The Indifferentiability of the Duplex and its Practical Applications*

Jean Paul Degabriele[1,3] Marc Fischlin[2] Jérôme Govinden[3]

[1] Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, UAE
jeanpaul.degabriele@tii.ae
[2] Cryptoplexity, Technische Universität Darmstadt, Germany
marc.fischlin@cryptoplexity.de
[3] CNS, Technische Universität Darmstadt, Germany
jerome.govinden@tu-darmstadt.de

Abstract. The Duplex construction, introduced by Bertoni *et al.* (SAC 2011), is the Swiss Army knife of permutation-based cryptography. It can be used to realise a variety of cryptographic objects—ranging from hash functions and MACs, to authenticated encryption and symmetric ratchets. Testament to this is the STROBE protocol framework which is a software cryptographic library based solely on the Duplex combined with a rich set of function calls. While prior works have typically focused their attention on specific uses of the Duplex, our focus here is its *indifferentiability*. More specifically, we consider the indifferentiability of the Duplex construction from an *online random oracle*—an idealisation which shares its same interface. As one of our main results we establish the indifferentiability of the Duplex from an online random oracle. However indifferentiability only holds for the standard Duplex construction and we show that the full-state variant of the Duplex cannot meet this notion. Our indifferentiability theorem provides the theoretical justification for the security of the Duplex in a variety of scenarios, amongst others, its use as a general-purpose cryptographic primitive in the STROBE framework. Next we move our attention to AEAD schemes based on the Duplex, namely SpongeWrap, which is the basis for NIST's Lightweight Cryptography standard Ascon. We harness the power of indifferentiability by establishing that SpongeWrap offers security against key-dependent message inputs, related-key attacks, and is also committing.

1 Introduction

Permutation-based cryptography supplants the block cipher with an unkeyed public permutation as the central building block for realising symmetric-key cryptographic primitives. The approach has led to innovative and versatile designs in symmetric-key cryptography and is now establishing itself as a field of study in its own right. The SHA-3 standard is a prime example of its success, which brings to light the versatility of the approach in yielding a variety of primitives such as SHA-3, SHAKE, KMAC, TupleHash, and ParallelHash [KCP16] from the same underlying construction, i.e., the Sponge [BDPV08]. The fact that the Sponge turned out to be such a versatile construct is not that surprising when one considers the central role that the random oracle [BR93] plays in cryptography in general. The power of the random oracle comes both from its conceptual simplicity and its ability to be a drop-in replacement for

a multitude of cryptographic primitives. With minor cosmetic changes, a random oracle can serve as a one-way function, a collision-resistant hash, a pseudorandom generator, a randomness extractor, or a pseudorandom function. In turn, the Sponge inherits all these lucrative properties from the random oracle through its indifferentiability theorem, under the assumption that the underlying public permutation acts as a random permutation.

In 2011, permutation-based cryptography received a boost from the advent of the Duplex construction [BDPV12]. The Duplex construction supersedes the Sponge in that it can perfectly mimic the Sponge but allows for yet more functionality. Whereas the Sponge operates in two phases, an 'absorb' phase in which an input is gradually mixed into the state followed by a 'squeeze' phase in which an output is gradually extracted from the state until it is of the desired size, the Duplex is capable of absorbing input and squeezing output simultaneously. The main motivation behind the design of the Duplex is that it allows for more efficient permutation-based AEAD schemes. In principle an AEAD scheme can be constructed from two Sponge instances, one operating as a pseudorandom generator and the other serving as a message authentication code. However that would entail passing over the data twice, whereas the Duplex yields a more efficient scheme that can process the data in *one pass*. A one-pass AEAD scheme based on the Duplex, known as SpongeWrap, was proposed in [BDPV12]. Many candidates of the CAESAR competition and the NIST lightweight competition were based on SpongeWrap. This includes in particular Ascon [DEMS16], the new NIST lightweight crypto standard and CAESAR's primary choice for authenticated encryption.

More recently, the Duplex has taken a life of its own among practitioners who, having recognised its elegance and versatility, have written cryptographic libraries that are based solely on this one primitive. Two such examples are the STROBE protocol framework by Mike Hamburg [Ham17] and the Stateful Hash Object (SHO) within Trevor Perrin's Noise Protocol framework [Per18]. STROBE and SHO are fairly similar frameworks for realising protocols from Duplex instances. A typical scenario would have two parties maintaining one or more Duplex instances to encrypt and decrypt messages, possibly incorporating a symmetric ratchet functionality, where either party can produce or verify, a MAC tag or a hash digest, of the transcript at any point in time. The tenet behind frameworks like STROBE and SHO is that the Duplex behaves like an idealised cryptographic primitive that can be plugged into any protocol and it will retain its security. Clearly, this would hold true if the Duplex, like the Sponge, were backed by an indifferentiability theorem, but unfortunately no such result exists in the literature. In fact, it is not even clear what ideal-functionality behaviour one can expect from the Duplex.

When the Duplex was first proposed in [BDPV12], it was observed that a sequence of queries to the Duplex could be evaluated by mapping that sequence of inputs to a separate input sequence and then feeding that to the Sponge. Accordingly, it was then argued that if the Sponge is secure then the Duplex must also be secure. This argument was formalised in the security proof of SpongeWrap in that same paper, by first proving the security of SpongeWrap in the random oracle model and then invoking the indifferentiability of the Sponge. While perfectly legitimate, this security treatment is lacking in some respects. By reducing the security of SpongeWrap directly to that of the Sponge, the security of the Duplex was left uncovered. In particular, a formal security definition for the Duplex was not provided. This left unattended the question of what security one should expect from the Duplex, thereby failing to recognise its merit as a cryptographic primitive in its own right. In principle one could rely on the mapping from the Duplex to the Sponge but we contend that this approach is cumbersome, unintuitive, and error-prone. Another drawback of this approach is that the mapping from the Duplex to the Sponge entails an unnecessary degradation in the resulting security bound. Indeed, as we expand upon later, the security bound for SpongeWrap that results from applying this method is actually worse than that presented in [BDPV12], an issue that thus far seems to have gone unnoticed.

Motivated by the popularity of the Duplex and its varied applications, we provide the first treatment of the Duplex in the indifferentiability framework of Maurer, Renner, and Holenstein [MRH04]. While there

have been a number of works that consider the security of the Duplex in terms of indistinguishability, our proof of indifferentiability is beneficial for a number of reasons. First off, we provide an idealisation of the Duplex that gives a direct and intuitive way to reason about the Duplex construction and the security of its applications. We call this ideal functionality the *online random oracle*. The composition theorem then guarantees that we can replace an instance of the Duplex with the online random oracle, with the usual care regarding multi-stage games [RSS11]. Thus our treatment provides the formal justification for employing the Duplex as a general-purpose primitive, as done in STROBE and SHO for instance. Our indifferentiability result is also useful in the case where the Duplex is endowed with a secret key, as in an AEAD scheme. In particular it allows for a simpler and more direct security analysis in the online random oracle model as opposed to a more involved analysis in the random permutation model. Besides simplifying the analysis in the standard AEAD setting, the online random oracle model also enables an easier treatment with respect to stronger security models, such as related-key and key-dependent-input security.

1.1 Summary of Our Contributions

The Online Random Oracle. Before we can delve into examining the indifferentiability of the Duplex with respect to an ideal permutation we need to identify a suitable ideal functionality. Our answer is the online random oracle (ORO), which is a natural adaptation of the random oracle to a stateful primitive that can be interacted with in an online manner, allowing to iterate over the outputs by appending new inputs. Besides enabling our treatment of indifferentiability, we believe this abstraction to be more intuitive, concise, and direct than having to map the sequence of inputs for the Duplex to a sequence of inputs for the Sponge and then replacing it with a random oracle.

A peculiarity of the ORO is that it only takes inputs of a fixed size, which in turn allows us to analyse the Duplex without padding. While this may seem more restrictive at first, since the Duplex can only take inputs of a certain size, it actually renders our treatment more general. In the Sponge construction, padding is not only required to allow for arbitrary-length inputs but it is also necessary for security and indifferentiability to hold. In particular, padding is required to delineate the boundary between input and output, thereby excluding the possibility of two inputs (say x and $x \,\|\, 0^r$) having related outputs. Now, in order to map the Duplex to the Sponge, the Duplex needs to include 'Sponge-compliant' padding in every round as in [BDPV12]. Consequently, a security treatment based on this mapping relies crucially on the presence of this padding. Lifting the need for sponge-compliant padding removes this extra layer of complexity, reducing padding from a matter of security to one of functionality, and allows practitioners the freedom to use the padding that best fits their application. While some AEAD constructions include sponge-compliant padding, e.g., Ketje [BDP+16], others do not, e.g. Ascon [DEMS16]. Accordingly our treatment serves in part to justify this latter type of constructions. Finally, we provide the ORO with a more pragmatic API that supports multiple concurrent sessions and the ability to switch easily between them. This serves to bring the ORO closer to practice, and although it may seem like a superficiality the specifics of the API have an effect on the simulator's efficiency used to prove indifferentiability.

Indifferentiability of the Duplex. We start by considering a variant of the Duplex that appeared in [MRV15, DMV17, DM19, Men22], known as the full-state Duplex, and show that indifferentiability is out of reach for this variant. This result justifies our choice of restricting our attention to the Duplex without full state absorption, requiring that the encoding functions cannot affect the entire input. We then go on to prove our main result of indifferentiability between the Duplex and the ORO. This result can be leveraged in a variety of settings. For instance, an immediate consequence is that the Duplex yields an online randomness extractor [DPR+13]. Alternatively, by lifting constructions based on the Duplex to the ORO model one can employ proof techniques similar to the random oracle model, like 'programmability'

and 'extractability'. We emphasize that we obtain a fairly good indifferentiability bound and that our simulator is very efficient. This serves to make our result all the more usable and meaningful in practice, resulting in good bounds when composability is invoked.

