext. IND-CCA2 security guarantees precisely this.

When striving for a general-purpose encryption scheme (e.g., one that is to be included in an industrial standard), "worry-free encryption" is an important goal, as we do not know in advance in which situations the encryption scheme is to be used. Unfortunately, over time it has been recognized that IND-CCA2 secure encryption is not fully "worry-free". There are two scenarios in which IND-CCA2 security fails to give the intuitively expected guarantees:

- **Key-dependent messages (KDM):** IND-CCA2 security does not guarantee security if the plaintexts that are encrypted are themselves functions of the secret keys [BRS02]. At a first glance, this may seem exotic, but it may happen in a number of natural use cases. For example, hard-disk encryption may encrypt the swap partition, which in turn contains the secret key for decrypting the hard-disk. Furthermore, KDM can also occur unintentionally in a complex protocol where secret keys are exchanged between parties. Finally, some protocols even intentionally create cyclic key-message-dependencies (key-cycles) [CL01]. In all these cases, IND-CCA2 security does not guarantee anything. But obviously, if we wish encryption to be "worry-free", we have to guarantee security even in these situations.

- **Revealing of secret keys (key corruptions):** When a secret key is revealed, then obviously all plaintexts encrypted with respect to that secret key will be revealed, too. However, ciphertexts encrypted with unrevealed secret keys should not reveal any information. There is a surprising subtlety due to which IND-CCA2 does not fully guarantee this (noticed, according to [DNRS03], already in 1985 in the context of Byzantine agreement). For example, assume there are n correlated plaintexts m_i (e.g., shares of a secret sharing scheme), all encrypted with different keys, and an adversary can ask to see $n/2$ of these keys. Intuitively, we would expect that he does not learn more about the shares than by just requesting $n/2$ shares. However, using IND-CPA (or IND-CCA2) security alone, we cannot prove this fact!

  The obvious way how we would prove such a fact is to consider two games: the original game in which each ciphertext c_i is an encryption of m_i, and the modified game, where only the ciphertexts c_i opened by the adversary contain m_i, all others contain 0. (In the second game, it is then obvious that the adversary learns nothing beyond $n/2$ shares.) If we knew in advance, which c_i are to be opened, IND-CPA would guarantee indistinguishability of these games. However, we have to give the c_i to the adversary before knowing which ones will be opened. Moreover, which c_i the adversary opens may even depend on the values of the c_i itself. This circular dependency obviates the use of IND-CPA for proving that the adversary learns nothing beyond $n/2$ shares.

  In fact, there are three variants of this problem: the problem we just described is the *receiver selective opening problem.* Tightly related is the *sender selective opening problem*, in which the adversary does not request secret keys, but gets the randomness and plaintexts that were used to produce the ciphertexts. And the *selective decommitment problem*, in which commitments are sent instead of ciphertexts, and opened instead of decrypted.

  In our setting, we will be concerned with *receiver selective openings* only, because sending secret keys seems to be a natural protocol operation and thus should be covered by "worry-free" encryption.

Thus, if our goal is a general-purpose "worry-free" encryption, we should strive to satisfy a security definition that deals gracefully both with KDM and with revealing of secret keys. In this work, we will present such a definition, dubbed PROG-KDM, for public-key encryption (in the random-oracle model) and show that it can be instantiated by natural practical constructions.

1.2 Prior approaches

We first give a very short survey of existing approaches towards definitions of secure encryption schemes with respect to KDM, and with respect to revealing secret keys:

- **KDM:** KDM security was first introduced by Black, Rogaway, and Shrimpton [BRS02] who defined IND-KDM, an extension of IND-CPA that allows the adversary to produce key-dependent messages. Being based on IND-CPA, IND-KDM does not protect against active attacks (chosen ciphertext attacks). They instantiate the definition in the random-oracle model. Later, Boneh, Halevi, Hamburg, and Ostrovsky [BHHO08] give an instantiation in the standard model. The definition of IND-KDM is extended to active attacks by Camenisch, Chandran, and Shoup [CCS09], who define and instantiate KDM-CCA2, a natural merger of IND-KDM and IND-CCA2. Furthermore, two additional definitions of KDM-security have been proposed, DKDM [BPS07] (for symmetric encryption, but easy to generalize to the public-key setting [BDU08]), and adKDM [BDU08]. These two definitions simultaneously cover KDM and secret key revealing, they will be discussed in the next paragraph.

- **Secret key revealing:** The first solution to the selective opening problem was presented by Canetti, Feige, Goldreich, and Naor [CFGN96]. They gave a definition for encryption schemes (in a generalized, interactive sense) called non-committing encryption (NCE) which has the following property: After encrypting a message, it is possible to "lie" about the message. That is, after encrypting a message m, and after the communication c has been observed, for any message m', a simulator can produce a fake internal view of the communicating parties that is consistent with c being an encryption of m'. In the specific case of non-interactive non-committing encryption (NINCE), this means that given a ciphertext c, it is possible to produce randomness such that encrypting m' would lead to c (sender-NINCE), and to produce a secret key such that decrypting c would lead to m' (receiver-NINCE).

  Why is receiver-NINCE a solution to the problem of secret key revealing? Recall our explanations why IND-CPA security is not sufficient to show that the adversary cannot recover more than $n/2$ shares when opening $n/2$ encryptions of shares. The problem was that with IND-CPA, we needed to produce fake encryptions for those ciphertexts that were not opened, and real encryptions (encryptions of the shares) for those that were opened. With NCE, in the modified game, we just produce fake encryptions for all ciphertexts, and only when a ciphertext c_i is opened, we retroactively make it look like a ciphertext to the share s_i. NCE guarantees that this is indistinguishable from the original situation where the shares are encrypted normally. Yet, in the modified game, only $n/2$ shares are ever accessed, hence the adversary cannot learn more than he could from $n/2$ shares.

  The problem with receiver-NINCE is that it is impossible to achieve unless we restrict the total length of all encrypted plaintexts to at most the length of the secret key (Nielsen [Nie02]): If by revealing a different secret key sk', one can open a given ciphertext c as a different message m', then sk' must contain all the information about m', hence sk' cannot be shorter than m'. Nielsen [Nie02] also showed that *in the random oracle model*, it is possible to construct receiver-NINCE schemes without any limitation on the total plaintext length. The basic observation is that instead of making sk' contain the information about m', we can re-program the random oracle (change the random oracle's so-far unaccessed values) so that decrypting c will lead to m'. (This observation also underlies our approach, see below.)

An alternative approach to the problem of revealing secret keys is to directly model the selective opening problem, leading to the definitions of IND-SO-CPA and SIM-SO-CPA (formulated in [BHK12] for the sender selective opening case, but they can be easily reformulated for the receiver selective opening case, too). IND-SO-CPA security directly models a game in which first the correlated plaintexts are chosen according to some (adversarially chosen) distribution, and then the adversary picks a subset of the ciphertexts to be opened. Then the adversary is either given all the remaining (unopened) plaintexts, or some fresh plaintexts (chosen randomly but consistently with the already opened plaintexts). IND-SO-CPA guarantees that the adversary cannot tell the difference.

SIM-SO-CPA also models a game with correlated plaintexts according to some adversarially chosen distribution, and allows the adversary to open a subset of the ciphertexts. Then, essentially, SIM-SO-CPA says that whatever the adversary can compute from the ciphertexts and the openings, a simulator can also compute from the opened plaintexts alone.

So both IND-SO-CPA and SIM-SO-CPA seem to guarantee that the adversary does not learn anything about the unopened plaintexts (except for what can be deduced from the opened ones). However, also IND-SO-CPA and SIM-SO-CPA (suitably adapted to the receiver selective opening case) are not sufficient for "worry-free" encryption.

In the case of IND-SO-CPA, there is a simple example: The protocol produces n shares s_i of a secret s using an $(n/2+1)$-out-of-n secret sharing scheme. Then each share s_i is encrypted and the resulting ciphertext c_i is sent to the adversary. The adversary may select $n/2$ of the ciphertexts which are subsequently opened. The adversary wins if he can reconstruct s. Assume that the secret sharing scheme has the property that $n/2$ shares already information-theoretically *determine* the secret s and all other shares (although $n/2+1$ shares are required to actually *compute* s). A "worry-free encryption" should make sure that the adversary cannot reconstruct s, since he only sees $n/2$ shares. But IND-SO-CPA only guarantees that after opening $n/2$ shares, we cannot distinguish the remaining $n/2$ shares s_i from freshly chosen shares s'_i that are consistent with the opened shares s_i. But since $n/2$ shares already information-theoretically determine all the other shares, consistency between the s'_i and the $n/2$ opened s_i implies that $s'_i = s_i$ for all i! IND-SO-CPA implies that s_i and s'_i are indistinguishable; this is trivial as they are equal. So IND-SO-CPA does not give any helpful guarantees in this case.[1]

To show the limitations of SIM-SO-CPA, we need a slightly more elaborate example protocol: Again, we produce shares s_i of a secret s. But now c_i is not an encryption of s_i, but an encryption of the tuple $m_i := (s_i, H(c_1, \dots, c_{i-1}))$ for some hash function H.[2] In this case SIM-SO-CPA cannot be applied: The definition requires the adversary to output a distribution $\mathcal{D}$ which then produces the plaintexts m_i, which are then encrypted to produce the ciphertexts c_i. However, $\mathcal{D}$ has no access to the ciphertexts c_i, so the plaintexts m_i cannot depend on the ciphertexts c_i. So SIM-SO-CPA cannot be use to show the security of the example protocol, and thus should not be considered sufficient if we want "worry-free encryption".

Finally, in an attempt to combine KDM-security with the possibility of revealing secret keys, Backes, Pfitzmann, and Scedrov [BPS07] suggested DKDM security (for symmetric encryption). However, DKDM has the limitation that one is not allowed to reveal a key as soon as it has been used for encrypting. This was resolved by Backes, Dürmuth, and Unruh [BDU08], who presented adKDM security (for the public-key case) and showed that it is satisfied by the OAEP encryption scheme [BR94] in the random oracle model. adKDM security allows the adversary to perform an arbitrary sequence of encryptions and decryptions, including encryptions using KDMs. Some of the encryptions may be marked by the adversary as "challenge encryptions". adKDM then guarantees that the adversary cannot tell whether these challenge encryptions are performed honestly, or whether zero-plaintexts are encrypted. Unfortunately, this is not sufficient to solve the selective opening problem (receiver case): As in the case of IND-CPA security, it is not clear how to show that, given n ciphertexts c_i with correlated plaintexts and $n/2$ adaptively chosen openings, the adversary cannot learn more than $n/2$ plaintexts. The reason is that adKDM requires us to choose during encryption which ciphertexts are supposed to be challenge ciphertexts, and these may not be opened. But like in the case of IND-CPA security, we do not know during encryption which ciphertexts will be opened.

To summarize, none of the aforementioned definitions fully solves the problem of KDMs and of revealing secret keys. IND-KDM and KDM-CCA2 only cover KDM security. DKDM and adKDM cover both KDM and revealing secret keys, but are restricted in their applicability in the case of selective openings. IND-SO-CPA and SIM-SO-CCA are restricted in the same fashion. Only receiver-NINCE seems to fully solve the problem of selective openings (because it allows us to "reprogram" the ciphertexts to contain what we want when opening, removing the need to know any plaintexts that are not opened). So the idea behind receiver-NINCE seems to be our best bet for dealing with revealing secret keys. However, existing definitions of receiver-NINCE do not deal with active attacks (CCA

[1]We stress that this does not constitute a proof that IND-SO-CPA security does not imply security of the example protocol. However, it demonstrates that at least the natural way of applying IND-SO-CPA to the present situation fails. The same holds for our arguments concerning SIM-SO-CPA and adKDM below.

[2]We did not encrypt $m_i := (s_i, c_1, \dots, c_{i-1})$, because this would lead to an exponential blow-up of the size of c_i.

security) and nor with KDMs. Also, receiver-NINCE can (unless we impose strong limits on the total size of the plaintexts) only be implemented in the random oracle model.

In this work, we will extend the idea of receiver-NINCE to incorporate active attacks and KDMs by defining the notion of PROG-KDM in which a simulator is able to change ("program") the plaintexts of ciphertexts when revealing the secret key. We will, however, not remove the need for the random oracle. We believe that finding a definition for "worry-free encryption" that works in the standard model is a highly revelant open problem.

1.3 Our contribution

We define a notion of security of encryption schemes in the random oracle model, PROG-KDM (Section 2). The definition covers key dependent messages and the revealing of secret keys, under active attacks. Security is modeled by ensuring that the real world where encryptions are done honestly should be indistinguishable from a simulated world in which the simulator is able to program ciphertexts to have a certain content at the moment when he has to produce the secret key. This ensures that the adversary can learn nothing about unopened plaintexts in the ideal model, and thus, by indistinguishability, also not in the real model. The definition follows very basically the idea of receiver-NINCE in the programmable random oracle model [Nie02], however adding KDM security (and nested encryptions) to the definition turns out to be non-trivial.

Furthermore, we show that our definition can be met. We show that a very natural construction satisfies PROG-KDM security in the ideal cipher model (see, e.g., [CPS08]). (The construction is given in Section 3, and the proof in Section 4.) More precisely, we consider a hybrid encryption scheme where the key-encapsulation mechanism is an arbitrary CCA-secure key-encapsulation mechanism, and where we use CBC mode [EMST76] for the symmetric part (authenticated with a one-time MAC, as CBC would not even be IND-CCA2 secure otherwise). We stress that this construction is very practical and natural, in fact, it would be a natural answer when asked for a simple and practically efficient encryption scheme.

2 Definition of PROG-KDM

As explained in the introduction, we wish to define a security notion for encryption schemes that allows us to program the ciphertexts, i.e., the real world where encryptions are done honestly should be indistinguishable from a simulated world in which the simulator is able to program ciphertexts to have a certain content at the moment when he has to produce the secret key.

In the standard model, this is obviously impossible: The ciphertexts are already fixed when revealing a secret key, so the secret key needs to carry the information about which plaintext the ciphertext should be programmed to have. Since we do not impose an upper bound on the length and number of ciphertexts, the secret key would have to carry an unbounded amount of information.

However, we can get around this impossibility if we work in the random oracle model. (In the following, we use the word random oracle for any oracle chosen uniformly out of a family of functions; thus also the ideal cipher model (see [CPS08], going back to [Sha49]) or the generic group model [Sho97] fall under this term. The "standard" random oracle [BR93] which is a uniformly randomly chosen function from the set of all functions we call "random hash oracle" for disambiguation.)

In the random oracle model, we can see the random oracle as a function that is initially undefined, and upon access, the function table is populated as needed (lazy sampling). This enables the following proof technique: When a certain random oracle location has not been queried yet, we may set it to a particular value of our choosing (this is called "programming the random oracle"). In our case this can be used to program a ciphertext c: As long as we make sure that the adversary has not yet queried the random oracle at the locations needed for decrypting c (e.g., because to find these locations he needs to know the secret key), we can still change the value of the oracle at these locations. This in turn may allow us to change the value that c decrypts to.

Summarizing, we look for an encryption scheme with the following property: There is a strategy for producing (fake) keys and ciphertexts, and for reprogramming the random oracle (we will call this strategy the "ciphertext simulator"), such that the following two things are indistinguishable: (a) (Normally) encrypting a value m, sending the resulting ciphertext c, and then sending the decryption key. (b) Producing a fake ciphertext c. Choosing m. And sending the decryption key.

Formally defining the required security property (PROG-KDM) turns out to be more complex than one might expect, though. We cannot just state that the ciphertext simulator is indistinguishable from an honest encryption oracle. The ciphertext simulator has a completely different interface from the honest encryption oracle. In particular, it expects the plaintext when being asked for the secret key, while the encryption oracle would expect these upon encryption. To cope with this problem, we define two "wrappers", the real and the fake challenger. The real challenger essentially gives us access to the encryption algorithm while the fake challenger, although it expects the plaintexts during encryption (to be indistinguishable from the real challenger), uses the plaintexts only when the decryption key is to be produced. These two challengers should then be indistinguishable. (The challengers additionally make sure that the adversary does not perform any forbidden queries such as submitting a ciphertext for decryption that was produced by the challenger.)