Stronger security for SpongeWrap. Our indifferentiability theorem is profitable also for the case where the Duplex is used as keyed primitive, its most popular application being the realisation of one-pass AEAD schemes. We use it to revisit the security of SpongeWrap in more demanding settings that, to the best of our knowledge, have not been considered so far: key-dependent message (KDM) security, related-key attacks (RKA), and commitment (CMT). KDM security guarantees security when an AEAD scheme is used to encrypt a message that depends on the key or an application where a number of keys are used to encrypt each other in the form of a cycle. Such cases arise in disk encryption, hardware security modules, and key management/distribution systems. RKA security models scenarios such as fault-injection attacks where an adversary may obtain ciphertexts evaluated under modified key values which is a concern when an adversary gets access to the hardware the scheme is running on. Recent works [GLR17, ADG+22, LGR21] have highlighted several practical issues, in relation to message franking, password-based AEAD, key rotation, and envelope encryption, that arise when an AEAD is non-committing.

We prove SpongeWrap to be KDM-AEAD secure as defined by Bellare and Keelveedhi [BK11]. To this end we first prove that the (nonce-based SpongeWrap version of the) ORO achieves KDM-AEAD security. Then we can apply the composition theorem of Barbosa and Farshim [BF18] for indifferentiable Authenticated Encryption to conclude that the nonce-based SpongeWrap of the Duplex is also secure. We provide a similar treatment covering security against related-key attacks (RKA-AEAD) and committing security (CMT-AEAD). Our analysis serves to highlight the fact that AEAD schemes based on the Duplex automatically offer these attractive properties without requiring any alteration that would detriment their efficiency or any additional assumption beyond what is already required.

2 Preliminaries

Notation. Unless otherwise stated, an algorithm may be randomised. For any algorithm A we use $y \leftarrow \mathsf{A}(x_1, x_2, \ldots)$ to denote the process of running A on the indicated inputs and fresh random coins, and assigning the output to y. By convention the running time of an adversary refers to the sum of its actual running time and the size of its description. We generically refer to the resources of an adversary as any subset of the following quantities: its running time, the number of queries that it makes to its oracles, and the total length (in bits) of its oracle queries. We write $\mathsf{B.sub1}$, $\mathsf{B.sub2}$, ... to denote a group of algorithms that share state and refer to them collectively as B.

If S is a set then $|S|$ denotes its size, and $z \twoheadleftarrow S$ denotes the process of selecting an element from S uniformly at random and assigning it to z. When assigning a value y to variable x or table entry $T[x]$ we write $x \leftarrow y$ or $T[x] \leftarrow y$. We assume throughout that tables are initialized to $\perp$ entries and sets are initially empty. When T is a table, we write $x \in T$ to indicate that the table entry $T[x] \neq \perp$. For sets X and Y, the set of all functions mapping from X to Y is denoted by $\mathsf{Func}(X, Y)$.

For a bit b and a positive integer n, we denote by b^n the string composed of b repeated n times. With $\{0,1\}^n$ we denote the set of all binary strings of length n, and $\{0,1\}^*$ denotes the set of all binary strings of finite length. The empty string is represented by ε. For any two strings u and v, $|u|$ denotes the length of u in bits, $u \| v$ denotes their concatenation, and $u \oplus v$ denotes their bitwise XOR operation. For $n \in \mathbb{N}$ and $u \in \{0,1\}^*$, $(u_1, \ldots, u_\ell) \xleftarrow{n} u$ denotes the n-bit parsing of u where $|u_i| = n$ for all $1 \leq i < \ell$ and $0 < |u_i| \leq n$. For a boolean expression exp, we denote its encoding into a single bit as $\langle \mathsf{exp} \rangle$. We use $u[i, j]$ to denote the substring of u from bit i to bit j inclusive, where the indexes start at 1. For a positive integer $n \leq |u|$, we use $\lfloor u \rfloor_n$ to denote the string obtained by truncating u to its leftmost n bits. We denote by $[t]$

$\mathsf{Dup.init}(X)$	$\mathsf{Dup.next}(X, id)$
if $\lvert X\rvert \neq r$	**if** $\lvert X\rvert \neq r$
return $\perp$	**return** $\perp$
$id \leftarrow \mathsf{GenID}$	$Y_{id} \leftarrow S_{id} \oplus \mathsf{encode}_{i[id]}(X)$
$S_{id} \leftarrow IV$	$S_{id} \leftarrow \mathsf{p}(Y_{id})$
$Y_{id} \leftarrow S_{id} \oplus \mathsf{encode}_0(X)$	$Z \leftarrow \mathsf{decode}_{i[id]+1}(S_{id})$
$S_{id} \leftarrow \mathsf{p}(Y_{id})$	$i[id] \leftarrow i[id] + 1$
$Z \leftarrow \mathsf{decode}_1(S_{id})$	**return** Z
$i[id] \leftarrow 1$	
return (Z, id)	

Figure 1: The Duplex construction from a public permutation p, where r is the (input) rate and encode_i maps the r-bit input X to $b = r + c$ bits resp. decode_i maps the permutation's output to r bits again. Algorithm GenID generates a unique identifier in order to be able to handle multiple sessions simultaneously.

the set $\{1, \ldots, t\}$, $t \in \mathbb{N}$.

We will at times make use of the code-based game-playing framework by Bellare and Rogaway [BR06]. Here the interaction between a game and the adversary is implicit, whereby the adversary is given as its input the output of the initialize procedure, it has oracle access to the other procedures described in the game, and its output is fed into the finalize procedure. The output of the finalize procedure is the output of the game. For a game Gm and an adversary $\mathcal{A}$, $\mathrm{Gm}^{\mathcal{A}} \Rightarrow x$ denotes the event that Gm outputs x when interacting with $\mathcal{A}$. Similarly, $\mathcal{A}^{\mathrm{Gm}} \Rightarrow y$ denotes the event that $\mathcal{A}$ outputs y when interacting with Gm.

AEAD Syntax. An authenticated encryption scheme with associated data $\mathsf{SE} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ is a triple of efficient algorithms such that:

- The key generation algorithm K is randomised, takes no input, and samples k-bit strings according to some distribution.
- The encryption algorithm $\mathsf{E} : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ is deterministic and takes as input a secret key K, a nonce N, associated data A, and a message M to return a ciphertext C.
- The decryption algorithm $\mathsf{D} : \{0,1\}^k \times \{0,1\}^n \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$ is deterministic and takes as input a secret key K, a nonce N, associated data A, and a ciphertext C to return a message $M \in \{0,1\}^*$ or $M = \perp$ indicating an invalid set of inputs.

We require that an authenticated encryption scheme be both *correct* and *tidy*. Correctness requires that for all K, N, A, M it hold that if $\mathsf{E}(K, N, A, M) = C$ then $\mathsf{D}(K, N, A, C) = M$. In a similar manner, tidiness requires that for all K, N, A, C it hold that if $\mathsf{D}(K, N, A, C) = M \neq \perp$ then $\mathsf{E}(K, N, A, M) = C$. Moreover, encryption must be length regular, i.e., we assume a function $\mathsf{cl}(\cdot, \cdot)$ such that $|C| = \mathsf{cl}(|A|, |M|)$ for any C returned by $\mathsf{E}(\cdot, \cdot, A, M)$.

2.1 The Duplex

The Duplex is a construction similar to the Sponge that is based on some public permutation p over b-bit strings. A user can interact with the Duplex either via an initialisation call $\mathsf{Dup.init}$ or a next call $\mathsf{Dup.next}$. An initialisation call starts a session which sets the initial state to some fixed value IV and processes the given input. The session can then be updated via one or more next calls. Both types of calls take an

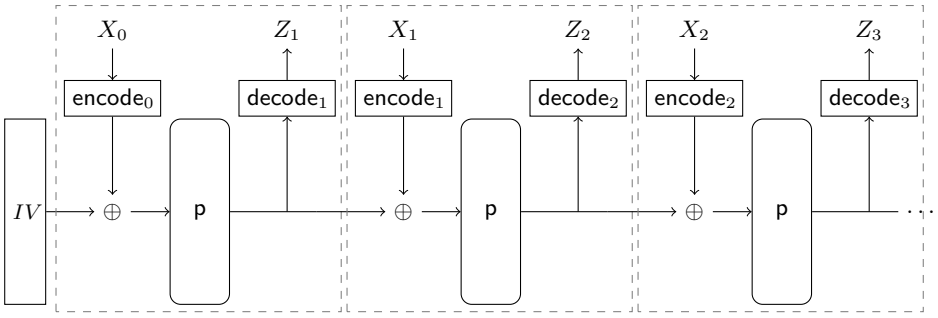

Figure 2: Duplex based on permutation p with round-dependent encoding and decoding. Note that the encoding and decoding functions in each round only access the same rate part of the intermediate state, i.e., the same bit positions, independent of the round. Subscripts in inputs and outputs as well as encoding and decoding functions denote the number of p applications which have been made so far.

input of size r bits and return an output of r bits. The value r is called the rate of the Duplex. A detailed pseudocode description of the Duplex is shown in Fig. 1.

The internal working of the Duplex is also described pictorially in Figure 2. The function encode_i works by appending the bits 0^c to its inputs and decode_i reverses this operation by truncating the rightmost c bits of its input. The quantity c is known as the capacity where $c = b - r$. We will occasionally consider more general encodings but unless stated otherwise they are to be interpreted as just described. At any point the state of the Duplex consists of a b-bit string S. We use S_R and S_C to denote the leftmost r bits and the rightmost c bits of S respectively. We will slightly abuse terminology by referring to S_R and S_C as the rate component and capacity component of S.

Typically, constructions based on the Duplex will need to use some form of padding to handle variable length inputs. Our formulation of the Duplex does not consider such padding to be part of it. This increases the generality of our treatment, in that security, specifically indifferentiability, will not rely on the particular type of padding being used as long as the input is of the right size. In contrast, note that for the Sponge to be indifferentiable from a random oracle it requires a specific type of padding, even if the input size is restricted to be an integral multiple of the rate.

Finally, note that we let the Duplex support multiple sessions. We do so by extending the interface to return a session identifier after each initialisation call and require that every subsequent next call refer to a specific session. Past formulations of the Duplex could support only a single session at any point in time where an initialisation call would automatically erase the state of the prior session. In contrast our interface allows one to easily switch between sessions without having to repeat the prior inputs in order to advance the Duplex to its last state. This reflects a more realistic implementation of the Duplex that relieves its users from making unnecessary queries.

2.2 Prior Security Treatments of the Duplex

The Duplex construction first appeared in [BDPV12] as a way of 'duplexing' the Sponge in order to realise *one-pass* AEAD schemes, like SpongeWrap, from public permutations. The crucial observation behind the design of the Duplex was that a call to Duplex could be mapped to a call to the Sponge, thereby reducing an instance of the Duplex to a sequence of Sponge calls. This was stated as the Duplexing-Sponge lemma (cf. [BDPV12, Lemma 3]), and was then used to analyse the AEAD security of SpongeWrap. Namely, in the AEAD game, Duplex queries were translated to Sponge queries to allow for an analysis of SpongeWrap in the random oracle model. The result was then translated to the random permutation model by leveraging the indifferentiability of the Sponge.

While this observation was very insightful and valuable in leading to the creation of the Duplex, there are significant limitations in resting the security of the Duplex on this approach. To begin with, it lacks a compact and intuitive security definition for the Duplex. This limits our understanding of the Duplex as well as our ability to use it correctly and securely. For instance, there are fundamental differences between the Duplex and the Sponge, which are easy to overlook if we restrict ourselves to think of the former in terms of the latter. One case in point is the fact that the Duplex is a stateful construction whereas the Sponge is in contrast stateless. Moreover, mapping from Duplex to Sponge introduces an additional step in the analysis which adds unnecessarily to its complexity. Secondly, by viewing the Duplex in terms of the Sponge we are implicitly imposing any limitation that may be specific to the Sponge onto the Duplex. For instance, a subtle condition for this mapping to work is the inclusion of 'Sponge-compliant' padding in *every* permutation call within the Duplex—see [BDPV12, BDP$^+$16] for details. While such padding is necessary in the case of the Sponge for indifferentiability to hold true, as we will show, this turns out not to be the case for the Duplex.

Subsequent works on the Duplex, namely [MRV15, DMV17, DM19, Men22], have focused on the Full-State Keyed Duplex (FSKD)—a variant of the Duplex which is always keyed and that can admit inputs covering the full width b of the permutation. Exploiting the fact that this primitive is always keyed, these treatments focus on security in terms of *indistinguishability*. An overview of the Duplex variants and their security in the sense of indistinguishability can be found in [Men22]. Instead we focus here on the *indifferentiability* of the Duplex. As mentioned before, indifferentiability is impossible to achieve for the full-state keyed Duplex.

2.3 Other Related Work

Barbosa and Farshim have introduced the notion of indifferentiable Authenticated Encryption (iAE) [BF18]. Moreover they show that indifferentiable Authenticated Encryption automatically guarantees security against Key-Dependent Messages, Related-Key Attacks, and is committing. They present two ideal functionalities for AEAD corresponding to the offline and online syntaxes. Both are based on random injections, as introduced by Rogaway and Shrimpton [RS06]. Both idealisations, have appealing properties but they also result in rather strong objects. Unfortunately, as we explain in Section 7.2, it is not possible for a Duplex-based single-pass AEAD scheme, like SpongeWrap, to be indifferentiable from either of these functionalities. Our work addresses this gap by showing that SpongeWrap achieves these advanced security properties based on the indifferentiability of the Duplex, even if SpongeWrap is itself not iAE secure.

3 The Online Random Oracle

We propose the *Online Random Oracle* (ORO) as an idealisation of the Duplex construction. This functionality is similar in spirit to the Ideal Extendible Input Function (IXIF) described in [DMV17] for the Full-State Keyed Duplex. However, there are some important differences between the two—which we explain below. As the name suggests, the ORO functionality can be viewed as an online adaptation of the well-known random oracle functionality. A *session* is initiated with a call to ORO.init, after which the input and output of the ORO can be extended by r bits at a time via calls to ORO.next. The ORO is not forgetful, in the sense that distinct sessions are subject to the condition that a common prefix across their input sequences will result in a common prefix in their corresponding output sequences. A pseudocode description of the online random oracle is presented in Fig. 3. Internally, for each session id, the corresponding input sequence is mapped to a path (a string) P_{id}, where for each path a corresponding output is sampled uniformly at random and stored in a table.

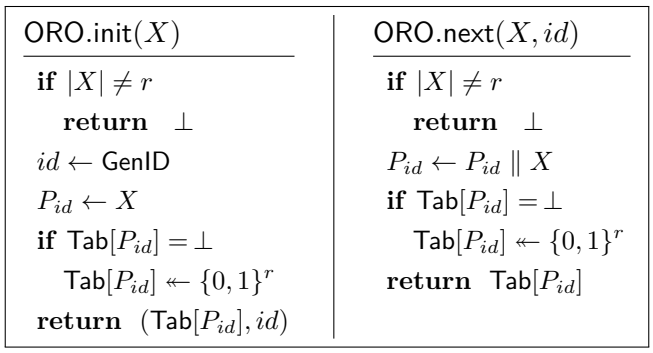

Figure 3: A lazy-sampled view of the Online Random Oracle functionality. Here, r is the input and output size and Tab is a table, initialized as empty by setting all entries to $\perp$. Algorithm GenID generates a unique identifier to support multiple sessions.

Sessions and Identifiers. Unlike a random oracle, the ORO is a stateful object which introduces some additional complexity in modelling it as a shared resource. Namely the output to a query depends on prior queries and when shared among multiple algorithms we need to ensure that the queries of one algorithm do not interfere with another algorithm's queries. We use sessions precisely to circumvent this issue. At each ORO.init call a unique session identifier is created and returned together with the output. A session identifier is simply a string and we do not impose any further restriction on its format except the existence of some algorithm GenID that can generate them uniquely and efficiently. Accordingly GenID can simply increment a counter value or a random string. Then in each ORO.next call the algorithm specifies which session should be updated. We require that any algorithm with access to the ORO only make ORO.next calls to session which it obtained from calls to ORO.init. Note that this requirement does not limit access to the ORO in any way, it simply ensures that distinct parties cannot interfere in each other's sessions. Moreover, all sessions, irrespective of which entity initiates them, are answered using the same randomness which can alternatively be viewed as being sampled all at once at the beginning of the game.

Differences Between the ORO and the IXIF. First let us observe that the ORO and the IXIF serve different purposes: the former one is tailored for *indifferentiability* whereas the other is intended for *indistinguishability*. Besides targeting distinct security notions, they also model different constructions. While the IXIF is intended as an idealisation of the full-state keyed Duplex, the ORO is an idealisation of the unkeyed Duplex without full-state absorption, and, accordingly, the two present different interfaces. A more subtle discrepancy is that the full-state keyed Duplex described in [DMV17] is "phased" differently from the standard Duplex [BDPV12], in that the input is *not* inserted at the same point where the output is extracted.

In addition to the main input the IXIF takes a flag and during an initialisation call it additionally takes a key handle and an initialisation vector. The flag determines how the internal path is constructed. If the flag is set, the path is constructed such that it also depends on the output that the IXIF returns. Otherwise the path is constructed as in the case of the ORO. The different phasing and flag introduced in [DMV17] is meant to reflect applications where the adversary's ability to overwrite the Duplex input is limited thereby allowing for a better security bound in such cases. Follow-up works have introduced yet other ways of phasing the Duplex, see [Men22] for a recent summary. In this work we stick with the original Duplex phasing and reflect the more conservative setting where the distinguisher can adaptively choose its input based on the prior output.

In the IXIF, the key handle reflects a multi-user setting where the distinguisher can query multiple instances of the IXIF—each instance corresponding to a distinct pair of key and initialisation vector.

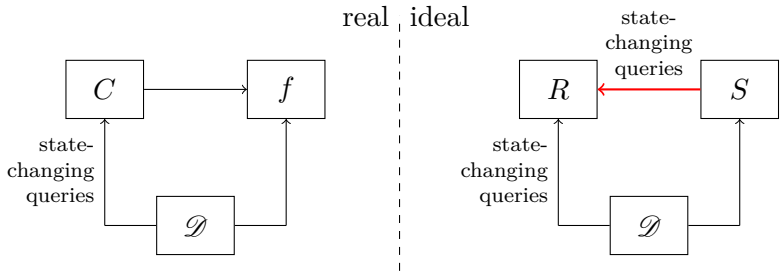

Figure 4: Issues with stateful constructions in the indifferentiability framework: The state changes by the simulator S in the ideal world may be detectable by the distinguisher $\mathscr{D}$.

Clearly the key handle serves a very different purpose from that of a session identifier in the ORO, but they also present a different interface. In contrast to the ORO, only one IXIF instance exists at any point in time and thus one cannot progress two or more instances simultaneously. That is, any sequence of inputs to a particular instance have to be evaluated consecutively from start to end. While this is without loss of generality a construction such as NORX [AJN14] that requires two or more simultaneous instances is forced to repeat all prior queries when switching from one instance to another.

4 Indifferentiability

In this section we recall the basic definitions of indifferentiability and discuss some peculiarities relevant to our results. Indifferentiability has been defined by Maurer et al. [MRH04] in terms of random systems; we use here the algorithmic approach of Coron et al. [CDMP05]:

Definition 4.1 *An algorithm* C *with access to an ideal primitive* f *is indifferentiable from an ideal object* R *if there exists a simulator* Sim *such that for any distinguisher* $\mathscr{D}$ *the advantage*

$$\mathbf{Adv}^{indiff}_{\mathsf{C},\mathsf{f},\mathsf{R},\mathsf{Sim}}(\mathscr{D}) = \left|\Pr\left[\mathscr{D}^{\mathsf{C}^{\mathsf{f}},\mathsf{f}} = 1\right] - \Pr\left[\mathscr{D}^{\mathsf{R},\mathsf{Sim}^{\mathsf{R}}} = 1\right]\right|$$

is negligible.