We first define the real challenger. The real challenger needs to allow us to query the encryption and decryption keys, to perform encryptions and decryptions, and to give us access to the underlying random oracle. However, if we only have these queries, situations like the following would lead to problems: The adversary wishes to get $Enc(ek_1, Enc(ek_2, m))$. We do not wish the adversary to have to request $Enc(ek_2, m)$ first and then resubmit it for the second encryption, because this would reveal $Enc(ek_2, m)$, and we might later wish to argue that $Enc(ek_2, m)$ stays secret. To be able to model such setting, we need to allow the adversary to evaluate sequences of queries without revealing their outcome. For this, we introduce queries such as $R := \mathtt{enc}_{\mathtt{ch}}(N, R_1)$. This means: Take the value from register R_1, encrypt it with the key with index $N \in \{0,1\}^*$, and store the result in register R. Also, we need a query to apply arbitrary functions to registers: $R := \mathtt{eval}_{\mathtt{ch}}(C, R_1, \dots, R_n)$ applies the circuit C to registers $R_1, \dots, R_n$. (This in particular allows us to load a fixed value into a register by using a circuit with zero inputs ($n = 0$). Finally, we have a query $\mathtt{reveal}_{\mathtt{ch}}(R_1)$ that outputs the content of a register.

Formally, the definition of the real challenger is the following:

Definition 1 (Real challenger) *Fix an oracle $\mathcal{O}$ and an encryption scheme (K, E, D) relative to that oracle. The* real challenger RC *is an interactive machine defined as follows.* RC *has access to the oracle $\mathcal{O}$.* RC *maintains a family $(ek_N, dk_N)_{N \in \{0,1\}^*}$ of key pairs (initialized as $(ek_N, dk_N) \leftarrow K(1^\eta)$ upon first use), a family $(reg_N)_{N \in \{0,1\}^*}$ of registers (initially all $reg_N = \bot$), and a family of sets $cipher_N$ (initially empty).* RC *responds to the following queries (when no answer is specified, the empty word is returned):*

- $R := \mathtt{getek}_{\mathtt{ch}}(N)$*:* RC *sets $reg_R := ek_N$.*
- $R := \mathtt{getdk}_{\mathtt{ch}}(N)$*:* RC *sets $reg_R := dk_N$.*
- $R := \mathtt{eval}_{\mathtt{ch}}(C, R_1, \dots, R_n)$ *where C is a Boolean circuit:*[3] *Compute $m := C(reg_{R_1}, \dots, reg_{R_n})$ and set $reg_R := m$.*
- $R := \mathtt{enc}_{\mathtt{ch}}(N, R_1)$*: Compute $c \leftarrow E^{\mathcal{O}}(ek_N, reg_{R_1})$, append c to $cipher_N$, and set $reg_R := c$.*
- $\mathtt{oracle}_{\mathtt{ch}}(x)$*: Return $\mathcal{O}(x)$.*
- $\mathtt{dec}_{\mathtt{ch}}(N, c)$*: If $c \in cipher_N$, return* `forbidden` *where* `forbidden` *is a special symbol (different from any bitstring and from a failed decryption $\bot$). Otherwise, invoke $m \leftarrow D^{\mathcal{O}}(dk_N, c)$ and return m.*
- $\mathtt{reveal}_{\mathtt{ch}}(R_1)$*: Return reg_{R_1}.*

Here N and c range over bitstrings, R ranges over bitstrings with $reg_R = \bot$ and the R_i range over bitstrings R with $reg_{R_i} \neq \bot$.

Notice that the fact that we can do "hidden evaluations" of complex expressions, also covers KDM security (security under key-dependent messages): We can make a register contain the computation of, e.g., $Enc(ek, dk)$ where dk is the decryption key corresponding to ek.

We now proceed to define the fake challenger. The fake challenger responds to the same queries, but computes the plaintexts as late as possible. In order to do this, upon a query such as $R := \mathtt{enc}_{\mathtt{ch}}(N, R_1)$, the fake challenger just stores the symbolic expression "$\mathtt{enc}_{\mathtt{ch}}(N, R_1)$" in register R (instead of an actual ciphertext). Only when the content of a register is to be revealed, the bitstrings are recursively computed (using the function `FCRetrieve` below) by querying the ciphertext simulator. Thus, before defining the fake challenger, we first have to define formally what a ciphertext simulator is:

Definition 2 (Ciphertext simulator) *A* ciphertext simulator CS *for an oracle $\mathcal{O}$ is an interactive machine that responds to the following queries: $\mathtt{fakeenc}_{\mathtt{cs}}(R, l)$, $\mathtt{dec}_{\mathtt{cs}}(c)$,*

[3] Note that from the description of a circuit, it is possible to determine the length of its output. This will be important in the definition of `FCLen` below.

$\mathtt{enc}_{\mathtt{cs}}(R, m)$, $\mathtt{getek}_{\mathtt{cs}}()$, $\mathtt{getdk}_{\mathtt{cs}}()$, *and* $\mathtt{program}_{\mathtt{cs}}(R, m)$. *Any query is answered with a bitstring (except* $\mathtt{dec}_{\mathtt{cs}}(c)$ *which may also return* $\bot$*). A ciphertext simulator runs in polynomial-time in the total length of the queries. A ciphertext simulator is furthermore given access to an oracle* $\mathcal{O}$*. The ciphertext simulator is also allowed to program* $\mathcal{O}$ *(that is, it may perform assignments of the form* $\mathcal{O}(x) := y$*). Furthermore, the ciphertext simulator has access to the list of all queries made to* $\mathcal{O}$ *so far.*[4]

The interesting queries here are $\mathtt{fakeenc}_{\mathtt{cs}}(R, l)$ and $\mathtt{program}_{\mathtt{cs}}(R, m)$. A $\mathtt{fakeenc}_{\mathtt{cs}}(R, l)$-query is expected to return a fake ciphertext for an unspecified plaintext of length l (associated with a handle R). And a subsequent $\mathtt{program}_{\mathtt{cs}}(R, m)$-query with $|m| = l$ is supposed to program the random oracle such that decrypting c will return m. The ciphertext simulator expects to get all necessary $\mathtt{program}_{\mathtt{cs}}(R, m)$-queries directly after a $\mathtt{getdk}_{\mathtt{cs}}()$-query revealing the key. (Formally, we do not impose this rule, but the PROG-KDM does not guarantee anything if the ciphertext simulator is not queried in the same way as does the fake challenger below.) We stress that we allow to first ask for the key and then to program. This is needed to handle key dependencies, e.g., if we wish to program the plaintext to be the decryption key. The definition of the fake challenger will make sure that although we reveal the decryption key before programming, we do not use its value for anything but the programming until the programming is done.

Note that we do not fix any concrete behavior of the ciphertext simulator since our definition will just require the existence of some ciphertext simulator.

We can now define the real challenger together with its recursive retrieval function `FCRetrieve`:

Definition 3 (Fake challenger) *Fix an oracle* $\mathcal{O}$*, a length-regular encryption scheme* (K, E, D) *relative to that oracle, and a ciphertext simulator* CS *for* $\mathcal{O}$*. The* fake challenger FC *for* CS *is an interactive machine defined as follows.* FC *maintains the following state:*

- *A family of instances* $(\mathrm{CS}_N)_{N\in\{0,1\}^*}$ *of* CS *(initialized upon first use). Each ciphertext simulator is given (read-write) oracle access to* $\mathcal{O}$*.*
- *A family* $(reg_R)_{R\in\{0,1\}^*}$ *of registers (initially all* $reg_R = \bot$*). Registers* reg_N *are either undefined (*$reg_N = \bot$*), or bitstrings, or queries (written "*$\mathtt{getek}_{\mathtt{ch}}(N)$*" or "*$\mathtt{getdk}_{\mathtt{ch}}(N)$*" or "*$\mathtt{eval}_{\mathtt{ch}}(C, R_1, \dots, R_n)$*" etc.).*
- *A family* $(cipher_N)_{N\in\{0,1\}^*}$ *of sets of bitstrings. (Initially all empty.)*

FC *answers to the same queries as the real challenger, but implements them differently:*

- $R := \mathtt{getek}_{\mathtt{ch}}(N)$ *or* $R := \mathtt{getdk}_{\mathtt{ch}}(N)$ *or* $R := \mathtt{eval}_{\mathtt{ch}}(C, R_1, \dots, R_n)$ *or* $R := \mathtt{enc}_{\mathtt{ch}}(N, R_1)$*: Set* $reg_R :=$ *"*$\mathtt{getek}_{\mathtt{ch}}(N)$*" or* $reg_R :=$ *"*$\mathtt{getdk}_{\mathtt{ch}}(N)$*" or* $reg_R :=$ *"*$\mathtt{eval}_{\mathtt{ch}}(C, R_1, \dots, R_n)$*" or* $reg_R :=$ *"*$\mathtt{enc}_{\mathtt{ch}}(N, R_1)$*", respectively.*
- $\mathtt{dec}_{\mathtt{ch}}(N, c)$*: If* $c \in cipher_N$*, return* `forbidden`*. Otherwise, query* $\mathtt{dec}_{\mathtt{cs}}(c)$ *from* CS_N *and return its response.*
- $\mathtt{oracle}_{\mathtt{ch}}(x)$*: Return* $\mathcal{O}(x)$*.*

[4]Our scheme will not make use of the list of the queries to $\mathcal{O}$, but for other schemes this additional power might be helpful.

- $\texttt{reveal}_{\texttt{ch}}(R_1)$*: Compute* $m \leftarrow \texttt{FCRetrieve}(R_1)$*. (*$\texttt{FCRetrieve}$ *is defined below in Definition 4.) Return* m*.*

Definition 4 (Retrieve function of FC**)** *The* retrieve function $\texttt{FCRetrieve}$ *has access to the registers* reg_R *and the ciphertext simulators* CS_N *of* FC*. It additionally stores a family* $(plain_N)_{N\in\{0,1\}^*}$ *of lists between invocations (all* $plain_N$ *are initially empty lists).* $\texttt{FCRetrieve}$ *takes an argument* R *(with* $reg_R \neq \bot$*) and is recursively defined as follows:*

- *If* reg_R *is a bitstring, return* reg_R*.*
- *If* $reg_R =$ *"*$\texttt{getek}_{\texttt{ch}}(N)$*": Query* CS_N *with* $\texttt{getek}_{\texttt{cs}}()$*. Store the answer in* reg_R*. Return* reg_R*.*
- *If* $reg_R =$ *"*$\texttt{eval}_{\texttt{ch}}(C, R_1, \dots, R_n)$*": Compute* $m_i := \texttt{FCRetrieve}(R_i)$ *for* $i = 1, \dots, n$*. Compute* $m' := C(m_1, \dots, m_n)$*. Set* $reg_R := m'$*. Return* m'*.*
- *If* $reg_R =$ *"*$\texttt{enc}_{\texttt{ch}}(N, R_1)$*" and there was no* $\texttt{getdk}_{\texttt{cs}}()$*-query to* CS_N *yet: Compute* $l := \texttt{FCLen}(R_1)$*. (*$\texttt{FCLen}$ *is defined in Definition 6 below.) Query* CS_N *with* $\texttt{fakeenc}_{\texttt{cs}}(R, l)$*. Denote the answer with* c*. Set* $reg_R := c$*. Append* $(R \mapsto R_1)$ *to the list* $plain_N$*. Append* c *to* $cipher_N$*. Return* c*.*
- *If* $reg_R =$ *"*$\texttt{enc}_{\texttt{ch}}(N, R_1)$*" and there was a* $\texttt{getdk}_{\texttt{cs}}()$*-query to* CS_N*: Compute* $m := \texttt{FCRetrieve}(R_1)$*. Query* CS_N *with* $\texttt{enc}_{\texttt{cs}}(R, m)$*. Denote the answer with* c*. Set* $reg_R := c$*. Append* $(R \mapsto R_1)$ *to* $plain_N$*. Append* c *to* $cipher_N$*. Return* c*.*
- *If* $reg_R =$ *"*$\texttt{getdk}_{\texttt{ch}}(N)$*": Query* CS_N *with* $\texttt{getdk}_{\texttt{cs}}()$*. Store the answer in* reg_R*. If this was the first* $\texttt{getdk}_{\texttt{cs}}(N)$*-query for that value of* N*, do the following for each* $(R' \mapsto R'_1) \in plain_N$ *(in the order they occur in the list):*
  - *Invoke* $m := \texttt{FCRetrieve}(R'_1)$*.*
  - *Send the query* $\texttt{program}_{\texttt{cs}}(R', m)$ *to* CS_N*.*

  Finally, return reg_R*.*

The retrieve function uses the auxiliary function $\texttt{FCLen}$ that computes what length a bitstring associated with a register should have. This function only makes sense if we require the encryption scheme to be length regular, i.e., the length of the output of the encryption scheme depends only on the lengths of its inputs.

Definition 5 (Length regular encryption scheme) *An encryption scheme* (K, E, D) *is* length-regular *if there are functions* $\ell_{ek}, \ell_{dk}, \ell_c$ *such that for all* $\eta \in \mathbb{N}$ *and all* $m \in \{0,1\}^*$ *and for* $(ek, dk) \leftarrow K(1^\eta)$ *and* $c \leftarrow E(ek, m)$ *we have* $|ek| = \ell_{ek}(\eta)$ *and* $|dk| = \ell_{dk}(\eta)$ *and* $|c| = \ell_c(\eta, |m|)$ *with probability* 1*.*

Definition 6 (Length function of FC**)** *The* length function $\texttt{FCLen}$ *has (read-only) access to the registers* reg_R *of* FC*.* $\texttt{FCLen}$ *takes an argument* R *(with* $reg_R \neq \bot$*) and is recursively defined as follows:*

- *If* reg_R *is a bitstring, return* $|reg_R|$*.*
- *If* $reg_R =$ *"*$\texttt{eval}_{\texttt{ch}}(C, R_1, \dots, R_n)$*": Return the length of the output of the circuit* C*. (Note that the length of the output of a Boolean circuit is independent of its arguments.)*

- *If $reg_R =$ “$\mathtt{getek}_{\mathtt{ch}}(N)$” or $reg_R =$ “$\mathtt{getdk}_{\mathtt{cs}}(N)$”: Let ℓ_{ek} and ℓ_{dk} be as in Definition 5. Return $\ell_{ek}(\eta)$ or $\ell_{dk}(\eta)$, respectively.*
- *If $reg_R =$ “$\mathtt{enc}_{\mathtt{ch}}(N, R_1)$”: Let ℓ_c be as in Definition 5. Return $\ell_c(\eta, \mathtt{FCLen}(R_1))$.*

We are now finally ready to define PROG-KDM security:

Definition 7 (PROG-KDM security) *A length-regular encryption scheme (K, E, D) (relative to an oracle $\mathcal{O}$) is PROG-KDM secure iff there exists a ciphertext simulator* CS *such that for all polynomial-time oracle machines $\mathcal{A}$,*[5] $\Pr[\mathcal{A}^{\mathrm{RC}}(1^\eta) = 1] - \Pr[\mathcal{A}^{\mathrm{FC}}(1^\eta) = 1]$ *is negligible in η. Here* RC *is the real challenger for (K, E, D) and $\mathcal{O}$ and* FC *is the fake challenger for* CS *and $\mathcal{O}$. Notice that $\mathcal{A}$ does not directly query $\mathcal{O}$.*

3 PROG-KDM via hybrid encryption

Setup. Let MAC be a one-time message authentication code (MAC) with key-space $\{0,1\}^{\ell^K_{\mathrm{MAC}}}$ and that outputs tags of length ℓ^T_{MAC} (see [CS03] for the definition of one-time MACs).