We note that we are usually interested in efficient simulators Sim.

4.1 Indifferentiability of Stateful Constructions and Session Identifiers

Our online random oracle functionality is stateful as it uses the path P_{id} of previous queries to evaluate next calls. Thus, whereas a random oracle is a resource that can be easily shared among multiple parties since queries are atomic, sharing an online random oracle requires a way of interleaving interdependent queries from different parties without cross interference. This issue arises in different aspects of the indifferentiability framework, from the composition theorem to the security definition itself.

When applying the composition theorem and transition from the real world to the ideal world, we may need to replace multiple instances of the real-world construction within the more complex cryptosystem, with a single instance of the ideal primitive. Thus the ideal functionality, and consequently the primitive's interface, must have a way of handling multiple sessions simultaneously. This issue arises even in the indifferentiability definition itself when transitioning from the real world to the ideal world, see Figure 4. In the real world only the distinguisher queries the construction and alters the state. In the ideal world both the distinguisher and the simulator may query the ideal object, and in most proofs the simulator does

query the ideal primitive to ensure matching answers. But then, without a means to differentiate among query sources, the distinguisher will be able to detect such state changes via calls to the construction as shown in the example below.

To address these issues, the ORO's syntax includes a session identifier id in order to differentiate among distinct sessions. In addition this syntax also permits this primitive to support multiple sessions simultaneously, which may be required by the higher-level system. It is assumed that the ORO session identifiers are hard to guess.

Assume that there are no session identifiers to differentiate queries to the ORO and consider the following scenario. The distinguisher (in either world) first queries $\mathsf{init}(X_0)$ and then $\mathsf{next}(X_1)$ and $\mathsf{next}(X_2)$ for random X_0, X_1, X_2 to the construction oracle. Let Z be the result of the final call. Afterwards it implicitly "resets" the state of the construction oracle by making another call $\mathsf{init}(X_0)$ followed by $\mathsf{next}(X_1)$ to $P_{id} = X_0||X_1$. Now the distinguisher makes a call $IV \oplus \mathsf{encode}_0(X_0)$ to the primitive oracle. Afterwards it makes a query $\mathsf{next}(X_2, id)$ to the construction oracle and then checks that it obtains once more the same result Z.

In the real world the state $P = X_0||X_1$ of the construction before $\mathscr{D}$'s final call $\mathsf{next}(X_2, id)$ will be sound and make the oracle return the same result Z as earlier. In contrast, in the ideal world our simulator will have changed the state to $P = X_0$ when simulating the primitive call about X_0, in order to provide a consistent answer. It is conceivable that any simulator would need to change the state, and if this happens then reconstructing the expected state $P_{id} = X_0||X_1$ is impossible, because the random value X_1 is unknown to the simulator. But then, with overwhelming probability, the distinguisher gets a mismatching value in the final call $\mathsf{next}(X_2, id)$.

We note that the indifferentiability framework already assumes different interfaces for distinguisher and simulator such that one could keep one state for $\mathscr{D}$ and a separate one for Sim to avoid the problems above. We have chosen the version with identifiers id since it is closer to real-life scenarios of various instances. In terms of our indifferentiability proof for Duplex it also allows the simulator to minimize the calls to the online random oracle to simulate answers for the primitive.

4.2 Indifferentiability and Multistage Games

Indifferentiability comes with compositional guarantees in the sense that, instead of using a construction C^f based on ideal primitive f in so-called single-stage security game, one can securely use the indifferentiable ideal construction in the game. However, as pointed out by Ristenpart et al. [RSS11] the composition theorem of indifferentiability does not necessarily provide security for multistage security games. Following the terminology in [BF18] we can view multistage security games $\mathcal{G}^{C^f, \mathscr{A}_1^{\mathsf{f}}, \mathscr{A}_2^{\mathsf{f}}, \ldots, \mathscr{A}_n^{\mathsf{f}}}$ as games in which several adversarial instances $\mathscr{A}_1^{\mathsf{f}}, \mathscr{A}_2^{\mathsf{f}}, \ldots, \mathscr{A}_n^{\mathsf{f}}$ with access to the ideal primitive f communicate with each other (through the game) in restricted form. The problem for the indifferentiability simulator Sim is that it usually needs to keep larger state for consistent simulations of the primitive, which it may not be able to pass along if we join the simulator and adversary into a single adversary against the game according to the composition theorem.

Key-dependent message security and security against related-key attacks for AEAD schemes are two prominent examples of multistage games where the general composition results cannot be applied. For key-dependent message security the adversary can see encryptions under message which may depend on the key. In the most simple form the adversary picks a function ϕ for the encryption scheme, one applies $\phi(K)$ to the key to derive the message M, and then encrypts M under this key and returns the ciphertext to the adversary $\mathscr{A}$. In the model with ideal primitives the function ϕ (and other algorithms) may depend on f. As discussed in [FP15] we can then view the attack as a two-stage game in which $\mathscr{A}_1^{\mathsf{f}}$ chooses ϕ^{f} and $\mathscr{A}_2^{\mathsf{f}}$ picks the key, evaluates ϕ^{f}, and returns the ciphertext back to $\mathscr{A}_1$. Now we have two adversarial instances with restricted communication, such that simulating the local instances by a single simulator is

infeasible (and even impossible, as [RSS11] shows).

Barbosa and Farshim [BF18] discuss that one can resurrect compositional guarantees if one can virtually reduce the adversarial instances to a single stage. This is the case if only one of the adversarial instances has access to primitive f, but the other instances can only access C^{f}. Since the game has access to C^{f}, too, one can execute the other instances via the game directly. More generally, they present the following theorem:

Theorem 4.2 ([BF18, Theorem 2]) *Let* (C,f) *be indifferentiable from* R *via some simulator* Sim. *Let* $\mathcal{G}^{\mathsf{C}^{\mathsf{f}},\mathscr{A}_1^{\mathsf{f}},\mathscr{A}_2^{\mathsf{C}^{\mathsf{f}}},\ldots,\mathscr{A}_n^{\mathsf{C}^{\mathsf{f}}}}$ *be a security game for adversary* $(\mathscr{A}_1,\ldots,\mathscr{A}_n)$. *Then there exists an adversary* $(\mathscr{B}_1,\ldots,\mathscr{B}_n)$ *and a distinguisher* $\mathscr{D}$ *such that*

$$\Pr\left[\mathcal{G}^{\mathsf{C}^{\mathsf{f}},\mathscr{A}_1^{\mathsf{f}},\mathscr{A}_2^{\mathsf{C}^{\mathsf{f}}},\ldots,\mathscr{A}_n^{\mathsf{C}^{\mathsf{f}}}}\right] \leq \Pr\left[\mathcal{G}^{\mathsf{R},\mathscr{B}_1^{\mathsf{R}},\mathscr{B}_2^{\mathsf{R}},\ldots,\mathscr{B}_n^{\mathsf{R}}}\right] + \mathbf{Adv}^{\mathit{indiff}}_{\mathsf{C},\mathsf{f},\mathsf{R},\mathsf{Sim}}(\mathscr{D})\,.$$

For key-dependent message security we can thus guarantee composition if we let ϕ not depend on the primitive f but on the construction C^{f} only. Similar restrictions apply to our other applications.

5 Differentiability of Full-State Duplex

In this section we show that one cannot achieve *indifferentiability* of the full-state duplex where the round inputs X_i may affect the entire state. This is in sharp contrast to the *indistinguishability* results in [MRV15, DMV17, DM19, Men22] and shows that one cannot prove the stronger security guarantees of indifferentiability for this duplex version.

Recall that indifferentiability [MRH04] allows to compare ideal objects. More concretely, assume that we have a construction C based on some ideal primitive f, like the duplex construction Dup based on the ideal permutation p. We would like to argue that the construction C^{f} looks like another ideal object R, say, our online random oracle ORO. The distinguisher $\mathscr{D}$ against the construction may also have access to the primitive f (and even the inverse permutation p^{-1} in case of the duplex), running an attack $\mathscr{D}^{\mathsf{C}^{\mathsf{f}},\mathsf{f}}$. This is contrasted with the setting where $\mathscr{D}$ communicates with the ideal object R and a simulator $\mathsf{Sim}^{\mathsf{R}}$ simulating the absent primitive in this setting. Indifferentiability then says that

$$\mathbf{Adv}^{\mathrm{indiff}}_{\mathsf{C},\mathsf{f},\mathsf{R},\mathsf{Sim}}(\mathscr{D}) = \left|\Pr\left[\mathscr{D}^{\mathsf{C}^{\mathsf{f}},\mathsf{f}}=1\right] - \Pr\left[\mathscr{D}^{\mathsf{R},\mathsf{Sim}^{\mathsf{R}}}=1\right]\right|$$

is negligible.

Below we argue that the full-state duplex cannot achieve indifferentiability from an online random oracle. Our result holds even under very mild assumptions, namely when the round-outputs Z_i only consists of a single bit and if $\mathscr{D}$ does not have access to the inverse permutation p^{-1}. To capture the full-state property in our generalised duplex setting we assume that for some round i, given any string $Y_i \in \{0,1\}^{r+c}$, one can (efficiently) determine an X_i such that $Y_i = \mathsf{encode}_i(X_i)$. In the attack below we describe for simplicity how the attack works if $i=1$. It can be easily adapted for other values of $i \geq 1$.

Our distinguisher $\mathscr{D}^{\mathsf{I}_1,\mathsf{I}_2}$, described in Fig. 5, has access to two oracle interfaces, either instantiated with $\mathsf{Dup}^{\mathsf{p}}$ and p in the real world, or with ORO and $\mathsf{Sim}^{\mathsf{ORO}}$ in the ideal world. For the analysis first assume that we are in the real world and I_1 is the duplex construction and I_2 the permutation p. Then the first call to I_1 about $X||W$ in Line 4 computes

$$\mathsf{p}(\ \mathsf{p}(IV \oplus \mathsf{encode}_0(X))\ \oplus \mathsf{encode}_1(W)\ ) = \mathsf{p}(\ Y \oplus \mathsf{encode}_1(W)\ )$$

such that

$$Z = \mathsf{decode}_2(\mathsf{p}(\ Y \oplus \mathsf{encode}_1(W)\ )).$$

$\mathscr{D}^{\mathsf{I}_1,\mathsf{I}_2}$

1 : // let $X \neq X'$, W be arbitrary round inputs
2 : query I_2 about $IV \oplus \mathsf{encode}_0(X)$ to get Y
3 : query I_2 about $IV \oplus \mathsf{encode}_0(X')$ to get Y'
4 : query I_1 about $X||W$ to get Z // via two calls, init and next
5 : compute W' with $\mathsf{encode}_1(W') = \mathsf{encode}_1(W) \oplus Y \oplus Y'$
6 : query I_1 about $X'||W'$ to get Z' // via two calls, init and next
7 : output 1 iff $Z = Z'$

Figure 5: Distinguisher $\mathscr{D}^{\mathsf{I}_1,\mathsf{I}_2}$ against Full-State Duplex.

The second call to I_1 in Line 6 about $X'||W'$ is processed as:

$$\begin{aligned}\mathsf{p}(\ \mathsf{p}(IV \oplus \mathsf{encode}_0(X'))\ \oplus\ \mathsf{encode}_1(W')\ ) &= \mathsf{p}(Y' \oplus\ \mathsf{encode}_1(W) \oplus Y \oplus Y'\ ) \\ &= \mathsf{p}(Y \oplus \mathsf{encode}_1(W))\end{aligned}$$

such that

$$Z' = \mathsf{decode}_2(\mathsf{p}(Y \oplus \mathsf{encode}_1(W))).$$

In this case both calls thus yield the same value $Z = Z'$ such that our distinguisher outputs 1.

Assume now that we are in the ideal setting and I_1 is an (online) random oracle ORO and I_2 the simulator Sim instead. Since the inputs $X||W$ and $X'||W'$ are distinct by assumption about $X \neq X'$, the random oracle returns the same values $Z = Z'$ with probability at most $2^{-|Z|} \leq \frac{1}{2}$. It follows that the distinguisher returns 1 with probability at most $\frac{1}{2}$ in this case. The overall advantage of the distinguisher is therefore at least $\frac{1}{2}$, showing that it successfully tells both cases apart.

While the attack we described exploits the full state absorption that happens just after the first round of the Duplex, it can be easily adapted to exploit full state absorption of the later rounds. When the full state absorption happens at the initial round, the attack can be adapted by using in addition I_2^{-1} to construct X', W' differently. Instead of choosing X' different from X and computing W' such that $\mathsf{encode}_1(W') = \mathsf{encode}_1(W) \oplus \mathsf{I}_2(IV \oplus \mathsf{encode}_0(X)) \oplus \mathsf{I}_2(IV \oplus \mathsf{encode}_0(X'))$, we can choose W' different from W and compute X' such that $\mathsf{encode}_0(X') = IV \oplus \mathsf{I}_2^{-1}(\mathsf{encode}_1(W') \oplus \mathsf{encode}_1(W) \oplus \mathsf{I}_2(IV \oplus \mathsf{encode}_0(X)))$. The attack then proceeds similarly as above with the same advantage.

6 Indifferentiability of Duplex from ORO

In this section we show, via a code-based argument, that the Duplex construction is indifferentiable from ORO.

6.1 Security Statement

As we have seen in the previous section, the security of the Duplex depends on the encoding and decoding functions. In the following we consider the basic case where all round functions are of the form $\mathsf{encode}_i(X) = X \parallel 0^c$ for all rounds such that the leading r bits form the rate part, and to which the input is simply added in the Duplex iterations. Each decoding function $\mathsf{decode}_i(S)$ truncates the capacity component and outputs the rate part of r bits in clear. Call these functions *plain*. In Section B.2 we extend the theorem to more general admissible encoding and decoding functions.

Theorem 6.1 *Let* Dup *be the Duplex construction described in Fig. 1 with plain encoding and decoding functions, composed from a random permutation* p *over b-bit strings, and let c denote its capacity. Further, let* ORO *be the online random oracle with rate* $r = b - c$*. Then there exists a simulator* Sim *such that for every distinguisher* $\mathscr{D}$ *making* q_p *queries to its permutation oracle and* q_c *queries to its construction oracle, where* $q_p, q_c < 2^{c-2}$*, its advantage in differentiating* Dup *from* ORO *is bounded by*

$$\mathbf{Adv}^{indiff}_{\mathsf{Dup},\mathsf{p},\mathsf{ORO},\mathsf{Sim}}(\mathscr{D}) \leq \frac{(q_p + q_c + 1)^2 + 6q_pq_c}{2^c}.$$

The simulator Sim *is described in Fig. 22. Let* $(r \cdot \ell)$ *be a bound on the maximum number of bits queried in an* ORO *session. Then for each of the* q_p *queries it receives,* Sim *runs in time proportional to* ℓ *and makes at most* ℓ *calls to the* ORO*.*

Proof Overview. The proof proceeds via a sequence of games, **G**0–**G**7. We start from the real-world game where the distinguisher $\mathscr{D}$ interacts with a random permutation p and the Duplex construction instantiated with this same random permutation. We then transform this game, step by step, to the ideal-world game, where the distinguisher has access to the online random oracle ORO and a simulator Sim, and argue that any two consecutive games can only be distinguished with (at most) negligible probability. Note that we build our simulator gradually as we progress through the game sequence, and accordingly we define several distinct simulator algorithms in the process. However, the simulator that the indifferentiability theorem refers to, is the one in the final game (**G**7). We also remark that the strategy of this simulator is independent of the distinguisher in question, as required by the standard notion of strong indifferentiability.

Analogous to the proof of indifferentiability for the Sponge [BDPV08], we associate to our simulator a directed graph, that evolves as $\mathscr{D}$ and Dup query the simulator. In this directed graph each node represents a string of size b, and an edge (U, V) represents the mapping $U \to V$. Accordingly, a forward query to the simulator on input U returns V, denoted as $V \leftarrow \mathsf{Sim}(U, +)$, and similarly, an inverse query to the simulator on input V returns U, denoted as $U \leftarrow \mathsf{Sim}(V, -)$. A pictorial example of this graph during **G**1 is shown in Figure 13 on page 30. The simulator maintains this directed graph in a data structure G, where for any node U, $\mathsf{G.in}(U)$ identifies the predecessor of U and $\mathsf{G.out}(U)$ its successor. From this directed graph of input-output mappings, we can derive a second directed graph, which we refer to as the *capacity graph*, by collapsing all nodes sharing the same capacity component into a single node identified by that capacity component. The simulator will ensure that no two adjacent nodes share the same capacity component, and thus all edges in the graph of input-output mappings will transfer to the capacity graph. The reason we care about the capacity graph is that every session evaluated by the Duplex will correspond to a unique path on the capacity graph. More specifically, calls to $\mathsf{Dup}^{\mathsf{Sim}}$ will result in a directed tree, rooted at IV_C, where each session corresponds to a path between the root and a leaf node. Consequently, the simulator must ensure that if the queries it receives from $\mathscr{D}$ correspond to edges on this tree they must be consistent with the online random oracle as otherwise $\mathscr{D}$ would be able to differentiate the real world from the ideal world. To accomplish this, our simulator will maintain its own copy of this tree and only extend it through forward (+) queries for which the queried node is already in the tree and then add a freshly sampled node adjacent to the queried node. The ability to detect when a query extends a path in the tree allows the simulator to recover the path corresponding to the relevant session on the online random oracle and sample the new node consistently with it. In addition the simulator will ensure that queries involving nodes outside the tree and inverse queries do not interfere with the tree structure.

A key requirement of the simulator is that it needs to simulate p from ORO without any knowledge of $\mathscr{D}$'s queries to ORO. A natural approach to prove indifferentiability, which is also the approach we adopt, is to start from p and transform it gradually into the desired simulator. However in the real world p has an interface to the construction which indirectly informs it of the distinguisher's queries to the construction. Consequently, the central challenge in the proof, when transitioning from p to the final

simulator, is in dropping the simulator's interface to the construction. The reason this is problematic is that removing the construction interface can alter the simulator's internal state, as it is no longer affected by the distinguisher's queries to the construction. This was also noted in [CFN+12], pointing out that the original indifferentiability proof of the Sponge [BDPV08] overlooked this step. In turn, [CFN+12] provided its own indifferentiability proof of the Sponge. We note, however, that we handle this transition quite differently from [CFN+12], and, in fact, our proof strategy turns out to be significantly different from that in [CFN+12] or [BDPV08]. Moreover, we are obviously considering a different construction (the Duplex instead of the Sponge) from these prior works, and, because we adopt a code-based proof approach, we also delve deeper into the implementation details of the simulator which allows us to better quantify its resources.

Below is an outline of the game transitions in our proof. Some of the games have a boxed (or unboxed) variant which is a reformulation of the prior game that is mainly intended for better exposition. In this outline we only list the game variant in which the substantive alteration occurs.

Game G0: This is the real-world game where the distinguisher $\mathscr{D}$ is given access to the Duplex $\mathsf{Dup}^{\mathsf{P}}$ via its construction oracle, as well as direct oracle access to the random permutation P.

Game G1: In this game, instead of sampling the random permutation at the start of the game we implement P through lazy-sampling where we store the input-output mappings as a directed graph in the data structure G. As this is merely a syntactic change, **G**0 and **G**1 are perfectly indistinguishable.