Fix an ideal cipher $\mathcal{O}_{\mathrm{IC}}$ with key space $\{0,1\}^{\ell^K_{\mathrm{IC}}}$ and message space $\{0,1\}^{\ell^M_{\mathrm{IC}}}$. Formally, $\mathcal{O}_{\mathrm{IC}}$ is a uniformly chosen function from $\{+,-\} \times \{0,1\}^{\ell^K_{\mathrm{IC}}} \times \{0,1\}^{\ell^M_{\mathrm{IC}}}$ to $\{0,1\}^{\ell^M_{\mathrm{IC}}}$ such that $\mathcal{O}_{\mathrm{IC}}(+, k, \cdot)$ is a permutation on $\{0,1\}^{\ell^M_{\mathrm{IC}}}$ for each $k \in \{0,1\}^{\ell^K_{\mathrm{IC}}}$ and $\mathcal{O}_{\mathrm{IC}}(-, k, \cdot)$ its inverse. Intuitively, this means that $\mathcal{O}_{\mathrm{IC}}$ is a family of permutations on $\{0,1\}^{\ell^M_{\mathrm{IC}}}$, indexed by a key $k \in \{0,1\}^{\ell^K_{\mathrm{IC}}}$, such that one can query the permutation and its inverse (by using $+$ or $-$ as the first argument to $\mathcal{O}_{\mathrm{IC}}$).

Assume that ℓ^K_{IC} and ℓ^M_{IC} are superlogarithmic in the security parameter (such that $2^{-\ell^K_{\mathrm{IC}}}$ and $2^{-\ell^M_{\mathrm{IC}}}$ are negligible).

Let $(K_{\mathrm{KEM}}, E_{\mathrm{KEM}}, D_{\mathrm{KEM}})$ be a CCA-secure key-encapsulation mechanism (KEM) with message space $\{0,1\}^{\ell^K_{\mathrm{MAC}}+\ell^K_{\mathrm{IC}}}$. (See [CS03] for a definition of CCA-secure KEM.) Assume that the length of a key output by K_{KEM} depends only on the security parameter. For readability, we will write messages in $\{0,1\}^{\ell^K_{\mathrm{MAC}}+\ell^K_{\mathrm{IC}}}$ as pairs $(k_{\mathrm{MAC}}, k_{\mathrm{IC}})$.

Fix an injective function $cbcpad : \{0,1\}^* \to (\{0,1\}^{\ell^M_{\mathrm{IC}}})^*$. Fix an efficiently sampleable distribution $\mathcal{D}_{\mathrm{IV}}$ on $\{0,1\}^{\ell^M_{\mathrm{IC}}}$. The function $cbcpad$ is the padding function that maps a message into a sequence of blocks for the ideal cipher. Assume that $cbcpad$ is length-regular, i.e., for $|m| = |m'|$ we have $|cbcpad(m)| = |cbcpad(m')|$, and that $cbcpad$ is efficiently computable and efficiently invertible. We define the encryption in CBC mode as follows: $E^{\mathcal{O}_{\mathrm{IC}}}_{\mathrm{CBC}}(k, m)$ first chooses $c_0 \leftarrow \mathcal{D}_{\mathrm{IV}}$ (the initialization vector)[6] and computes $(m_1, \dots, m_n) \leftarrow cbcpad(m)$. Then it computes $c_i := \mathcal{O}_{\mathrm{IC}}(+, k, c_{i-1} \oplus m_i)$ for $i = 1, \dots, n$ and returns $c := c_0 \| \dots \| c_n$. $D^{\mathcal{O}_{\mathrm{IC}}}_{\mathrm{CBC}}(k, c)$ first splits c as $c_0 \| \dots \| c_n$ with $c_i \in \{0,1\}^{\ell^M_{\mathrm{IC}}}$.

[5]Here we consider $\mathcal{A}$ polynomial-time if it runs a polynomial number of steps in η, and the number of steps performed by RC or FC is also polynomially-bounded. This additional requirement is necessary since for an encryption scheme with multiplicative overhead (say, length-doubling), a sequence of queries $R_i := \mathtt{enc}_{\mathtt{ch}}(N, R_{i-1})$ of polynomial length will lead to the computation of an exponential-length ciphertext.

[6]Notice that a fixed IV can also be modeled by letting $\mathcal{D}_{\mathrm{IV}}$ assign probability 1 to that IV.

Then it computes $m_i := \mathcal{O}_{\mathrm{IC}}(-, k, c_i) \oplus c_{i-1}$, and $m := cbcpad^{-1}(m_1, \dots, m_n)$. If $|c|$ is not a multiple of ℓ_{IC}^M or $|c| = 0$ or $cbcpad^{-1}(m_1, \dots, m_n)$ does not exist, D_{CBC} returns $\perp$, otherwise D_{CBC} returns m.

Construction. We construct a PROG-KDM secure encryption scheme $(K_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}}, E_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}}, D_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}})$ in the ideal cipher model as follows:

- **Key generation:** $K_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}} := K_{\mathrm{KEM}}$.
- **Encryption:** $E_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}}(ek, m)$ performs the following steps: First, it computes $((k_{\mathrm{MAC}}, k_{\mathrm{IC}}), c_{\mathrm{KEM}}) \leftarrow E_{\mathrm{KEM}}(ek)$. Then it computes $c_{\mathrm{CBC}} \leftarrow E_{\mathrm{CBC}}^{\mathcal{O}_{\mathrm{IC}}}(k_{\mathrm{IC}}, m)$ and $t \leftarrow \mathrm{MAC}(k_{\mathrm{MAC}}, c_{\mathrm{CBC}})$ and returns $c := (c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t)$.
- **Decryption:** $D_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}}(dk, c)$ performs the following steps: First, it parses c as $(c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t)$. Then it computes $(k_{\mathrm{MAC}}, k_{\mathrm{IC}}) \leftarrow D_{\mathrm{KEM}}(dk, c_{\mathrm{KEM}})$, checks whether $t = \mathrm{MAC}(k_{\mathrm{MAC}}, c_{\mathrm{CBC}})$, computes and $m \leftarrow D_{\mathrm{CBC}}^{\mathcal{O}_{\mathrm{IC}}}(k_{\mathrm{IC}}, c_{\mathrm{CBC}})$. If parsing, D_{KEM}, D_{CBC}, or the MAC-check fails, return $\perp$. Otherwise, return m.

4 Proof of PROG-KDM security

4.1 Basic definitions

Definition 8 (Ciphertext simulator for the hybrid encryption) *The ciphertext simulator* CS *initially picks a key pair $(ek, dk) \leftarrow K_{\mathrm{KEM}}(1^\eta)$. Then it answers to the following queries:*

- *Upon query* $\mathtt{getek}_{\mathtt{cs}}()$ *or* $\mathtt{getdk}_{\mathtt{cs}}()$, *return ek or dk, respectively.*
- *Upon query* $\mathtt{fakeenc}_{\mathtt{cs}}(R, l)$: *Let n be the number of blocks returned by $cbcpad(0^l)$. Compute $((k_{\mathrm{MAC}}, k_{\mathrm{IC}}), c_{\mathrm{KEM}}) \leftarrow E_{\mathrm{KEM}}(ek)$. Choose $c_{\mathrm{CBC}} := c_0 \| \dots \| c_n$ with $c_0 \leftarrow \mathcal{D}_{\mathrm{IV}}$ and $c_i \xleftarrow{\$} \{0,1\}^{\ell_{\mathrm{IC}}^M}$ for $i \geq 1$. Let $t \leftarrow \mathrm{MAC}(k_{\mathrm{MAC}}, c_{\mathrm{CBC}})$. Return $c := (c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t)$.*
- *Upon query* $\mathtt{dec}_{\mathtt{cs}}(c)$: *Return $D_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}}(dk, c)$.*
- *Upon* $\mathtt{program}_{\mathtt{cs}}(R, m)$*-query: Let k_{IC}, c_{CBC} refer to the values computed in the (unique)* $\mathtt{fakeenc}_{\mathtt{cs}}(R, \cdot)$*-query. Let $c_{\mathrm{CBC}} =: c_0 \| \dots \| c_n$ ($n{+}1$ blocks of length ℓ_{IC}^M). Let $(m_1, \dots, m_n) \leftarrow cbcpad(m)$. Let $x_i := c_{i-1} \oplus m_i$ for all i. If all x_i are pairwise distinct and all c_i are pairwise distinct, program $\mathcal{O}_{\mathrm{IC}}(+, k_{\mathrm{IC}}, x_1) := c_1, \dots, \mathcal{O}_{\mathrm{IC}}(+, k_{\mathrm{IC}}, x_n) := c_n$, and $\mathcal{O}_{\mathrm{IC}}(-, k_{\mathrm{IC}}, c_1) := x_1, \dots, \mathcal{O}_{\mathrm{IC}}(-, k_{\mathrm{IC}}, c_n) := x_n$. (Otherwise do not program anything.)*
- *Upon query* $\mathtt{enc}_{\mathtt{cs}}(R, m)$, *perform the queries* $\mathtt{fakeenc}_{\mathtt{cs}}(R, |m|)$ *and* $\mathtt{program}_{\mathtt{cs}}(R, m)$ *on yourself. Return what the* $\mathtt{fakeenc}_{\mathtt{cs}}$*-query returns.*

Definition 9 (Lazy ideal cipher) *The lazy ideal cipher $\mathcal{O}_{\mathrm{IC,lazy}}$ is a stateful probabilistic oracle. It maintains a function $\mathcal{O}_{\mathrm{IC,lazy}} : \{+,-\} \times \{0,1\}^{\ell_{\mathrm{IC}}^K} \times \{0,1\}^{\ell_{\mathrm{IC}}^M} \to \{0,1\}^{\ell_{\mathrm{IC}}^M}$, initially undefined. (The function is called like the oracle itself in slight abuse of notation.) Upon a query $\mathcal{O}_{\mathrm{IC,lazy}}(d, k, x)$, the oracle responds as follows: If $\mathcal{O}_{\mathrm{IC,lazy}}(d, k, x) = \perp$, it chooses*

$y \xleftarrow{\$} \{0,1\}^{\ell^M_{\mathrm{IC}}}$ and sets $\mathcal{O}_{\mathrm{IC,lazy}}(d,k,x) := y$ and $\mathcal{O}_{\mathrm{IC,lazy}}(\bar d,k,y) := x$ (where $\bar d := -$ if $d = +$ and vice versa). Then it returns $\mathcal{O}_{\mathrm{IC,lazy}}(d,k,x)$.

Lemma 10 (Lazy ideal cipher) *The ideal cipher and the lazy ideal cipher are indistinguishable. More precisely, for any polynomial-time machine $\mathcal{A}$, $|\Pr[\mathcal{A}^{\mathcal{O}_{\mathrm{IC}}}(1^\eta) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_{\mathrm{IC,lazy}}}(1^\eta) = 1]|$ is negligible in the security parameter η.*

Proof. We first define another oracle $\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}$. $\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}$ is defined like $\mathcal{O}_{\mathrm{IC,lazy}}$, except that, instead of choosing $y \xleftarrow{\$} \{0,1\}^{\ell^M_{\mathrm{IC}}}$, it chooses $y \xleftarrow{\$} \{0,1\}^{\ell^M_{\mathrm{IC}}} \setminus \mathrm{range}\ \mathcal{O}_{\mathrm{IC,lazy}}(d,k,\cdot)$.

Since $\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}$ chooses y according to the distribution of the original ideal cipher $\mathcal{O}_{\mathrm{IC}}$ conditioned on the fact that $\mathcal{O}_{\mathrm{IC}}$ matches the function $\mathcal{O}_{\mathrm{IC,lazy}}$ wherever the latter is defined so far, we have that $\mathcal{O}_{\mathrm{IC}}$ and $\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}$ are perfectly indistinguishable, hence $\Pr[\mathcal{A}^{\mathcal{O}_{\mathrm{IC}}}(1^\eta) = 1] = \Pr[\mathcal{A}^{\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}}(1^\eta) = 1]$.

Since $\mathcal{A}$ is polynomial-time, at any point in an execution of $\mathcal{A}^{\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}}$ or $\mathcal{A}^{\mathcal{O}_{\mathrm{IC,lazy}}}$, range $\mathcal{O}_{\mathrm{IC,lazy}}(d,k,\cdot)$ has polynomial size for all d,k. Since ℓ^M_{IC} is superlogarithmic, $\{0,1\}^{\ell^M_{\mathrm{IC}}}$ has superpolynomial size. Thus the uniform distribution on $\{0,1\}^{\ell^M_{\mathrm{IC}}} \setminus \mathrm{range}\ \mathcal{O}_{\mathrm{IC,lazy}}(d,k,\cdot)$ and that on $\{0,1\}^{\ell^M_{\mathrm{IC}}}$ are statistically indistinguishable. Since $\mathcal{A}$ only performs polynomially many queries, it follows that $|\Pr[\mathcal{A}^{\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}}(1^\eta) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_{\mathrm{IC,lazy}}}(1^\eta) = 1]|$ is negligible. With $\Pr[\mathcal{A}^{\mathcal{O}_{\mathrm{IC}}}(1^\eta) = 1] = \Pr[\mathcal{A}^{\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}}(1^\eta) = 1]$ the lemma follows. □

4.2 Preparing the ground

In the following, let $\mathcal{A}$ be an adversary as in Definition 7. Without loss of generality, we assume that all values R and N that $\mathcal{A}$ uses in his queries to RC or FC are chosen from fixed polynomial-size sets $\mathcal{R}$ and $\mathcal{N}$. (As opposed to, e.g., an adversary that picks uniformly chosen random η-bit strings for R and N.) This can be achieved, e.g., if $\mathcal{A}$ uses consecutive integers for R and N; since R and N are treated as opaque identifiers by RC and FC this does not change the outcome of the interaction between $\mathcal{A}$ and RC or FC.

Game 1. $\mathcal{A}$ interacts with RC. ◇

Game 2. $\mathcal{A}$ interacts with FC, with the following modifications to `FCRetrieve`:

- If $reg_R =$ "$\mathtt{enc_{ch}}(N, R_1)$": Compute $m :=$ `FCRetrieve`(R_1). Let $c \leftarrow E^{\mathcal{O}_{\mathrm{IC}}}_{\mathrm{hyb}}(ek_N, m)$ where ek_N denotes the encryption key ek maintained by CS_N. Set $reg_R := c$. Append $(R \mapsto R_1)$ to the list $plain_N$. Append c to $cipher_N$. Return c.
- If $reg_R =$ "$\mathtt{getdk_{ch}}(N)$": Query CS_N with $\mathtt{getdk_{cs}}()$. Store the answer in reg_R. Return reg_R. (That is, in comparison to Definition 4, we do not invoke `FCRetrieve` and do not perform $\mathtt{program_{cs}}(R, m)$-queries.)

◇

We claim that

$$\Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 1] \approx \Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 2] \quad (1)$$

where $\approx$ means that the difference is negligible.

To show this, observe that in Game 2, $\texttt{FCRetrieve}(R)$ computes the value that reg_R would have in Game 1. (In Game 2, `FCRetrieve` does not program the ideal cipher, and its only side effects are that it caches its return values, and that it modifies the sets plain_N and cipher_N which, however, `FCRetrieve` never reads.)

Thus $\texttt{reveal}_{\texttt{ch}}$-queries are answered in Game 1 and Game 2 in the same way. Also $\texttt{oracle}_{\texttt{ch}}$-queries are answered in the same way because the oracle is not programmed. $\texttt{dec}_{\texttt{cs}}(N, c)$-queries might be answered differently: Both Game 2 and Game 1 return `forbidden` to such a query if $c \in \mathit{cipher}_N$. However, in Game 1, cipher_N may contain more values because in Game 2 some encryptions are performed later (because `FCRetrieve` computes the values reg_R lazily). If, however, this occurs, this means that in Game 2 a ciphertext c is used that will only be computed by a later `FCRetrieve`-call. Since ciphertexts produced by E_{KEM} have superlogarithmic min-entropy, and since such ciphertexts are part of the ciphertexts produced by E_{hyb}, this happens with negligible probability. Thus (1) follows.