Game G2: We now replace P with a simulator $\mathsf{Sim}_{\mathsf{fc}}$ that samples nodes in the graph according to a different distribution. Specifically, it ensures that all sampled nodes have a *fresh capacity component* and samples the rate component uniformly at random. To accomplish this, $\mathsf{Sim}_{\mathsf{fc}}$ maintains a list C of the capacity component of every node that is added to G and IV_C, and samples the capacity component uniformly from the set $\{0,1\}^c \setminus \mathsf{C}$.

Note that distinct nodes in G can still share the same capacity component, as queried nodes are also added to G and there is no restriction on which nodes can be queried. However this modification ensures that no two adjacent nodes share the same capacity component and consequently the capacity graph contains no loops. Moreover since all sampled nodes have a fresh capacity component, there will also be no cycles in the capacity graph.

We also introduce some labeling that leaves the game functionally unaltered, but which will become handy later on. Everytime a node is added to G, we mark it with '\$' or '$*$' to indicate whether the node was added via sampling or as a query input, respectively. Note that, due to the fresh-capacity sampling, for each capacity component stored in C there exists at most one node with that capacity component and the mark \$. We call such a node the *representative* of that capacity component and we additionally store it alongside each entry in C (if it exists).

Game G3: We now adapt the simulator's behaviour to take into account the interface from which it receives a query. We do so by providing two separate interfaces: $\mathsf{Sim}_{\mathsf{p}}$ which handles direct primitive queries from the distinguisher, and $\mathsf{Sim}_{\mathsf{c}}$ which handles internal calls from the construction. We also introduce new labelling and in this game the interface-dependent behaviour of the simulator differs only in how this labelling is applied. Every node that is added to G is additionally labelled with 'd' or 't', where 't' represents nodes on a computation path of the duplex (starting with "root" IV and potentially branching of from an earlier computation path, thus forming a tree of paths), and 'd' marks disconnected nodes through direct primitive queries. This labelling extends to the corresponding entries in C and is applied as follows. Nodes added by $\mathsf{Sim}_{\mathsf{c}}$ are always labelled with 't', whereas nodes added by $\mathsf{Sim}_{\mathsf{p}}$ are labelled with 'd' except when (1) there already exists another

node in G with the same capacity component labelled 't', or (2) the node's predecessor is labelled 't'. In addition the node IV is labelled 't' before any query is made.

Through this labelling we have effectively subdivided C into two disjoint sets $\mathsf{C_d}$ and $\mathsf{C_t}$. Thus, the capacity graph can now be subdivided into two subgraphs whose nodes correspond to these two sets. Of particular note is the capacity subgraph corresponding to $\mathsf{C_t}$, which represents Duplex evaluations carried out by the distinguisher either through its construction oracle or computed locally via its primitive oracle. Specifically, this capacity subgraph is a directed tree (hence the label 't') with IV_C as its root, and every path from the root to any other node corresponds to a unique sequence of Duplex queries. While the two subgraphs can be connected in general, edges can only be directed from $\mathsf{C_d}$ to $\mathsf{C_t}$ and not vice versa.

At this point we have a two-part simulator which imposes enough structure on the graph to allow it to discern queries that relate to a Duplex session from ones that do not. In the remaining game transitions we further separate $\mathsf{Sim_p}$ from $\mathsf{Sim_c}$, and we gradually transform $\mathsf{Dup}^{\mathsf{Sim_c}}$ into the ORO and $\mathsf{Sim_p}$ into our final simulator.

Game G4: In this game we introduce an ORO instance, but we only make it accessible to the simulator and not the distinguisher. That is, the construction oracle continues to be the Duplex construction, which makes primitive queries to the simulator, which in turn can now query the ORO. The simulator will now use the ORO to sample the rate component of nodes labelled ($,t) rather than sampling them uniformly at random. The sampling of the capacity components remains unchanged.

Now, every node labelled ($,t) corresponds to a distinct node in the tree within the capacity graph. In turn every node in the tree identifies a distinct sequence of Duplex queries determined by the path from the root to that node. Thus, for every new node labelled ($,t) the simulator will determine this sequence of Duplex queries, submit them to the ORO, and assign its final output as the rate component of that node. Intuitively, we have adjusted the simulator to sample the nodes in the graph so that the output of the Duplex matches that of the ORO. In addition, we have not altered the output distribution of the simulator because every node identifies a distinct sequence of Duplex queries, and accordingly the corresponding ORO output is guaranteed to be uniformly distributed.

Game G5: In the previous game we have aligned the output of the construction oracle with the ORO. Looking ahead, we ultimately want that the current combination of the Duplex accessing $\mathsf{Sim_p}$ which in turn accesses the ORO be replaced with the ORO directly. Although the construction oracle is already reproducing the output of the ORO, we are not yet ready to make this swap. The main reason is that $\mathsf{Sim_c}$ and $\mathsf{Sim_p}$ share memory, and thus the presence of $\mathsf{Sim_c}$ heavily affects the operation of $\mathsf{Sim_p}$. In this and the next game hop, we work towards lifting the influence that $\mathsf{Sim_c}$ has on $\mathsf{Sim_p}$ in preparation for that last step.

In this game we make four main changes. The first one is that we remove any internal calls that $\mathsf{Sim_p}$ makes to $\mathsf{Sim_c}$, so that they are separate algorithms, although they still share memory. The second change is to let $\mathsf{Sim_p}$ keep track of "its own copy" of $\mathsf{C_t}$. Technically, we introduce another label p to identify the nodes marked 't' that are sampled by $\mathsf{Sim_p}$. We store the capacity components of these nodes in a new data structure $\mathsf{L_p}$, initialised to $\{IV_C\}$, and it clearly follows that $\mathsf{L_p} \subseteq \mathsf{C_t}$. Moreover the set $\mathsf{C_t} \setminus \mathsf{L_p}$ identifies the capacity components of the nodes sampled by $\mathsf{Sim_c}$. The third change is that we replace the lines which test for membership in $\mathsf{C_t}$ with a membership test in $\mathsf{L_p}$. This change makes partial progress in making $\mathsf{Sim_p}$ rely on the data that it generates itself. The fourth, and perhaps the most important part of this game hop, is to limit the possibility of $\mathsf{Sim_p}$ outputting a node with a capacity component contained in $\mathsf{L_p} \subseteq \mathsf{C_t}$. The most direct way in which the distinguisher can cause this, is by reproducing the internal Duplex calls corresponding to prior

construction queries. To avoid this, we change $\mathsf{Sim_p}$ so that it overwrites these capacity components, previously set by $\mathsf{Sim_c}$, by resampling them anew. Since construction queries do not reveal the capacity components of the nodes that they sample, this resampling will ensure that they remain hidden from the distinguisher with high probability. As a result of this modification, the distinguisher will then be unlikely to query these values to $\mathsf{Sim_p}$. The technical challenge of this game hop lies in showing that this resampling leaves the distribution of $\mathsf{Sim_p}$'s outputs largely unaffected.

Game G6: We now adjust the sampling procedures in $\mathsf{Sim_p}$ so that it is not affected by data structures which $\mathsf{Sim_c}$ writes to. More specifically, it will now sample capacity components from $\{0,1\}^c \setminus \mathsf{C_d} \cup \mathsf{L_p}$ instead of $\{0,1\}^c \setminus \mathsf{C}$. Note that we now sample from a slightly larger set, and by using the fundamental lemma of game playing, it can be shown that this modification is only detectable if at any point the sampled value lies in their set difference, i.e., $\mathsf{C_t} \setminus \mathsf{L_p}$. This happens only with small probability.

Game G7: We are finally ready to replace the combination of the Duplex construction, $\mathsf{Sim_c}$, and the ORO, directly with the ORO itself. We can make this change because at this point $\mathsf{Sim_c}$ is for the most part acting as a relay forwarding calls to the ORO and forwarding back its replies. The only side effect is that $\mathsf{Sim_c}$ adds nodes into the data structure G when forwarding queries between the Duplex and the ORO. When we replace the Duplex with the ORO these values will no longer be added to G, which is still used by $\mathsf{Sim_p}$. However this would only affect the operation of $\mathsf{Sim_p}$ if it is ever queried on the nodes added by $\mathsf{Sim_c}$. Once again, these are the nodes whose capacity components are contained in $\mathsf{C_t} \setminus \mathsf{L_p}$, and most importantly, are hidden from the distinguisher. Thus this swap will only be noticeable with negligible probability and $\mathsf{Sim_p}$ now serves as the final indifferentiability simulator.

The full details of the proof can be found in Appendix B.1.

7 Revisiting the Security of SpongeWrap

Having shown the Duplex to be indifferentiable from the ORO we now put this result to use by analysing the security of constructions based on the Duplex. Specifically, we can now leverage the indifferentiability of the Duplex to translate security proofs in the ORO model to security proofs in the random permutation model. One reason why this is advantageous is that we generally expect security proofs in the ORO model to be simpler and more intuitive than ones in the random permutation model. The target of our analysis will be SpongeWrap [BDPV12], which is arguably the most direct approach for constructing an AEAD scheme from the Duplex, and it also served as the basis for several other Duplex-based AEAD constructions—such as NIST's Lightweight Cryptography standard Ascon [DEMS16]. Towards proving stronger security for SpongeWrap, one avenue would be to prove it indifferentiable from one of the ideal AEAD primitives put forth by Barbosa and Farshim [BF18]. As shown therein, this would automatically imply that SpongeWrap retains security under related-key attacks and key-dependent messages, offers misuse resistance, and is suitable for message franking applications. Unfortunately, for reasons that we explain in Section 7.2, SpongeWrap, and generally most Duplex-based AEAD schemes, are unable to meet the idealised AEAD notions put forth in [BF18]. Nevertheless, with the exception of misuse-resistance, all security properties implied by ideal AEAD are still perfectly within reach. In the rest of this section, we show, in the ORO model, that SpongeWrap benefits from these advanced security properties and then use Theorem 6.1 to translate these results to the random permutation model.