Game 3. As Game 2, but we replace $\mathcal{O}_{\mathrm{IC}}$ by the lazy ideal cipher $\mathcal{O}_{\mathrm{IC,lazy}}$. $\diamond$

By Lemma 10, we have

$$\Pr[\mathcal{A} \text{ outputs } 1 : \mathit{Game}\ 2] \approx \Pr[\mathcal{A} \text{ outputs } 1 : \mathit{Game}\ 3] \tag{2}$$

Game 4. $\mathcal{A}$ interacts with FC, but `FCRetrieve` is modified as follows:

- If reg_R = "$\texttt{enc}_{\texttt{ch}}(N, R_1)$": Compute $m := \texttt{FCRetrieve}(R_1)$. **Query $\mathbf{CS}_N$ with $\texttt{fakeenc}_{\texttt{cs}}(R, |m|)$. Denote the answer with c.** Set $\mathit{reg}_R := c$. Append $(R \mapsto R_1)$ to the list plain_N. Append c to cipher_N. **Send a $\texttt{program}_{\texttt{cs}}(R, m)$-query to $\mathbf{CS}_N$.** Return c.
- If reg_R = "$\texttt{getdk}_{\texttt{ch}}(N)$": Query CS_N with $\texttt{getdk}_{\texttt{cs}}()$. Store the answer in reg_R. Return reg_R.

And we use $\mathcal{O}_{\mathrm{IC,lazy}}$ instead of $\mathcal{O}_{\mathrm{IC}}$. $\diamond$

We claim that

$$\Pr[\mathcal{A} \text{ outputs } 1 : \mathit{Game}\ 3] \approx \Pr[\mathcal{A} \text{ outputs } 1 : \mathit{Game}\ 4] \tag{3}$$

To show this, we have to show that (a) computing $c \leftarrow E_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC,lazy}}}(ek_N, m)$ has the same effect as (b) sending $\texttt{fakeenc}_{\texttt{cs}}(R, |m|)$ and $\texttt{program}_{\texttt{cs}}(R, m)$ to CS_N and letting c denote the answer of the $\texttt{fakeenc}_{\texttt{cs}}$-query. By definition of E_{hyb}, E_{CBC}, and $\mathcal{O}_{\mathrm{IC,lazy}}$, (a) is the same as performing the following steps:

- $((k_{\mathrm{MAC}}, k_{\mathrm{IC}}), c_{\mathrm{KEM}}) \leftarrow E_{\mathrm{KEM}}(ek_N)$. $c_0 \leftarrow \mathcal{D}_{\mathrm{IV}}$. $(m_0, \dots, m_n) := \mathit{cbcpad}(m)$.
- For $i = 1, \dots, n$, assign $\mathcal{O}_{\mathrm{IC,lazy}}(+, k_{\mathrm{IC}}, c_{i-1} \oplus m_i) := c_i \xleftarrow{\$} \{0,1\}^{\ell^M_{\mathrm{IC}}}$ and $\mathcal{O}_{\mathrm{IC,lazy}}(-, k_{\mathrm{IC}}, c_i) := c_{i-1} \oplus m_i$ [unless $\mathcal{O}_{\mathrm{IC,lazy}}$ is already defined at one of these positions].
- Let $c_{\mathrm{CBC}} := c_0 \| \dots \| c_n$ and $t \leftarrow \mathrm{MAC}(k_{\mathrm{MAC}}, c_{\mathrm{CBC}})$ and $c := (c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t)$.

By definition of CS_N, (b) is the same as performing the following steps:

- Let n be the number of blocks returned by $\mathit{cbcpad}(0^{|m|})$ (which has the same length as $\mathit{cbcpad}(m)$). $((k_{\text{MAC}}, k_{\text{IC}}), c_{\text{KEM}}) \leftarrow E_{\text{KEM}}(ek_N)$. $c_0 \leftarrow \mathcal{D}_{\text{IV}}$.
- For $i = 1, \dots, n$, pick $c_i \xleftarrow{\$} \{0,1\}^{\ell_{\text{IC}}^M}$.
- Let $c_{\text{CBC}} := c_0 \| \dots \| c_n$ and $t \leftarrow \text{MAC}(k_{\text{MAC}}, c_{\text{CBC}})$ and $c := (c_{\text{KEM}}, c_{\text{CBC}}, t)$.
- Let $(m_1, \dots, m_n) \leftarrow \mathit{cbcpad}(m)$.
- For $i = 1, \dots, n$, assign $\mathcal{O}_{\text{IC,lazy}}(+, k_{\text{IC}}, c_{i-1} \oplus m_i) := c_i$ and $\mathcal{O}_{\text{IC,lazy}}(-, k_{\text{IC}}, c_i) := c_{i-1} \oplus m_i$ [unless $c_i = c_j$ or $c_{i-1} \oplus m_i = c_{j-1} \oplus m_j$ for some $i \neq j$].

It is easy to see that these two sequences of steps have the same effect unless one of the conditions in square parentheses occur. In the first sequence, the condition is that $\mathcal{O}_{\text{IC,lazy}}$ is already defined at one of the positions. This happens only with negligible probability since k_{IC} and c_i ($i \geq 1$) are chosen uniformly at random (and at most polynomially many locations of $\mathcal{O}_{\text{IC,lazy}}$ are assigned prior to executing the sequence of steps). In the second sequence, the condition is that $c_i = c_j$ or $c_{i-1} \oplus m_i = c_{j-1} \oplus m_j$ for some $i \neq j$. This happens only with negligible probability because the c_i ($i \geq 1$) are chosen uniformly and independently. Thus with overwhelming probability, (a) and (b) have the same effect. Hence (3) follows.

Game 5. As Game 4, but we replace the lazy ideal cipher $\mathcal{O}_{\text{IC,lazy}}$ by the ideal cipher $\mathcal{O}_{\text{IC}}$. ◇

By Lemma 10, we have

$$\Pr[\mathcal{A} \text{ outputs } 1 : \mathit{Game}\ 4] \approx \Pr[\mathcal{A} \text{ outputs } 1 : \mathit{Game}\ 5]. \tag{4}$$

Game 6. $\mathcal{A}$ interacts with FC, but `FCRetrieve` is modified as follows:

- If reg_R = "$\texttt{enc}_{\texttt{ch}}(N, R_1)$": Compute m := $\texttt{FCRetrieve}(R_1)$. Query CS_N with $\texttt{fakeenc}_{\texttt{cs}}(R, \textbf{FCLen}(\boldsymbol{R_1}))$. Denote the answer with c. Set $\mathit{reg}_R := c$. Append $(R \mapsto R_1)$ to the list plain_N. Append c to cipher_N. Send a $\texttt{program}_{\texttt{cs}}(R, m)$-query to CS_N. Return c.
- If reg_R = "$\texttt{getdk}_{\texttt{ch}}(N)$": Query CS_N with $\texttt{getdk}_{\texttt{cs}}()$. Store the answer in reg_R. Return reg_R.

◇

After the invocation m := $\texttt{FCRetrieve}(R_1)$ we have that $\mathit{reg}_{R_1} = m$. Thus $\texttt{FCLen}(R_1)$ returns $|m|$. Since `FCLen` does not have side-effects, replacing $|m|$ by $\texttt{FCLen}(R_1)$ does not change anything, we have

$$\Pr[\mathcal{A} \text{ outputs } 1 : \mathit{Game}\ 5] = \Pr[\mathcal{A} \text{ outputs } 1 : \mathit{Game}\ 6]. \tag{5}$$

We now describe a game in which all random choices are fixed in the beginning. This will allow us to show that two games (Game 7 and Game 8) exhibit the same behavior when running with the same randomness. In order to be able to show this, it is necessary to make explicit, which component of the initially chosen randomness *Rand* is used at which step of the respective games. E.g., *Rand* contains random tapes t^R_{fakeenc} that constitute the randomness used during $\texttt{fakeenc}_{\texttt{cs}}(R, \cdot)$-queries to CS_N. This ensures that even if that

$\texttt{fakeenc}_{\texttt{cs}}(R,\cdot)$-query occurs at completely different times in the two games, it will use the same randomness in both games nevertheless. If, instead, we had just fixed a single random tape t, and would use the first unused bits of t whenever we need randomness, then changing the order of queries would change which random bits are used by which query.

Game 7. As Game 6, we initially pick a tuple *Rand* containing the following values:

- A key pair $(ek_N, dk_N) \leftarrow K_{\text{KEM}}(1^\eta)$ for each $N \in \mathcal{N}$.
- Values $((k_{\text{MAC}}^{N,R}, k_{\text{IC}}^{N,R}), c_{\text{KEM}}^{N,R}) \leftarrow E_{\text{KEM}}(ek_N)$ for each $N \in \mathcal{N}$, $R \in \mathcal{R}$.
- A random tape t_{fakeenc}^R for each $R \in \mathcal{R}$.
- A random tape $t_{\mathcal{A}}$.
- A uniformly chosen function $\mathcal{O}_{\text{IC}}^0 : \{+,-\} \times \{0,1\}^{\ell_{\text{IC}}^K} \times \{0,1\}^{\ell_{\text{IC}}^M} \to \{0,1\}^{\ell_{\text{IC}}^M}$ such that $\mathcal{O}_{\text{IC}}^0(+,k,\cdot)$ is a permutation for each $k \in \{0,1\}^{\ell_{\text{IC}}^K}$ and $\mathcal{O}_{\text{IC}}^0(-,k,\cdot)$ its inverse.

Then the adversary uses $t_{\mathcal{A}}$ as its random tape, and the ideal cipher $\mathcal{O}_{\text{IC}}$ is initialized as $\mathcal{O}_{\text{IC}}^0$. (But $\mathcal{O}_{\text{IC}}$ may change during the game due to reprogramming.)

And the ciphertext simulator CS_N is modified as follows (in comparison with Definition 8):

- The initial key pair (ek, dk) is not chosen randomly, but instead set to $(ek, dk) := (ek_N, dk_N)$.
- Upon query $\texttt{fakeenc}_{\texttt{cs}}(R,l)$: Let n be the number of blocks returned by $cbcpad(0^l)$. Set $(k_{\text{MAC}}, k_{\text{IC}}, c_{\text{KEM}}) := (k_{\text{MAC}}^{N,R}, k_{\text{IC}}^{N,R}, c_{\text{KEM}}^{N,R})$. Set $c_{\text{CBC}} := c_0\|\dots\|c_n$ with $c_0 \leftarrow \mathcal{D}_{\text{IV}}$ and $c_i \overset{\$}{\leftarrow} \{0,1\}^{\ell_{\text{IC}}^M}$ for $i \geq 1$ and compute $t \leftarrow \text{MAC}(k_{\text{MAC}}, c_{\text{CBC}})$, but use the randomness from t_{fakeenc}^R for the random choices involved in the computation of c_{CBC} and t. Return $c := (c_{\text{KEM}}, c_{\text{CBC}}, t)$.

Additionally, the changes to `FCRetrieve` from Game 6 are also applied in Game 7. $\diamond$

Notice that all random choices in Game 7 are contained in *Rand*. In particular, for fixed *Rand*, Game 7 is deterministic.

Since the only difference between Game 6 and Game 7 is that the random choices are performed earlier in Game 7, we have that

$$\Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 6] = \Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 7]. \tag{6}$$

(Notice that for every R, `FCRetrieve`(R) is only executed once, afterwards it just reuses the result of the previous execution. Thus none of $k_{\text{MAC}}^{N,R}, k_{\text{IC}}^{N,R}, c_{\text{KEM}}^{N,R}, t_{\text{fakeenc}}^R$ is used twice.)

By (1,2,3,4,5,6), we immediately get the following lemma:

Lemma 11 *There is a negligible function μ such that*

$$|\Pr[\mathcal{A} \textit{ outputs } 1 : \textit{Game } 1] - \Pr[\mathcal{A} \textit{ outputs } 1 : \textit{Game } 7]| \leq \mu.$$

4.3 Reordering

Game 8. $\mathcal{A}$ interacts with FC, but `FCRetrieve` is modified as follows:

- If $reg_R =$ "$\mathtt{enc}_{\mathtt{ch}}(N, R_1)$" and there was no $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_N yet: Query CS_N with $\mathtt{fakeenc}_{\mathtt{cs}}(R, \mathtt{FCLen}(R_1))$. Denote the answer with c. Set $reg_R := c$. Append $(R \mapsto R_1)$ to the list $plain_N$. Append c to $cipher_N$. Return c. **(Do not compute $\boldsymbol{m := \mathtt{FCRetrieve}(R_1)}$ and do not perform a $\boldsymbol{\mathtt{program}_{\mathtt{cs}}(R, m)}$-query.)**
- If $reg_R =$ "$\mathtt{enc}_{\mathtt{ch}}(N, R_1)$" and there was a $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_N: Compute $m := \mathtt{FCRetrieve}(R_1)$. Query CS_N with $\mathtt{fakeenc}_{\mathtt{cs}}(R, \mathtt{FCLen}(R_1))$. Denote the answer with c. Set $reg_R := c$. Append $(R \mapsto R_1)$ to the list $plain_N$. Append c to $cipher_N$. Send a $\mathtt{program}_{\mathtt{cs}}(R, m)$-query to CS_N. Return c.
- If $reg_R =$ "$\mathtt{getdk}_{\mathtt{ch}}(N)$": Query CS_N with $\mathtt{getdk}_{\mathtt{cs}}()$. Store the answer in reg_R. **If this was the first $\boldsymbol{\mathtt{getdk}_{\mathtt{cs}}(N)}$-query for that value of $\boldsymbol{N}$, do the following for each $\boldsymbol{(R' \mapsto R'_1) \in plain_N}$ (in the order they occur in the list): Invoke $\boldsymbol{m := \mathtt{FCRetrieve}(R'_1)}$ and send a $\boldsymbol{\mathtt{program}_{\mathtt{cs}}(R', m)}$-query to $\boldsymbol{\mathrm{CS}_N}$.** Finally, return reg_R.

And the randomness is fixed in the beginning as in Game 7. ◇

We will now show that in Game 7 and Game 8 the adversary will give the same output unless certain events occur (such as accessing $\mathcal{O}_{\mathrm{IC}}$ at a location that will later be reprogrammed). These events we will subsequently show to have negligible probability. Proving the equivalence of Game 7 and Game 8 is, however, far from trivial. Although the only difference between Game 7 and Game 8 is that certain $\mathtt{FCRetrieve}$-calls are executed lazily (that is, when their value is first needed), and that certain $\mathtt{program}_{\mathtt{cs}}$-queries are performed later, we have to deal with the fact that $\mathtt{FCRetrieve}$-calls and $\mathtt{program}_{\mathtt{cs}}$-queries have side-effects. E.g., $\mathtt{program}_{\mathtt{cs}}$-queries will program the oracle, and $\mathtt{FCRetrieve}$-calls may change the variables $plain_N$, $cipher_N$ and may also produce further $\mathtt{program}_{\mathtt{cs}}$-queries. Also, doing an $\mathtt{FCRetrieve}$-request later may move the point in time when a $\mathtt{getdk}_{\mathtt{cs}}()$-query is sent to CS_N, which again moves further $\mathtt{FCRetrieve}$-calls and $\mathtt{program}_{\mathtt{cs}}$-queries. Due to this complex reordering, it does not seem possible to translate the difference between Game 7 and Game 8 into a sequence of simple transformations such as swapping two operations. Instead, we will identify certain invariants (Definition 16 below) that relate an execution of Game 7 with an execution of Game 8 (for fixed and identical randomness) and that are strong enough to be able to inductively show that these invariants hold at any point in the execution of the two games (unless the abovementioned events occur), even though things are done in a different order in both games. The invariants will then directly imply that the view of the adversary and hence his output is the same in both games.

For randomness *Rand*, we denote by *Game* 7_{Rand} and *Game* 8_{Rand} the games *Game* 7 and *Game* 8 in which the fixed randomness *Rand* is used (instead of choosing *Rand* randomly in the beginning of the game as described in Game 7). Notice that for fixed *Rand*, *Game* 7_{Rand} and *Game* 8_{Rand} are deterministic.

Before continuing, we define various events that we will need in the remainder of this section. We will later show that all these events have negligible probability.