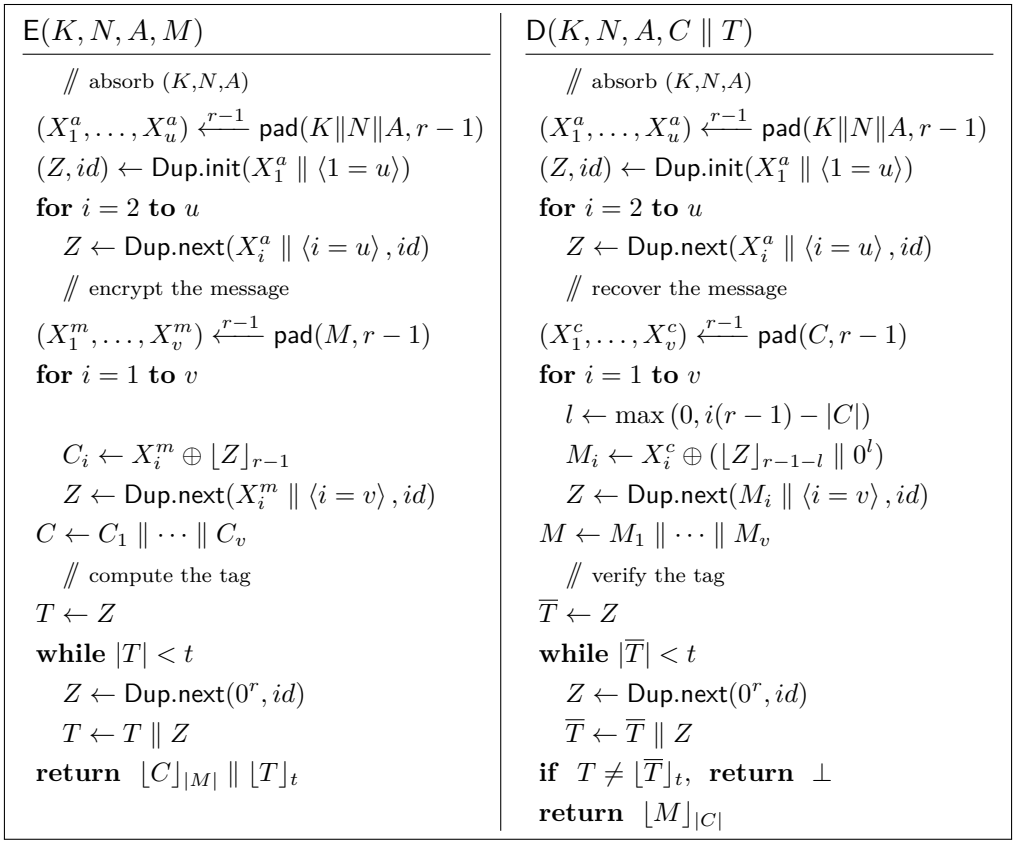

Figure 6: Pseudocode description of SpongeWrap (nSW[Dup]) expressed as a function of the Duplex.

7.1 A Nonce-Based Variant of SpongeWrap

SpongeWrap was introduced in [BDPV12] as a one-pass AEAD scheme based on the Duplex where its security was argued based on the indifferentiability of the Sponge. As explained in Section 2.2, this approach has some drawbacks. Most notably, it results in an effective security bound on the order of $\mathcal{O}(N^4)$ when the total query count entails N permutation calls. In contrast, our analysis will result in a bound on the order of $\mathcal{O}(N^2)$. This quantitative improvement is a direct consequence of basing the security analysis in the online random oracle model as opposed to the random oracle model.

A pseudocode description of SpongeWrap (nSW[Dup]), expressed as a function of the Duplex is shown in Fig. 6, together with a pictorial description of the encryption and decryption procedures in Fig. 7 and Fig. 8. It is assumed that $\mathsf{pad}(\cdot, r-1)$ only appends bits at the end of its first input such that it constitutes an injective mapping to strings of size $a(r-1)$ for some integer $a \geq 1$. In every Duplex call, a bit is appended to each input in order to delineate the boundary between the associated data and the message and between the ciphertext and the tag. Specifically, this bit is always set to zero except for the last block of the associated data and the last block of the message. This variant of SpongeWrap differs from its original formulation in three ways. It explicitly exposes a fixed-size nonce, whereas in the original version, the associated data was required to be non-empty and non-repeating, thereby filling the role of a variable-length nonce. Thus our adaptation of SpongeWrap makes it compliant with the standard nonce-based AEAD syntax. Secondly, it does away with Sponge-compliant padding at every permutation call since it is not needed for the indifferentiability of the Duplex. Finally, the inclusion of the encoding and decoding functions in the Duplex renders our variant slightly more general as it allows for some variation

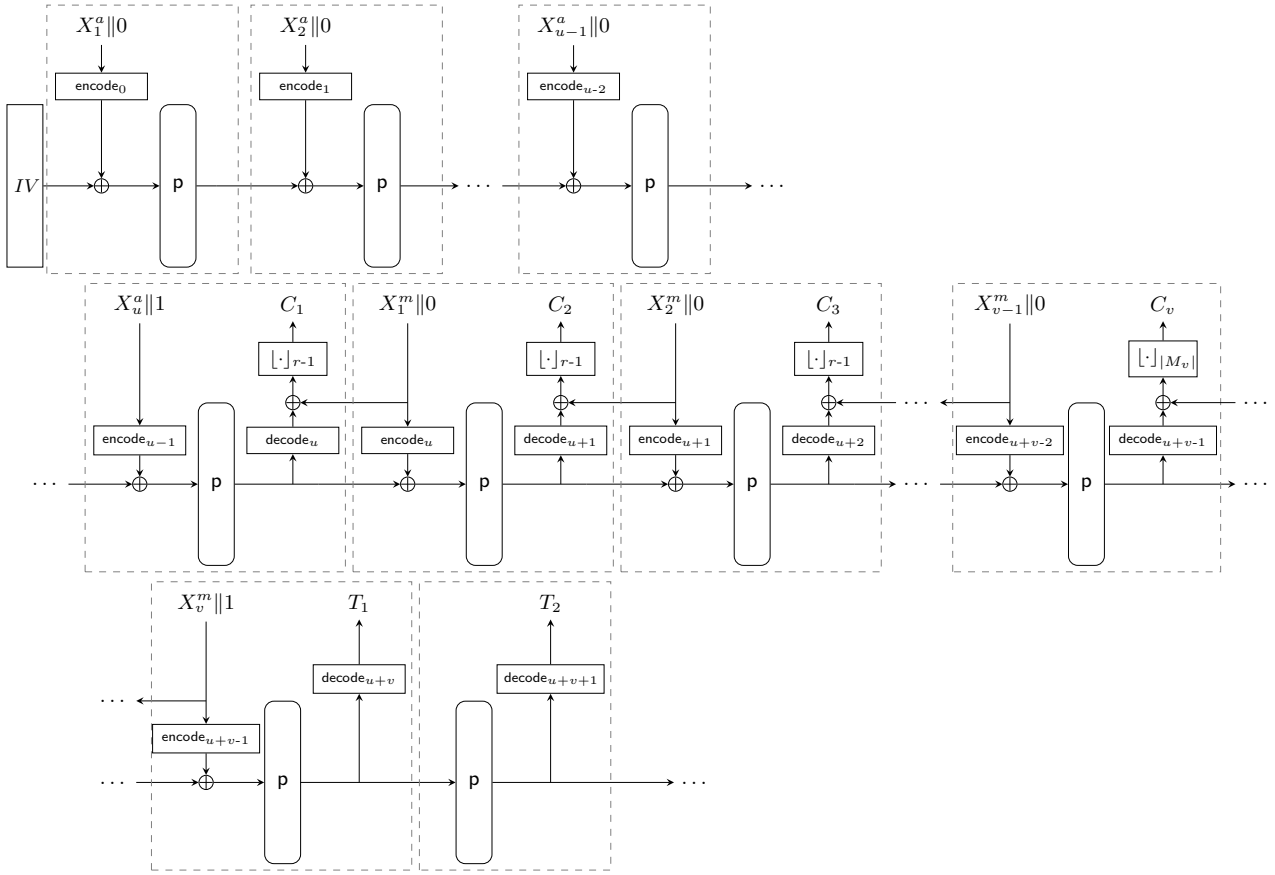

Figure 7: Encryption of SpongeWrap (nSW[Dup]). The key K, nonce N and associated data A gets padded into u blocks as $(X_1^a, \ldots, X_u^a) \xleftarrow{r-1} \mathsf{pad}(K\|N\|A, r-1)$ and the plaintext gets padded into v blocks as $(X_1^m, \ldots, X_v^m) \xleftarrow{r-1} \mathsf{pad}(M, r-1)$ with $u, v \geq 1$.

in how SpongeWrap is realised in practice. We specify some additional notation specific to SpongeWrap nSW[Dup] at the beginning of Appendix C.

We analyse SpongeWrap with respect to KDM, RKA and context commitment security. Being already a fairly efficient scheme (it is the basis for several CAESAR candidates) it is noteworthy that nSW[Dup] achieves KDM, RKA and context commitment security without any additional overhead or assumptions. In comparison, other AEAD constructions that achieve KDM-AEAD security are the ideal AEAD constructions by Barbosa and Farshim [BF18] and the generic transformation by Bellare and Keelveedhi [BK11]. The fastest construction from [BF18] is a *three-pass* scheme, where each pass can be implemented via a Sponge evaluation, and the construction from [BK11] augments an AEAD scheme with a pre-computation step requiring a hash evaluation and re-keying the AEAD scheme for every encryption/decryption call. For RKA-AEAD security, the construction from Barbosa and Farshim [BF18] and the construction N* from Faust et al. [FKOS22] are both three-pass schemes. Finally, for context commitment, the different existing constructions (cf. [CR22, Table 2]) require an additional function evaluation that could be a hash, a MAC or a PRF.