Definition 12 (Event: Early ideal cipher queries) *In an interaction between $\mathcal{A}$ and* FC *(or a variant thereof), for any R, let k^R_{IC} denote the key k_{IC} chosen in a* $\mathtt{fakeenc}_{\mathtt{cs}}(R, \cdot)$-

query to some ciphertext simulator instance. $k_{\mathrm{IC}}^{R} = \bot$ *if no such query was performed. (Notice that by construction of* `FCRetrieve`*, there will be at most one such query for each R.)*

EarlyIC *denotes the event that for some R, there is an* $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{R}, \cdot)$*-query to the ideal cipher after the* $\mathtt{fakeenc}_{\mathtt{cs}}(R,\cdot)$*-query but before any* $\mathtt{program}_{\mathtt{cs}}(R,\cdot)$*-query occurred.*

Definition 13 (Event: Very early ideal cipher queries) *In the games Game 7 and Game 8,* VeryEarlyIC *denotes the event that for some* $R \in \mathcal{R}$*,* $N \in \mathcal{R}$*, there is an* $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{R,N}, \cdot)$*-query before any* $\mathtt{fakeenc}_{\mathtt{cs}}(R,\cdot)$*-query to* CS_N *occurred.*

We stress that in Definition 12 and Definition 13, we consider $\mathcal{O}_{\mathrm{IC}}$-queries that occur through $\mathtt{oracle}_{\mathtt{ch}}$-query made by $\mathcal{A}$, as well as $\mathcal{O}_{\mathrm{IC}}$-queries that occur indirectly through the execution of the decryption algorithm by an instance of the ciphertext simulator.

Intuitively, VeryEarlyIC corresponds to the fact that $k_{\mathrm{IC}}^{R,N}$ is used before it is ever used. We only define VeryEarlyIC for Game 7 and Game 8 (where $k_{\mathrm{IC}}^{R,N}$ is fixed in the beginning as part of *Rand*) because in these games, we can explicitly refer to the value $k_{\mathrm{IC}}^{R,N}$ before the $\mathtt{fakeenc}_{\mathtt{cs}}(R,\cdot)$-query.

Definition 14 (Event: Guessing a ciphertext) *In Game 7 and Game 8, let* GuessCipher *denote the event that* $\mathcal{A}$ *makes a* $\mathtt{dec}_{\mathtt{ch}}(\cdot, c)$*-query to* FC *with* $c = (c_{\mathrm{KEM}}^{R,N}, \cdot, \cdot)$ *and until that point, no* $\mathtt{fakeenc}_{\mathtt{cs}}(R,\cdot)$*-query was sent to* CS_N*.*

Intuitively, this event implies that $\mathcal{A}$ guesses part of a ciphertext that has not even been produced yet. We only define GuessCipher for Game 7 and Game 8 (where $c_{\mathrm{KEM}}^{R,N}$ is fixed in the beginning as part of *Rand*) because in these games, we can explicitly refer to the value $c_{\mathrm{KEM}}^{R,N}$ before the $\mathtt{fakeenc}_{\mathtt{cs}}(R,\cdot)$-query.

Definition 15 (Good randomness) *We call Rand* good *if the following holds:*

- *The events* EarlyIC*,* VeryEarlyIC*, and* GuessCipher *do not occur in Game* 7_{Rand} *nor in Game* 8_{Rand}*. (Notice that no probabilities are involved here because the games Game* 7_{Rand} *and Game* 8_{Rand} *are deterministic.)*
- *For any* $R, R' \in \mathcal{R}$ *and any* $N, N' \in \mathcal{N}$*, if* $k_{\mathrm{IC}}^{R,N} = k_{\mathrm{IC}}^{R',N'}$ *then* $R = R'$ *and* $N = N'$*. (Note: By definition,* $k_{\mathrm{IC}}^{R,N}$ *is a component of Rand.)*

We will now proceed to show that for good *Rand*, $\mathcal{A}$ outputs 1 in *Game* 7_{Rand} iff $\mathcal{A}$ outputs 1 in *Game* 8_{Rand}. In order to show this, we show a stronger invariant first, given by the following definition:

Definition 16 (Consistent executions) *Consider the execution of Game* 7_{Rand} *(called the* left execution *in the following). Consider the execution of Game* 8_{Rand} *(called the* right execution *in the following). We say that the Rand-*executions are consistent *if the following conditions are satisfied:*

(i) For any R, if there is a (finished) invocation of `FCRetrieve`(R) *in the left and the right execution, then both return the same value.*

(ii) *If in both executions, there was an i-th query to* FC *made by $\mathcal{A}$, then that query is of the same kind and has the same arguments in both executions. (E.g., if the i-th query in the left execution is $R := \mathtt{enc}_{\mathtt{ch}}(N, R_1)$, then the i-th query in the right execution, if there was one, is $R := \mathtt{enc}_{\mathtt{ch}}(N, R_1)$ with the same R, N, R_1.)*

(iii) *If in both executions, there was a (finished) i-th query to* FC *made by $\mathcal{A}$, then both queries returned the same value.*

(iv) *For any R, N, l_1, l_2, if there are (finished) queries $\mathtt{fakeenc}_{\mathtt{cs}}(R, l_1)$ and $\mathtt{fakeenc}_{\mathtt{cs}}(R, l_2)$ to CS_N in the left and right execution, respectively, then they return the same value.*

(v) *For any R, if there are (finished) invocations $\mathtt{FCLen}(R)$ in the left and right execution, then they return the same value.*

(vi) *For any R, m_1, m_2, N_1, N_2, if there is a $\mathtt{program}_{\mathtt{cs}}(R, m_1)$-query to CS_{N_1} in the left execution and a $\mathtt{program}_{\mathtt{cs}}(R, m_2)$-query to CS_{N_2} in the right execution, then $m_1 = m_2$ and $N_1 = N_2$.*

(vii) *If $(R \mapsto R_1) \in plain_{N_1}$ at some point in the left execution, and $(R \mapsto R_2) \in plain_{N_2}$ at some point in the right execution, then $R_1 = R_2$ and $N_1 = N_2$.*

(viii) *For any d, k, x, y_1, y_2, if there are assignments $\mathcal{O}_{\mathrm{IC}}(d, k, x) := y_1$ and $\mathcal{O}_{\mathrm{IC}}(d, k, x) := y_2$ in the left and right execution, respectively, then $y_1 = y_2$.*

(ix) *For any R, m_1, m_2, if there is a $\mathtt{program}_{\mathtt{cs}}(R, m_1)$-query in the left execution and a $\mathtt{program}_{\mathtt{cs}}(R, m_2)$-query in the right execution, then both queries program $\mathcal{O}_{\mathrm{IC}}$ at the same locations.*

Lemma 17 *If Rand is good, then the Rand-executions are consistent.*

Proof. We show this claim by induction over the length of the left and right execution. More precisely, we show that for all n and m, conditions (i)–(ix) hold for prefixes of lengths n and m of the left and right execution, respectively. (The length of a prefix is counted in runtime steps.) To show this, we assume that conditions (i)–(ix) hold for all prefixes of lengths n', m' with $(n' < n \wedge m' \le m)$ or $(n' \le n \wedge m' < m)$. In other words, when showing conditions (i)–(ix), we assume as our induction hypothesis that they already hold if we truncate at least one of the executions by at least one step.

Proof of (i): "For any R, if there is a (finished) invocation of $\mathtt{FCRetrieve}(R)$ *in the left and the right execution, then both return the same value."*

Since all invocations of $\mathtt{FCRetrieve}(R)$ in the same execution with the same R return the same value (since the first invocation sets reg_R to its return value), it is sufficient to consider the first invocation of $\mathtt{FCRetrieve}(R)$ in both executions.

We write reg^l_R and reg^r_R for the value of reg_R at the beginning of the $\mathtt{FCRetrieve}(R)$ invocation in the left and right execution, respectively. By induction hypothesis (condition (ii)), at the beginning of the first $\mathtt{FCRetrieve}(R)$-call, the list of queries

performed by $\mathcal{A}$ so far in one execution is a prefix of the list of queries performed by $\mathcal{A}$ so far in the other execution. Since we consider the first invocation of $\texttt{FCRetrieve}(R)$, reg^l_R and reg^r_R are not bitstrings. In this case, the values of reg^l_R and reg^r_R are determined by the sequence of queries made by $\mathcal{A}$ to FC. Thus $\mathit{reg}^l_R = \mathit{reg}^r_R$.
By induction hypothesis (condition (i)) all $\texttt{FCRetrieve}(R')$-invocations that terminated before $\texttt{FCRetrieve}(R)$ returned the same value in both executions. In particular, the $\texttt{FCRetrieve}(R')$-invocations performed by $\texttt{FCRetrieve}(R)$ returned the same value in both executions. If $\mathit{reg}^l_R = \mathit{reg}^r_R$ is one of "$\texttt{getek}_{\texttt{ch}}(N)$", "$\texttt{eval}_{\texttt{ch}}(C, R_1, \dots, R_n)$", or "$\texttt{getdk}_{\texttt{ch}}(N)$", then this implies that $\texttt{FCRetrieve}(R)$ returns the same value in both executions. (Note that the encryption/decryption keys are fixed by *Rand* and thus the answers in the cases "$\texttt{getek}_{\texttt{ch}}(N)$" and "$\texttt{getdk}_{\texttt{ch}}(N)$" are determined by *Rand*, too.)
Now consider the case $\mathit{reg}^l_R = \mathit{reg}^r_R =$ "$\texttt{enc}_{\texttt{ch}}(N, R_1)$". In this case, the return value of $\texttt{FCRetrieve}(R)$ is that of the query $\texttt{fakeenc}_{\texttt{cs}}(R, \texttt{FCLen}(R_1))$ sent to CS_N. (Note that although $\texttt{FCRetrieve}$ is defined differently in Game 7 and Game 8, this holds with respect to both definitions.) By induction hypothesis (condition (iv)), the $\texttt{fakeenc}_{\texttt{cs}}(R, \texttt{FCLen}(R_1))$-query returns the same value in both executions. Thus $\texttt{FCRetrieve}(R)$ returns the same value in both executions.

Proof of (ii): "If in both executions, there was an i-th query to FC *made by $\mathcal{A}$, then that query is of the same kind and has the same arguments in both executions. (E.g., if the i-th query in the left execution is $R := \texttt{enc}_{\texttt{ch}}(N, R_1)$, then the i-th query in the right execution, if there was one, is $R := \texttt{enc}_{\texttt{ch}}(N, R_1)$ with the same R, N, R_1.)"*
By induction hypothesis (condition (iii)), we have that all earlier queries to FC returned the same values in the left and right execution. Since $\mathcal{A}$'s behavior depends only on its random tape $t_{\mathcal{A}}$ (which is part of *Rand*) and the return values of the queries to FC, this implies that $\mathcal{A}$ makes the same query in both executions.

Proof of (iii): "If in both executions, there was a (finished) i-th query to FC *made by $\mathcal{A}$, then both queries returned the same value."*
By induction hypothesis (condition (ii)), we know that the i-th query in both executions is the same. Depending on that query, we distinguish the following cases:

- If the query is of the form $R := \dots$: Then FC does not return anything, hence the same in both executions.
- If the query is $\texttt{reveal}_{\texttt{ch}}(R_1)$: Then FC returns the return value of $\texttt{FCRetrieve}(R_1)$ in both executions. By induction hypothesis (condition (i)), that value is the same in both executions.
- If the query is $\texttt{oracle}_{\texttt{ch}}(d, k, x)$: We say "$\mathcal{O}_{\mathrm{IC}}$ was programmed left" if there was an assignment $\mathcal{O}_{\mathrm{IC}}(d, k, x) := y$ for some y in the left execution before the $\texttt{oracle}_{\texttt{ch}}(d, k, x)$-query. Analogously "$\mathcal{O}_{\mathrm{IC}}$ was programmed right". We distinguish the following subcases:
  * $\mathcal{O}_{\mathrm{IC}}$ was not programmed left nor programmed right: In this case, in both executions $\mathcal{O}_{\mathrm{IC}}(d, k, x) = \mathcal{O}^0_{\mathrm{IC}}(d, k, x)$ is returned. Since $\mathcal{O}^0_{\mathrm{IC}}$ is part of *Rand*, the same answer is given in both cases.

  * $\mathcal{O}_{\mathrm{IC}}$ was programmed right but not programmed left: In the right execution, programming occurs only within a $\mathtt{program}_{\mathtt{cs}}(R, m)$-query to some CS_N. And that query will (due to the changes in Game 7) program only oracle locations of the form $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{N,R}, \cdot)$. Thus, since $\mathcal{O}_{\mathrm{IC}}$ was programmed right, we have that $k = k_{\mathrm{IC}}^{N,R}$ for some $N \in \mathcal{N}$ and $R \in \mathcal{R}$.
    Since $\mathcal{O}_{\mathrm{IC}}$ was not programmed left, we have that at some point in the left execution, there was an $\mathcal{O}_{\mathrm{IC}}(d, k_{\mathrm{IC}}^{N,R}, x)$-query and until that point $\mathcal{O}_{\mathrm{IC}}(d, k_{\mathrm{IC}}^{N,R}, x)$ had not been programmed. We claim that until that point, there was no $\mathtt{program}_{\mathtt{cs}}(R, \cdot)$-query in the left execution. If there had been a $\mathtt{program}_{\mathtt{cs}}(R, \cdot)$-query, by induction hypothesis (condition (ix)) and due to the fact that $\mathcal{O}_{\mathrm{IC}}(d, k_{\mathrm{IC}}^{R,N}, x)$ has been programmed by the $\mathtt{program}_{\mathtt{cs}}(R, \cdot)$-query in the right execution, it follows that that $\mathtt{program}_{\mathtt{cs}}(R, \cdot)$-query would also have programmed $\mathcal{O}_{\mathrm{IC}}(d, k_{\mathrm{IC}}^{R,N}, x)$ in the left execution. Thus the query to $\mathcal{O}_{\mathrm{IC}}(d, k_{\mathrm{IC}}^{R,N}, x)$ occurred before a $\mathtt{program}_{\mathtt{cs}}(R, \cdot)$-query happened in the left execution. Thus the event $\mathsf{EarlyIC}$ or $\mathsf{VeryEarlyIC}$ occurs in the left execution. This is a contradiction to the assumption that *Rand* is good. Thus the case that $\mathcal{O}_{\mathrm{IC}}$ was programmed right but not programmed left does not occur.
  * $\mathcal{O}_{\mathrm{IC}}$ was programmed left but not programmed right: Analogously to the case that $\mathcal{O}_{\mathrm{IC}}$ was programmed right but not programmed left, we show that this case does not occur (just exchange "left" and "right" in the proof).
  * $\mathcal{O}_{\mathrm{IC}}$ was programmed both right and left: By induction hypothesis (condition (viii)), $\mathcal{O}_{\mathrm{IC}}(d, k, x)$ was assigned the same value in both executions. Thus the same answer is given in both cases.

– If the query is $\mathtt{dec}_{\mathtt{ch}}(N, c)$: The return value of a $\mathtt{dec}_{\mathtt{ch}}(N, c)$-query to FC depends on the three things (besides its arguments N, c): On whether $c \in \mathit{cipher}_N$ (in which case $\mathtt{dec}_{\mathtt{ch}}(N, c)$ returns `forbidden`). On the decryption key dk used by CS_N. And on the results to the oracle queries performed by the decryption algorithm $D_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}}(dk, c)$.
  Due to the modifications from Game 7, the decryption key used by CS_N is dk_N (part of *Rand*) in both executions.
  Assume now that $c \in \mathit{cipher}_N$ in the left execution and $c \notin \mathit{cipher}_N$ in the right execution. A value c is appended to cipher_N by `FCRetrieve` only after some $\mathtt{fakeenc}_{\mathtt{cs}}$-query to CS_N returned c (this holds both in Game 7 and Game 8). Thus in the left execution, there was a $\mathtt{fakeenc}_{\mathtt{cs}}(R, l)$-query to CS_N for some R, N, l that returned c. By definition of $\mathtt{fakeenc}_{\mathtt{cs}}$ (see Game 7), this implies that $c = (c_{\mathrm{KEM}}^{N,R}, \cdot, \cdot)$. If in the right execution, there had been a $\mathtt{fakeenc}_{\mathtt{cs}}(R, l')$-query to CS_N for some l' before the $\mathtt{dec}_{\mathtt{ch}}(N, c)$-query, then, by induction hypothesis (condition (iv)), that $\mathtt{fakeenc}_{\mathtt{cs}}$-query would also have returned c, and we would have $c \in \mathit{cipher}_N$. Thus there was no such query to CS_N. Thus, in the right execution, there was a $\mathtt{dec}_{\mathtt{cs}}(\cdot, c)$-query to FC with $c = (c_{\mathrm{KEM}}^{N,R}, \cdot, \cdot)$ without a prior $\mathtt{fakeenc}_{\mathtt{cs}}(R, \cdot)$-query to CS_N. Thus the event $\mathsf{GuessCipher}$ occurred in the

right execution, in contradiction to the fact that *Rand* is good. Thus the case that $c \in \mathit{cipher}_N$ in the left execution and $c \notin \mathit{cipher}_N$ in the right execution does not occur.

The case that $c \notin \mathit{cipher}_N$ in the left execution and $c \in \mathit{cipher}_N$ in the right execution is excluded analogously.

It remains to show that the results to the oracle queries performed by the decryption algorithm $D_{\mathrm{hyb}}^{\mathcal{O}_{\mathrm{IC}}}(dk_N, c)$ are the same in both executions. This is shown analogously to the fact that $\mathtt{oracle}_{\mathtt{ch}}(d,k,x)$-queries to FC return the same value in both executions (see the proof of condition (iii), subcase $\mathtt{oracle}_{\mathtt{ch}}(d,k,x)$, page 21).

Proof of (iv): "For any R, N, l_1, l_2, if there are (finished) queries $\mathtt{fakeenc}_{\mathtt{cs}}(R, l_1)$ and $\mathtt{fakeenc}_{\mathtt{cs}}(R, l_2)$ to CS_N in the left and right execution, respectively, then they return the same value."

First, note that in both executions, all $\mathtt{fakeenc}_{\mathtt{cs}}$-queries are of the form $\mathtt{fakeenc}_{\mathtt{cs}}(R, \mathtt{FCLen}(R))$. By induction hypothesis (condition (v)), this implies that for queries $\mathtt{fakeenc}_{\mathtt{cs}}(R, l_1)$ and $\mathtt{fakeenc}_{\mathtt{cs}}(R, l_2)$ in the left and right execution (to the same CS_N), respectively, we have $l_1 = l_2$. By construction of CS_N (Definition 8), the return value $c = (c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t)$ of a $\mathtt{fakeenc}_{\mathtt{cs}}(R, l)$-query depends only on l and the randomness used by CS_N in that query. In Game 7 and Game 8, that randomness is $k_{\mathrm{MAC}}^{N,R}, k_{\mathrm{IC}}^{N,R}, c_{\mathrm{KEM}}^{N,R}, t_{\mathrm{fakeenc}}^{R}$. All these values are determined by *Rand*. Hence the return value of $\mathtt{fakeenc}_{\mathtt{cs}}(R, l_1)$ and $\mathtt{fakeenc}_{\mathtt{cs}}(R, l_2)$ is the same.

Proof of (v): "For any R, if there are (finished) invocations $\mathtt{FCLen}(R)$ in the left and right execution, then they return the same value."

We write reg_R^l and reg_R^r for the value of reg_R when $\mathtt{FCLen}(R)$ is invoked in the left and right execution, respectively. Since $\mathtt{FCLen}(R)$ is never invoked for undefined reg_R, we have that reg_R^l and reg_R^r are defined. For any R', by $\widetilde{\mathit{reg}}_{R'}$ we denote the initial value of $\mathit{reg}_{R'}$, i.e., the value of $\mathit{reg}_{R'}$ directly after the $R' := \dots$-query to FC that assigned a value to it. Notice that the value $\widetilde{\mathit{reg}}_{R'}$ is completely determined by the sequence of queries made by $\mathcal{A}$ to FC. By induction hypothesis (condition (ii)) these queries are the same in both executions. Thus $\widetilde{\mathit{reg}}_{R'}^l = \widetilde{\mathit{reg}}_{R'}^r$ whenever both are defined (where $\widetilde{\mathit{reg}}_{R'}^l = \widetilde{\mathit{reg}}_{R'}^r$ denote the value of $\widetilde{\mathit{reg}}_{R'}$ in the left and right execution, respectively). Let $\widetilde{\mathtt{FCLen}}$ be defined like $\mathtt{FCLen}$, but using the values $\widetilde{\mathit{reg}}_{R'}$ instead of the values $\mathit{reg}_{R'}$. Then $\widetilde{\mathtt{FCLen}}(R)$ has the same value in both executions (since reg_R and thus also $\widetilde{\mathit{reg}}_R$ is defined in both executions). Furthermore, notice that by construction, within the same execution, for all R', $\widetilde{\mathtt{FCLen}}(R') = \mathtt{FCLen}(R')$. (This is due to the fact that $\mathtt{FCRetrieve}$ assigns to $\mathit{reg}_{R'}$ only bitstrings m with $|m| = \mathtt{FCLen}(R')$.) Hence also $\mathtt{FCLen}(R)$ returns the same value in both executions.

Proof of (vi): "For any R, m_1, m_2, N_1, N_2, if there is a $\mathtt{program}_{\mathtt{cs}}(R, m_1)$-query to CS_{N_1} in the left execution and a $\mathtt{program}_{\mathtt{cs}}(R, m_2)$-query to CS_{N_2} in the right execution, then $m_1 = m_2$ and $N_1 = N_2$."

Both in the left and in the right execution, a $\mathtt{program}_{\mathtt{cs}}(R, m)$-query is only performed if for some R', N, the bitstring m was the result of $\mathtt{FCRetrieve}(R')$ and $(R \mapsto R') \in$

$plain_N$. Thus m_1 was returned by $\texttt{FCRetrieve}(R_1)$ in the left execution and m_2 by $\texttt{FCRetrieve}(R_2)$ in the right execution, and we have $(R \mapsto R_1) \in plain_{N_1}$ in the left execution and $(R \mapsto R_2) \in plain_{N_2}$ in the left execution. By induction hypothesis (condition (vii)), we have that $R_1 = R_2$ and $N_1 = N_2$. Hence, by induction hypothesis (condition (i)), $m_1 = m_2$.

Proof of (vii): "If $(R \mapsto R_1) \in plain_{N_1}$ at some point in the left execution, and $(R \mapsto R_2) \in plain_{N_2}$ at some point in the right execution, then $R_1 = R_2$ and $N_1 = N_2$."
Both in Game 7 and Game 8, $plain_N$ is only modified by $\texttt{FCRetrieve}(R)$ with $reg_R =$ "$\texttt{enc}_{\texttt{ch}}(N, R')$", and in this case $(R \mapsto R')$ is appended to $plain_N$. It follows that there was an $\texttt{FCRetrieve}(R)$ call in the left and in the right execution, and in the left execution, we had $reg_R =$ "$\texttt{enc}_{\texttt{ch}}(N_1, R_1)$" and in the right execution, we had $reg_R =$ "$\texttt{enc}_{\texttt{ch}}(N_2, R_2)$". This implies that there was $R := \texttt{enc}_{\texttt{ch}}(N_1, R_1)$-query and an $R := \texttt{enc}_{\texttt{ch}}(N_2, R_2)$-query to FC by $\mathcal{A}$ in the left and right execution, respectively. By induction hypothesis (condition (ii)) and using the fact that $\mathcal{A}$ does not perform two $R := \dots$-queries with the same R, it follows that $N_1 = N_2$ and $R_1 = R_2$.

Proof of (viii): "For any d, k, x, y_1, y_2, if there are assignments $\mathcal{O}_{\text{IC}}(d, k, x) := y_1$ and $\mathcal{O}_{\text{IC}}(d, k, x) := y_2$ in the left and right execution, respectively, then $y_1 = y_2$."
Since $\mathcal{O}_{\text{IC}}(d, k, x)$ is only programmed within $\texttt{program}_{\texttt{cs}}(R, m)$-queries to CS_N with $k = k_{\text{IC}}^{N,R}$, it follows that in the left and the right execution, we have queries $\texttt{program}_{\texttt{cs}}(R_l, m_l)$ to CS_{N_l} and $\texttt{program}_{\texttt{cs}}(R_r, m_r)$ to CS_{N_r}, respectively, with $k_{\text{IC}}^{N_l,R_l} = k_{\text{IC}}^{N_r,R_r}$. Since *Rand* is good, this implies that $R_l = R_r$. Thus, by induction hypothesis (condition (vi)), we have that $m_l = m_r$. The value c_{CBC} used by the $\texttt{program}_{\texttt{cs}}(R_l, m_l)$-query and the $\texttt{program}_{\texttt{cs}}(R_r, m_r)$-query, respectively, is the same by induction hypothesis (condition (iv), using the fact that c_{CBC} is part of the return value of $\texttt{fakeenc}_{\texttt{cs}}$). Since the values assigned to $\mathcal{O}_{\text{IC}}$ in $\texttt{program}_{\texttt{cs}}(R, m)$-queries only depend on R, m and c_{CBC}, it follows that $\mathcal{O}_{\text{IC}}(d, k, x)$ is assigned the same value in the left and the right execution.

Proof of (ix): "For any R, m_1, m_2, if there is a $\texttt{program}_{\texttt{cs}}(R, m_1)$-query in the left execution and a $\texttt{program}_{\texttt{cs}}(R, m_2)$-query in the right execution, then both queries program $\mathcal{O}_{\text{IC}}$ at the same locations."
By induction hypothesis (condition (vi)), we have $m_1 = m_2$. The value c_{CBC} used by the $\texttt{program}_{\texttt{cs}}(R, m_1)$-query and the $\texttt{program}_{\texttt{cs}}(R, m_2)$-query, respectively, is the same by induction hypothesis (condition (iv), using the fact that c_{CBC} is part of the return value of a $\texttt{fakeenc}_{\texttt{cs}}(R, \cdot)$-query). By induction hypothesis (condition (vi)), both $\texttt{program}_{\texttt{cs}}$-queries are sent to a ciphertext simulator CS_N with the same index N. Thus the value k_{IC} used by the $\texttt{program}_{\texttt{cs}}$-query is $k_{\text{IC}} = k_{\text{IC}}^{R,N}$ in both executions. Since the locations at which $\mathcal{O}_{\text{IC}}$ is assigned in $\texttt{program}_{\texttt{cs}}(R, m)$-queries only depend on k_{IC}, m and c_{CBC} (cf. Definition 8), it follows that $\mathcal{O}_{\text{IC}}$ is assigned at the same locations in the left and the right execution.

This concludes the proof of Lemma 17. □

Lemma 18 *There is a negligible function μ such that*

$$\Big|\Pr[\mathcal{A} \textit{ outputs } 1 : \textit{Game } 1] - \Pr[\mathcal{A} \textit{ outputs } 1 : \textit{Game } 8]\Big| \le \Pr[\mathsf{EarlyIC} : \textit{Game } 8] + \mu.$$

Proof. For good *Rand*, Lemma 17 implies that condition (iii) from Definition 16 holds for executions of $\textit{Game } 7_{\textit{Rand}}$ and $\textit{Game } 8_{\textit{Rand}}$. Thus $\mathcal{A}$ gets the same answers to all its queries in both executions. Since the output of $\mathcal{A}$ only depends on its randomness (which is $t_{\mathcal{A}}$, contained in *Rand*) and the answers to its queries, it follows that for good *Rand*, $\mathcal{A}$ outputs 1 in $\textit{Game } 7_{\textit{Rand}}$ iff $\mathcal{A}$ outputs 1 in $\textit{Game } 8_{\textit{Rand}}$. By averaging over all randomnesses *Rand*, we get

$$|\Pr[\mathcal{A} = 1 : \textit{Game } 7] - \Pr[\mathcal{A} = 1 : \textit{Game } 8]| \le \Pr[\textit{Rand} \text{ is not good}]. \tag{7}$$

We thus have to bound $\Pr[\textit{Rand} \text{ is not good}]$. By definition of good randomness, we have

$$\begin{aligned}\Pr[\textit{Rand} \text{ is not good}] \le\ & \Pr[\mathsf{EarlyIC} : \textit{Game } 7] + \Pr[\mathsf{EarlyIC} : \textit{Game } 8] \\ & + \Pr[\mathsf{VeryEarlyIC} : \textit{Game } 7] + \Pr[\mathsf{VeryEarlyIC} : \textit{Game } 8] \\ & + \Pr[\mathsf{GuessCipher} : \textit{Game } 7] + \Pr[\mathsf{GuessCipher} : \textit{Game } 8] \\ & + \Pr[\mathsf{KCollision}]. \end{aligned} \tag{8}$$

where $\mathsf{KCollision}$ denotes the event that a randomly chosen *Rand* satisfies $k_{\mathrm{IC}}^{R,N} = k_{\mathrm{IC}}^{R',N'}$ for some $R, R' \in \mathcal{R}$, $N, N' \in \mathcal{N}$ with $(R,N) \ne (R',N')$.

Since the values $k_{\mathrm{IC}}^{R,N}$ are chosen uniformly from $\{0,1\}^{\ell_{\mathrm{IC}}^K}$, and ℓ_{IC}^K is superpolynomial, and $\mathcal{R}$ and $\mathcal{N}$ are of polynomial size, we have that $\Pr[\mathsf{KCollision}]$ is negligible.

In Game 7 and Game 8, the values $k_{\mathrm{MAC}}^{N,R}$, $k_{\mathrm{IC}}^{R,N}$, and $c_{\mathrm{KEM}}^{R,N}$ are only accessed within a $\texttt{fakeenc}_{\texttt{cs}}(R,\cdot)$-query to CS_N. Thus, the event $\mathsf{VeryEarlyIC}$ would imply that $k_{\mathrm{IC}}^{N,R}$ occurs in an oracle query before $k_{\mathrm{IC}}^{N,R}$ (or the derived values $k_{\mathrm{MAC}}^{N,R}$ and $c_{\mathrm{KEM}}^{R,N}$) have been accessed for the first time. Since $k_{\mathrm{IC}}^{N,R}$ is chosen uniformly and has superpolynomial length ℓ_{IC}^K, this happens with negligible probability. Thus $\Pr[\mathsf{VeryEarlyIC} : \textit{Game } 7]$ and $\Pr[\mathsf{VeryEarlyIC} : \textit{Game } 8]$ are negligible.

Similarly, we analyze the probability of $\mathsf{GuessCipher}$ (in both *Game* 7 and *Game* 8). Since $k_{\mathrm{IC}}^{N,R}$ has superlogarithmic min-entropy, and since from $c_{\mathrm{KEM}}^{N,R}$ one can, given ek_N, compute $k_{\mathrm{IC}}^{N,R}$, we get that $c_{\mathrm{KEM}}^{N,R}$ has superlogarithmic min-entropy. Thus the probability that $c_{\mathrm{KEM}}^{N,R}$ occurs in a $\texttt{dec}_{\texttt{ch}}$-query before $k_{\mathrm{MAC}}^{N,R}$, $k_{\mathrm{IC}}^{R,N}$, or $c_{\mathrm{KEM}}^{R,N}$ have been accessed is negligible. Since these values are only accessed in a $\texttt{fakeenc}_{\texttt{cs}}(R,\cdot)$-query to CS_N, it follows that $\Pr[\mathsf{GuessCipher} : \textit{Game } 7]$ and $\Pr[\mathsf{GuessCipher} : \textit{Game } 8]$ are negligible.

In Game 7, for each R, the $\texttt{program}_{\texttt{cs}}(R,\cdot)$-query immediately follows the corresponding $\texttt{fakeenc}_{\texttt{cs}}(R,\cdot)$-query. Thus, no $\mathcal{O}_{\mathrm{IC}}$-queries occur in between. Thus the event $\mathsf{EarlyIC}$ does not occur. Hence $\Pr[\mathsf{EarlyIC} : \textit{Game } 7] = 0$. Notice that this reasoning does not apply to Game 8.

Thus all summands on the right hand side of (8) except for $\Pr[\mathsf{EarlyIC} : \textit{Game } 8]$ are negligible. Hence $\Pr[\textit{Rand} \text{ is not good}] \le \Pr[\mathsf{EarlyIC} : \textit{Game } 8] + \mu_1$ for some negligible μ_1. With (7), we get

$$|\Pr[\mathcal{A} = 1 : \textit{Game } 7] - \Pr[\mathcal{A} = 1 : \textit{Game } 8]| \le \Pr[\mathsf{EarlyIC} : \textit{Game } 8] + \mu_1.$$

With Lemma 11, it follows that

$$|\Pr[\mathcal{A} = 1 : \textit{Game } 1] - \Pr[\mathcal{A} = 1 : \textit{Game } 8]| \le \Pr[\mathsf{EarlyIC} : \textit{Game } 8] + \mu_1 + \mu_2$$

for some negligible μ_2. With $\mu := \mu_1 + \mu_2$, Lemma 18 follows. □

4.4 Cleaning up

We undo the changes from Game 7. That is, Game 8 is like Game 7, but instead of using the randomness *Rand*, the fake challenger FC and the adversary $\mathcal{A}$ again use their own randomness. More precisely, we have the following game:

Game 9. $\mathcal{A}$ interacts with FC, but `FCRetrieve` is modified as follows:

- If $\textit{reg}_R =$ "$\mathtt{enc}_{\mathtt{ch}}(N, R_1)$" and there was a $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_N: Compute $m :=$ $\mathtt{FCRetrieve}(R_1)$. Query CS_N with $\mathtt{fakeenc}_{\mathtt{cs}}(R, \mathtt{FCLen}(R_1))$. Denote the answer with c. Set $\textit{reg}_R := c$. Append $(R \mapsto R_1)$ to the list $\textit{plain}_N$. Append c to $\textit{cipher}_N$. Send a $\mathtt{program}_{\mathtt{cs}}(R, m)$-query to CS_N. Return c.

The randomness of $\mathcal{A}$ and FC is not fixed at the beginning. ◇

Notice that we omitted the case "$\textit{reg}_R =$ '$\mathtt{enc}_{\mathtt{ch}}(N, R_1)$' and there was no $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_N yet" and the case "$\textit{reg}_R =$ '$\mathtt{getdk}_{\mathtt{ch}}(N)$'" in the description of Game 9 because already in Game 8, these cases were as in the original description of `FCRetrieve` (Definition 4). Thus the only change between Game 8 and Game 9 is that the randomness is not fixed any more. Analogously to (6), we thus have

$$\Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 8] = \Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 9] \tag{9}$$

$$\text{and} \qquad \Pr[\mathsf{EarlyIC} : \textit{Game } 8] = \Pr[\mathsf{EarlyIC} : \textit{Game } 9] \tag{10}$$

Game 10. $\mathcal{A}$ interacts with FC, but `FCRetrieve` is modified as follows:

- If $\textit{reg}_R =$ "$\mathtt{enc}_{\mathtt{ch}}(N, R_1)$" and there was a $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_N: Compute $m :=$ $\mathtt{FCRetrieve}(R_1)$. **Query CS_N with $\mathtt{enc}_{\mathtt{cs}}(R, m)$.** Denote the answer with c. Set $\textit{reg}_R := c$. Append $(R \mapsto R_1)$ to the list $\textit{plain}_N$. Append c to $\textit{cipher}_N$. **(Do not perform a $\mathtt{program}_{\mathtt{cs}}(R, m)$-query.)** Return c.

◇

After $m := \mathtt{FCRetrieve}(R_1)$, $\textit{reg}_{R_1} = m$ and hence $\mathtt{FCLen}(R_1) = |m|$. Thus $\mathtt{fakeenc}_{\mathtt{cs}}(R, \mathtt{FCLen}(R_1))$ is the same as $\mathtt{fakeenc}_{\mathtt{cs}}(R, |m|)$. By definition of CS_N (Definition 8), an $\mathtt{enc}_{\mathtt{cs}}(R, m)$-query has the same effect as a $\mathtt{fakeenc}_{\mathtt{cs}}(R, |m|)$-query followed by a $\mathtt{program}_{\mathtt{cs}}(R, m)$-query. Thus an $\texttt{enc}_{\texttt{cs}}(R, m)$-query (as in Game 10) has the same effect as a $\texttt{fakeenc}_{\texttt{cs}}(R, \texttt{FCLen}(R_1))$-query followed by a $\texttt{program}_{\texttt{cs}}(R, m)$-query (as in Game 9). Hence

$$\Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 9] = \Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 10] \tag{11}$$

$$\text{and} \quad \Pr[\mathsf{EarlyIC} : \textit{Game } 9] = \Pr[\mathsf{EarlyIC} : \textit{Game } 10]. \tag{12}$$

Game 11. $\mathcal{A}$ interacts with FC (without any modifications). ◇

Since Game 10 and Game 11 are the same game (the modifications listed in Game 10 match what `FCRetrieve` does anyway in Definition 4), we have

$$\Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 10] = \Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 11] \tag{13}$$

$$\text{and} \quad \Pr[\mathsf{EarlyIC} : \textit{Game } 10] = \Pr[\mathsf{EarlyIC} : \textit{Game } 11]. \tag{14}$$

By Lemma 18 and (9,11,13) we have that

$$\Big|\Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 1] - \Pr[\mathcal{A} \text{ outputs } 1 : \textit{Game } 11]\Big| \le \Pr[\mathsf{EarlyIC} : \textit{Game } 8] + \mu$$

for some negligible μ. With (10,12,14) the following lemma immediately follows:

Lemma 19 *There is a negligible function μ such that*

$$\Big|\Pr[\mathcal{A} \textit{ outputs } 1 : \mathcal{A} \textit{ interacts with } \mathrm{RC}] - \Pr[\mathcal{A} \textit{ outputs } 1 : \mathcal{A} \textit{ interacts with } \mathrm{FC}]\Big|$$
$$\le \Pr[\mathsf{EarlyIC} : \mathcal{A} \textit{ interacts with } \mathrm{FC}] + \mu.$$

4.5 Analyzing FC

Lemma 20 *In an interaction of $\mathcal{A}$ and* FC*, we have that* $\Pr[\mathsf{EarlyIC}]$ *is negligible.*

Proof. Again, we analyze a sequence of games:

Game 12. $\mathcal{A}$ interacts with FC. ◇

Our goal is to show that $\Pr[\mathsf{EarlyIC} : \textit{Game } 12]$ is negligible.

Game 13. As Game 12, but we initially pick $R^* \xleftarrow{\$} \mathcal{R}$ and $N^* \xleftarrow{\$} \mathcal{N}$. (But R^* and N^* are never used afterward.) ◇

Using the notation of Definition 12, let $\mathsf{EarlyIC}^*$ denotes the event that there is an $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{R^*}, \cdot)$-query to the ideal cipher after the $\texttt{fakeenc}_{\texttt{cs}}(R^*, \cdot)$-query to CS_{N^*} but before any $\texttt{program}_{\texttt{cs}}(R^*, \cdot)$-query occurred.

Comparing this to the definition of $\mathsf{EarlyIC}$ (Definition 12), we see that $\mathsf{EarlyIC}$ is the event that $\mathsf{EarlyIC}^*$ occurs for some R^*, N^*. Thus

$$\Pr[\mathsf{EarlyIC} : \textit{Game } 12] = \Pr[\mathsf{EarlyIC} : \textit{Game } 13] \le |\mathcal{R}| \cdot |\mathcal{N}| \cdot \Pr[\mathsf{EarlyIC}^* : \textit{Game } 13].$$

Since $|\mathcal{R}|$ and $|\mathcal{N}|$ are of polynomial size, we have

$$\Pr[\mathsf{EarlyIC}^* : \mathit{Game}\ 13] \text{ is negligible} \quad \Longrightarrow \quad \Pr[\mathsf{EarlyIC} : \mathit{Game}\ 12] \text{ is negligible}. \tag{15}$$

Game 14. As Game 13, but the game aborts when a $\mathtt{getdk}_{\mathtt{cs}}()$-query is sent to CS_{N^*} (before executing that query). $\diamond$

We will show that

$$\Pr[\mathsf{EarlyIC}^* : \mathit{Game}\ 13] = \Pr[\mathsf{EarlyIC}^* : \mathit{Game}\ 14]. \tag{16}$$

To show this, it is sufficient to show that in Game 13, if $\mathsf{EarlyIC}^*$ occurs, then it already occurs before the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*} (if there is one). Assume this is not the case. Then we distinguish the following cases:

- *The first $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{R^*}, \cdot)$-query occurs after the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*}, and there was a $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*} before the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*}.*

  By definition of $\mathtt{FCRetrieve}$, if $\mathtt{FCRetrieve}$ sent a $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*} prior to sending $\mathtt{getdk}_{\mathtt{cs}}()$ to CS_{N^*}, then $(R^* \mapsto R_1) \in \mathit{plain}_{N^*}$ for some R_1. Again by definition of $\mathtt{FCRetrieve}$, the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*} is followed by a $\mathtt{program}_{\mathtt{cs}}(R', \cdot)$-query for each $(R' \mapsto \cdot) \in \mathit{plain}_{N^*}$ (with no $\mathcal{O}_{\mathrm{IC}}$-queries in between as these are only performed by FC during $\mathtt{dec}_{\mathtt{ch}}$- and $\mathtt{oracle}_{\mathtt{ch}}$-queries). In particular, there is a $\mathtt{program}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*} after the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*} with no $\mathcal{O}_{\mathrm{IC}}$-queries in between. Thus the first $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{R^*}, \cdot)$ occurs after a $\mathtt{program}_{\mathtt{cs}}(R^*, \cdot)$-query, hence by definition the event $\mathsf{EarlyIC}^*$ does not occur.

- *The first $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{R^*}, \cdot)$-query occurs after the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*}, and there was no $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*} before the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*}.*

  Since $k_{\mathrm{IC}}^{R^*}$ is only defined after a $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query, the first $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{R^*}, \cdot)$-query occurs after a $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query. Since there was no $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*} before the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*}, $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$ occurs after the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*}. By definition of $\mathtt{FCRetrieve}$, this can only happen as part of an $\mathtt{enc}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*}. And in that case, the $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query is immediately followed by a $\mathtt{program}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*} (with no $\mathcal{O}_{\mathrm{IC}}$-queries in between). Thus, since the first $\mathcal{O}_{\mathrm{IC}}(\cdot, k_{\mathrm{IC}}^{R^*}, \cdot)$-query occurs after $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$, it occurs after a $\mathtt{program}_{\mathtt{cs}}(R^*, \cdot)$-query. Hence the event $\mathsf{EarlyIC}^*$ does not occur.

Thus, we know that in Game 13, if $\mathsf{EarlyIC}^*$ occurs, then it occurs before the first $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*} (if there is one). Thus aborting at the first $\mathtt{getdk}_{\mathtt{cs}}()$-query does not change the probability of $\mathsf{EarlyIC}^*$ and (16) follows.

In the following, by a *KEM-oracle*, we mean an oracle that initially picks a key pair $(ek, dk) \leftarrow K_{\mathrm{KEM}}(1^\eta)$. This oracle supports the following queries: Upon $\mathtt{getek}_{\mathtt{kem}}()$, return ek. Upon $\mathtt{enc}_{\mathtt{kem}}()$, compute $(k, c) \leftarrow E_{\mathrm{KEM}}(ek)$ and return (k, c). Only the first such query is answered. Upon $\texttt{dec}_{\texttt{kem}}(c)$, if (k^*, c^*) with $c = c^*$ was returned by an $\texttt{enc}_{\texttt{kem}}()$-query, return k^*. Otherwise compute $k \leftarrow D_{\text{KEM}}(dk, c)$ and return k.

We also define the *fake KEM-oracle* which is defined like the KEM-oracle, except that upon an $\texttt{enc}_{\texttt{kem}}()$-query, it computes $(k', c) \leftarrow E_{\text{KEM}}(ek)$, picks $k \xleftarrow{\$} \{0,1\}^{\ell^K_{\text{MAC}}+\ell^K_{\text{IC}}}$, and returns (k, c).

Notice that it follows directly from the definition of a CCA-secure KEM (see [CS03]) that no polynomial-time adversary can distinguish between the KEM-oracle and the fake KEM-oracle.

Game 15. As Game 14, but the ciphertext simulator CS_{N^*} (but not CS_N with $N \neq N^*$) is changed as follows:

- **At the beginning, it initializes a KEM-oracle and performs a $\texttt{getek}_{\texttt{kem}}()$-query on the KEM-oracle to initialize ek. dk is not initialized.**
- Upon query $\texttt{fakeenc}_{\texttt{cs}}(R, l)$ with $R = R^*$ (the case $R \neq R^*$ is unchanged): Let n be the number of blocks returned by $\mathit{cbcpad}(0^l)$. **Query $\texttt{enc}_{\texttt{kem}}()$ from the KEM-oracle and parse the result as $((k_{\text{MAC}}, k_{\text{IC}}), c_{\text{KEM}})$.** Choose $c_{\text{CBC}} \xleftarrow{\$} (\{0,1\}^{\ell^M_{\text{IC}}})^{n+1}$. Let $t \leftarrow \text{MAC}(k_{\text{MAC}}, c_{\text{CBC}})$. Return $c := (c_{\text{KEM}}, c_{\text{CBC}}, t)$.
- Upon query $\texttt{dec}_{\texttt{cs}}(c)$: Parse c as $(c_{\text{KEM}}, c_{\text{CBC}}, t)$. **Query $\texttt{dec}_{\texttt{kem}}(c_{\text{KEM}})$ from the KEM-oracle and parse the result as $(k_{\text{MAC}}, k_{\text{IC}})$.** Check whether $t = \text{MAC}(k_{\text{MAC}}, c_{\text{CBC}})$. Compute $m \leftarrow D^{\mathcal{O}_{\text{IC}}}_{\text{CBC}}(k_{\text{IC}}, c_{\text{CBC}})$. If parsing, the $\texttt{dec}_{\texttt{kem}}(c_{\text{KEM}})$-query, D_{CBC}, or the MAC-check fails, return $\perp$. Otherwise, return m.

$\diamond$

In the following, by $n^*, k^*_{\text{MAC}}, k^*_{\text{IC}}, c^*_{\text{KEM}}, c^*_{\text{CBC}}, t^*, c^*$ we denote the corresponding values $n, k_{\text{MAC}}, k_{\text{IC}}, c_{\text{KEM}}, c_{\text{CBC}}, t, c$ from the (unique) $\texttt{fakeenc}_{\texttt{cs}}(R^*, \cdot)$-query to CS_{N^*}.

In Game 15, we have outsourced invocations to K_{KEM}, E_{KEM}, and D_{KEM} to the KEM-oracle. Notice that CS_{N^*} does not need to access the decryption key dk any more, since we abort before sending a $\texttt{getdk}_{\texttt{cs}}()$-query to CS_{N^*}. Furthermore, by construction of `FCRetrieve`, there is only one $\texttt{fakeenc}_{\texttt{cs}}(R, \cdot)$-query with $R = R^*$ (since `FCRetrieve` caches its results), thus only one $\texttt{enc}_{\texttt{kem}}$-query will be sent to the KEM-oracle. Thus

$$\Pr[\mathsf{EarlyIC}^* : \mathit{Game}\ 14] = \Pr[\mathsf{EarlyIC}^* : \mathit{Game}\ 15]. \tag{17}$$

Game 16. As Game 15, but we use the fake KEM-oracle instead of the KEM-oracle. $\diamond$

Since *Game* 15 and *Game* 16 run in polynomial-time, and since $(K_{\text{KEM}}, E_{\text{KEM}}, D_{\text{KEM}})$ is a CCA-secure KEM, we get

$$\Pr[\mathsf{EarlyIC}^* : \mathit{Game}\ 15] \approx \Pr[\mathsf{EarlyIC}^* : \mathit{Game}\ 16] \tag{18}$$

where $\approx$ means that the difference is negligible.

Game 17. As Game 16, but we further modify CS_{N^*}: Upon a query $\texttt{dec}_{\texttt{cs}}(c)$, it parses c as $(c_{\text{KEM}}, c_{\text{CBC}}, t)$. If $c_{\text{KEM}} = c^*_{\text{KEM}}$ and $t \neq \text{MAC}(k^*_{\text{MAC}}, c_{\text{CBC}})$, the $\texttt{dec}_{\texttt{cs}}(c)$-query returns $\perp$. Otherwise, the $\texttt{dec}_{\texttt{cs}}(c)$-query proceeds as described in Game 15. $\diamond$

Notice that in Game 16, if $c_{\mathrm{KEM}} = c^*_{\mathrm{KEM}}$, the $\mathtt{dec}_{\mathtt{kem}}(c_{\mathrm{KEM}})$-query to the fake KEM-oracle returns $(k_{\mathrm{MAC}}, k_{\mathrm{IC}}) = (k^*_{\mathrm{MAC}}, k^*_{\mathrm{IC}})$. Thus, if $t \neq \mathrm{MAC}(k^*_{\mathrm{MAC}}, c_{\mathrm{CBC}})$, we also have $t \neq \mathrm{MAC}(k_{\mathrm{MAC}}, c_{\mathrm{CBC}})$, and the $\mathtt{dec}_{\mathtt{cs}}(c)$-query to CS_{N^*} would return $\bot$. Thus the additional check introduced in Game 17 only returns $\bot$ if $\mathtt{dec}_{\mathtt{cs}}(c)$ would have returned $\bot$ in Game 16 anyway. Notice also that the $\mathtt{dec}_{\mathtt{cs}}(c)$-query does not perform queries to $\mathcal{O}_{\mathrm{IC}}$, thus the probability of $\mathsf{EarlyIC}^*$ does not change. Hence

$$\Pr[\mathsf{EarlyIC}^* : \textit{Game } 16] = \Pr[\mathsf{EarlyIC}^* : \textit{Game } 17]. \tag{19}$$

Let $\mathsf{DecChall}$ denote the event that a $\mathtt{dec}_{\mathtt{kem}}(c^*_{\mathrm{KEM}})$-query is sent to the fake KEM-oracle.

By definition, $\mathsf{EarlyIC}^*$ implies that an $\mathcal{O}_{\mathrm{IC}}(\cdot, k^*_{\mathrm{IC}}, \cdot)$-query is performed. Notice, however, that in Game 17, k^*_{IC} is only ever accessed by $\mathtt{program}_{\mathtt{cs}}(R^*, \cdot)$-queries to CS_{N^*} and by $\mathtt{dec}_{\mathtt{kem}}(c^*_{\mathrm{KEM}})$-queries to the fake KEM oracle. (The ciphertext c^*_{KEM} produced by the fake KEM oracle is independent of k^*_{IC}.) Furthermore, by definition of $\mathtt{FCRetrieve}$, a $\mathtt{program}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*} only occurs after a $\mathtt{getdk}_{\mathtt{cs}}()$-query to CS_{N^*}. Due to the change from Game 14, we abort when a $\mathtt{getdk}_{\mathtt{cs}}()$-query is sent to CS_{N^*}, thus no $\mathtt{program}_{\mathtt{cs}}(R^*, \cdot)$-query is sent to CS_{N^*}. Thus, unless $\mathsf{DecChall}$ occurs, k^*_{IC} is never accessed. Since k^*_{IC} has superpolynomial length, it follows that there is a negligible μ such that

$$\Pr[\mathsf{EarlyIC}^* : \textit{Game } 17] \leq \Pr[\mathsf{DecChall} : \textit{Game } 17] + \mu. \tag{20}$$

We proceed to bound $\Pr[\mathsf{DecChall} : \textit{Game } 17]$. Thus, assume that $\mathsf{DecChall}$ occurs in an execution of Game 17. By definition of $\mathsf{DecChall}$, this implies that a $\mathtt{dec}_{\mathtt{kem}}(c^*_{\mathrm{KEM}})$-query is sent the fake KEM-oracle. This, again, implies that there was a $\mathtt{dec}_{\mathtt{cs}}((c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t))$-query to CS_{N^*} with with $c_{\mathrm{KEM}} = c^*_{\mathrm{KEM}}$. Due to the additional check introduced in Game 17, the $\mathtt{dec}_{\mathtt{kem}}(c^*_{\mathrm{KEM}})$-query to the fake KEM-oracle is only reached if additionally $t = \mathrm{MAC}(k^*_{\mathrm{MAC}}, c_{\mathrm{CBC}})$. Furthermore, $\mathtt{FCRetrieve}$ only sends a $\mathtt{dec}_{\mathtt{cs}}((c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t))$-query to CS_{N^*} if $(c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t) \notin \mathit{cipher}_{N^*}$. Since $(c^*_{\mathrm{KEM}}, c^*_{\mathrm{CBC}}, t^*) \in \mathit{cipher}_{N^*}$ (the $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query to CS_{N^*} added that ciphertext to cipher_{N^*}), it follows that $(c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t) \neq (c^*_{\mathrm{KEM}}, c^*_{\mathrm{CBC}}, t^*)$. Assume that $c_{\mathrm{CBC}} = c^*_{\mathrm{CBC}}$. Then $t^* \overset{(*)}{=} \mathrm{MAC}(k^*_{\mathrm{MAC}}, c^*_{\mathrm{CBC}}) = \mathrm{MAC}(k^*_{\mathrm{MAC}}, c_{\mathrm{CBC}}) = t$ where $(*)$ follows from the way t^* is chosen in the $\mathtt{fakeenc}_{\mathtt{cs}}(R^*, \cdot)$-query. This contradicts $(c_{\mathrm{KEM}}, c_{\mathrm{CBC}}, t) \neq (c^*_{\mathrm{KEM}}, c^*_{\mathrm{CBC}}, t^*)$. Thus we have $c_{\mathrm{CBC}} \neq c^*_{\mathrm{CBC}}$.

Summarizing, whenever $\mathsf{DecChall}$ occurs, we have that $c_{\mathrm{CBC}} \neq c^*_{\mathrm{CBC}}$, but $t = \mathrm{MAC}(k^*_{\mathrm{MAC}}, c_{\mathrm{CBC}})$. Furthermore, k^*_{MAC} is only used to compute $t^* := \mathrm{MAC}(k^*_{\mathrm{MAC}}, c^*_{\mathrm{CBC}})$ (this occurs only once), and to perform checks of the form $t \overset{?}{=} \mathrm{MAC}(k^*_{\mathrm{MAC}}, c_{\mathrm{CBC}})$. (The ciphertext c^*_{KEM} produced by the fake KEM oracle is independent of k^*_{MAC}.) Hence the adversary has produced as forgery (t, c_{CBC}) with respect to the MAC-key k^*_{MAC}. Since we assumed that MAC is a one-time MAC (as defined in [CS03]), a polynomial-time adversary produces such a forgery only with negligible probability. Thus $\Pr[\mathsf{DecChall} : \textit{Game } 17]$ is negligible.

With (20), we get that $\Pr[\mathsf{EarlyIC}^* : \textit{Game } 17]$ is negligible. From (16)–(19), we get $\Pr[\mathsf{EarlyIC}^* : \textit{Game } 13] \approx \Pr[\mathsf{EarlyIC}^* : \textit{Game } 17]$. Hence $\Pr[\mathsf{EarlyIC}^* : \textit{Game } 13]$ is negligible. With (15), this implies that $\Pr[\mathsf{EarlyIC} : \textit{Game } 12]$ is negligible. By definition of Game 12, this means that in an interaction of $\mathcal{A}$ and FC, $\Pr[\mathsf{EarlyIC}]$ is negligible. □

From Lemmas 19 and 20, we immediately get our final result:

Theorem 21 *The hybrid encryption scheme* $(K_{\mathrm{hyb}}, E_{\mathrm{hyb}}, D_{\mathrm{hyb}})$ *is PROG-KDM secure with respect to the ciphertext simulator from Definition 8.*

References

[BDPR98] Mihir Bellare, Amit Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *Advances in Cryptology, Proceedings of CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer-Verlag, 1998. Extended version online available at `http://eprint.iacr.org/1998/021.ps`.

[BDU08] Michael Backes, Markus Dürmuth, and Dominique Unruh. Oaep is secure under key-dependent messages. In *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 506–523. Springer, December 2008.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In David Wagner, editor, *Proceedings of CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125. Springer, 2008.

[BHK12] Florian Böhl, Dennis Hofheinz, and Daniel Kraschewski. On definitions of selective opening security. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 522–539. Springer, 2012.

[BPS07] Michael Backes, Birgit Pfitzmann, and Andre Scedrov. Key-dependent message security under active attacks – BRSIM/UC-soundness of symbolic encryption with key cycles. In *Proc. of 20th IEEE Computer Security Foundation Symposium (CSF)*, June 2007. Preprint on IACR ePrint 2005/421.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR94] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology: EUROCRYPT '94*, volume 950 of *LNCS*, pages 92–111. Springer, 1994.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In *Proc. 9th Annual Workshop on Selected Areas in Cryptography (SAC)*, pages 62–75, 2002.

[CCS09] Jan Camenisch, Nishanth Chandran, and Victor Shoup. A public key encryption scheme secure against key dependent chosen plaintext and adaptive chosen ciphertext attacks. In Antoine Joux, editor, *Eurocrypt 2009*, volume 5479 of *LNCS*, pages 351–368. Springer, 2009.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Twenty-Eighth Annual ACM Symposium on Theory of Computing, Proceedings of STOC 1995*, pages 639–648. ACM Press, 1996. Extended version online available at `http://www.wisdom.weizmann.ac.il/~oded/PS/tr682.ps`.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology: EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, 2001.

[CPS08] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 1–20. Springer, 2008.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003. Online available at `http://shoup.net/papers/cca2.ps`.

[DNRS03] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry Stockmeyer. Magic functions. *Journal of the ACM*, 50(6):852–921, 2003. Extended version online available at `http://www.wisdom.weizmann.ac.il/~naor/PAPERS/magic.ps`.

[EMST76] William F. Ehrsam, Carl H. W. Meyer, John L. Smith, and Walter L. Tuchman. Message verification and transmission error detection by block chaining. US Patent 4074066, 1976.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, April 1984.

[Nie02] Jesper B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology, Proceedings of CRYPTO '02*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer-Verlag, 2002.

[RS92] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Advances in Cryptology, Proceedings of CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer-Verlag, 1992. Online available at `http://research.microsoft.com/crypto/dansimon/me.htm`.

[Sha49] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT '97*, volume 1233 of *LNCS*, pages 256–266. Springer, 1997.

Symbolindex

η	Security parameter	
ek	Encryption key	
dk	Decryption key	
$x \leftarrow A$	Pick x according to algorithm/distribution A	
$x \overset{\$}{\leftarrow} S$	Pick x uniformly from set S	
reg_R	Register R (in real/fake challenger)	
$\mathcal{O}$	Denotes some oracle (usually the random oracle)	7
$\|x\|$	Absolute value of x	
$\mathcal{A}$	Usually denotes the adversary	
$\mathbb{N}$	Set of natural numbers $1, 2, \ldots$	7
RC	Real challenger	9
$cipher_N$	Set of ciphertexts produced in real/fake challenger in session N	9
$R := \mathtt{getek}_{\mathtt{ch}}(N)$	Real/fake challenger query: Load encryption key (session N)	9
$R := \mathtt{getdk}_{\mathtt{ch}}(N)$	Real/fake challenger query: Load decryption key (session N)	9
$R := \mathtt{eval}_{\mathtt{ch}}(C, R_1, \ldots, R_n)$	Real/fake challenger query: Evaluate circuit C	9
$R := \mathtt{enc}_{\mathtt{ch}}(N, R_1)$	Real/fake challenger query: Encrypt reg_{R_1} (session N)	9
$\mathtt{oracle}_{\mathtt{ch}}(x)$	Real/fake challenger query: Oracle query	9
$\mathtt{dec}_{\mathtt{ch}}(N, c)$	Real/fake challenger query: Decrypt c (session N)	9
`forbidden`	Output denoting attempted decryption of challenge ciphertext	9
$\mathtt{reveal}_{\mathtt{ch}}(R_1)$	Query type of the real/fake challenger	9
$\mathtt{fakeenc}_{\mathtt{cs}}(R, l)$	Ciphertext simulator query: Produce fake encrytion	9
$\mathtt{dec}_{\mathtt{cs}}(c)$	Ciphertext simulator query: Decrypt	9
$\mathtt{enc}_{\mathtt{cs}}(R, m)$	Ciphertext simulator query: Encryption (non-fake)	9
$\mathtt{getek}_{\mathtt{cs}}()$	Ciphertext simulator query: Get encryption key	9
$\mathtt{getdk}_{\mathtt{cs}}()$	Ciphertext simulator query: Get decryption key	9
$\mathtt{program}_{\mathtt{cs}}(R, m)$	Ciphertext simulator query: Program the oracle	9
CS	Ciphertext simulator	9
FC	Fake challenger	10
`FCRetrieve`	Retrieve function of FC	11

plain_N	Plaintexts of fake encryptions from CS_N (part of state of `FCRetrieve`)	11
$\ell_{ek}(\eta)$	Length of an encryption key	11
$\ell_{dk}(\eta)$	Length of a decryption key	11
$\ell_c(\eta, l)$	Length of the encryption of a ciphertext of length l	11
`FCLen`	Length function of FC	11
MAC	Algorithm computing a MAC	12
ℓ_{MAC}^K	Length of the key of MAC	12
ℓ_{MAC}^T	Length of tag produced by MAC	12
ℓ_{IC}^K	Key length of the ideal cipher	12
ℓ_{IC}^M	Message length of the ideal cipher	12
$\mathcal{O}_{\mathrm{IC}}$	Ideal cipher	12
K_{KEM}	Key generation of the KEM	12
E_{KEM}	Encryption algorithm of the KEM	12
D_{KEM}	Decryption algorithm of the KEM	12
k_{MAC}	Key for MAC	12
k_{IC}	Key for ideal cipher $\mathcal{O}_{\mathrm{IC}}$	12
cbcpad	Padding function for CBC	12
E_{CBC}	Code block chaining mode (encryption function)	12
D_{CBC}	Code block chaining mode (decryption function)	12
$\mathcal{D}_{\mathrm{IV}}$	Distribution of initialization vectors in CBC mode	12
K_{hyb}	Key generation of hybrid encryption scheme	13
E_{hyb}	Encryption algorithm of hybrid encryption scheme	13
D_{hyb}	Decryption algorithm of hybrid encryption scheme	13
$\mathcal{O}_{\mathrm{IC,lazy}}$	Lazy ideal cipher	13
$\tilde{\mathcal{O}}_{\mathrm{IC,lazy}}$	Intermediate lazy ideal cipher	14
$\mathcal{N}$	Set of all N used by adversary	14
$\mathcal{R}$	Set of all R used by adversary	14
$\approx$	Negligibly close	15
Rand	Fixed randomness used in some games	17
ek_N	Encryption key for CS_N (fixed at beginning)	17
dk_N	Decryption key for CS_N (fixed at beginning)	17
$k_{\mathrm{MAC}}^{N,R}$	MAC-key used by CS_N for register R (fixed at beginning)	17
$k_{\mathrm{IC}}^{N,R}$	Ideal cipher key used by CS_N for register R (fixed at beginning)	17
$c_{\mathrm{KEM}}^{N,R}$	KEM-ciphertext chosen by CS_N for register R (fixed at beginning)	17
t_{fakeenc}^R	Random tape used by CS_N in $\mathtt{fakeenc}_{\mathtt{cs}}$-queries	17
$t_{\mathcal{A}}$	Random tape used by the adversary (fixed at beginning)	17
$\mathcal{O}_{\mathrm{IC}}^0$	The ideal cipher before any reprogramming	17

Index