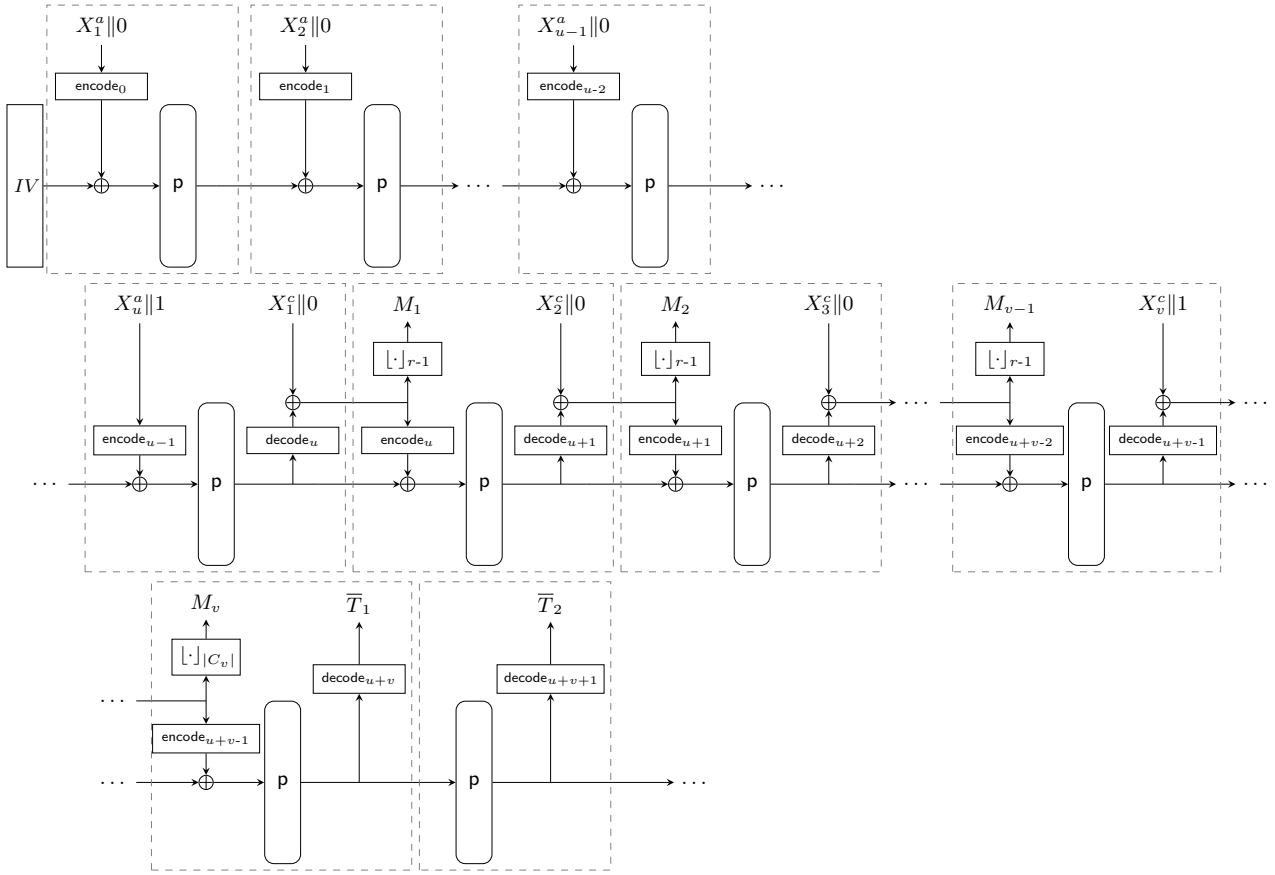

Figure 8: Decryption of SpongeWrap (nSW[Dup]). The key K, nonce N and associated data A gets padded into u blocks as $(X_1^a, \ldots, X_u^a) \xleftarrow{r-1} \mathsf{pad}(K\|N\|A, r-1)$ and the ciphertext gets padded into v blocks as $(X_1^c, \ldots, X_v^c) \xleftarrow{r-1} \mathsf{pad}(C, r-1)$ with $u, v \geq 1$.

7.2 SpongeWrap is Differentiable From Ideal AEAD

In [BF18] Barbosa and Farshim put forth two ideal AEAD functionalities corresponding to the online and offline syntaxes. In broad terms, the offline functionality corresponds to a tweakable length-expanding random injection, and the online (encryption) functionality processes each call through a tweakable random injection where its output additionally yields a state that is fed back into the tweak of the next call, but its input and output are expanded to also carry a state. Being a random injection with sufficient expansion, it directly follows that the offline ideal AEAD functionality is misuse-resistant [RS06]. On the other hand, it is well known that a single-pass scheme, like SpongeWrap, cannot be misuse-resistant. Accordingly, indifferentiability with respect to the offline functionality is unattainable. As for the online functionality, it is not misuse-resistant when considering the aggregate ciphertext over multiple calls. However, it still guarantees that the first encryption call of two distinct messages under the same nonce-key pair will yield totally uncorrelated ciphertexts. Clearly, SpongeWrap does not meet this requirement since a repeated nonce-key pair will always xor the message with the same string.

7.3 KDM-AEAD Security

Key-dependent-message security in the context of symmetric encryption was first studied by Black, Rogaway, and Shrimpton in [BRS03]. Subsequently, Bellare and Keelveedhi [BK11] extended this security

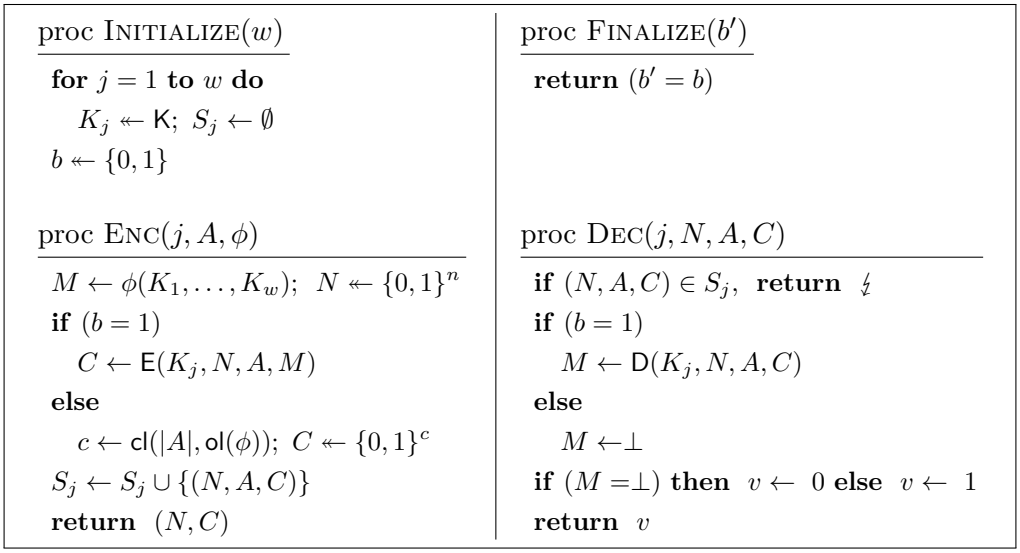

Figure 9: The KDAE game for defining AEAD security in the presence of key-dependent messages.

notion to nonce-based AEAD schemes. By means of generic attacks, they showed that key-dependent-data security is only possible when nonces are sampled at random and the header is independent of the key. Accordingly, only one of the four security definitions that they considered is satisfiable—reproduced here as Definition 7.1 and its corresponding game is described in Fig. 9. In order to constrain the nonce to be sampled uniformly at random, it is sampled by the ENC oracle and then returned to the adversary rather than being chosen directly by the adversary. Also note that the DEC oracle returns either the $\lightning$ symbol to indicate a prohibited input, or a bit indicating whether decryption succeeded or failed. As noted in [BK11], this latter choice is without loss of generality.

Definition 7.1 (KDM-AEAD Security) *Let* $\mathsf{SE} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be an AEAD scheme with key size* k *and the* KDAE *game be as defined in Fig. 9. Further let* $\mathcal{A}$ *be any adversary whose queries are such that* $\phi : (\{0,1\}^k)^w \to \{0,1\}^{\mathsf{ol}(\phi)}$*, where the output length* $\mathsf{ol}(\phi)$ *of* ϕ *is constant and* w *is its input to* INITIALIZE*. Then its corresponding* KDAE *advantage is given by:*

$$\mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{SE}}(\mathcal{A}) = 2 \cdot \Pr\left[\mathsf{KDAE}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1\,.$$

If we work with idealised schemes such as the Duplex with the permutation p then the message-derivation function ϕ, chosen by the adversary in each encryption query, may depend on the idealised primitives as well. Formally, the description of the function ϕ may entail oracle gates, where we write $\phi^{(\cdot)}$ to denote such functions. However, as noted in [RSS11], if we allow ϕ to call p directly, then we demonstrably cannot apply the composition theorem for indifferentiability anymore. On the other hand, as pointed out by Barbosa and Farshim [BF18], compositional guarantees luckily still hold if ϕ only makes calls to the construction instead of direct calls to the primitive. See also Section 4.2 for a more comprehensive discussion. In particular, if we transfer the compositional result in [BF18] to our setting, it suffices to consider KDM security with respect to the online random oracle (where each function ϕ may call ORO). Then the security of SpongeWrap ($\mathsf{nSW}[\mathsf{Dup}]$) in the p-model with respect to key-derivation functions of the form $\phi^{\mathsf{Dup}[\mathsf{p}]}$ follows, where the difference in the advantages between the two settings is bounded by the indifferentiability advantage $\mathbf{Adv}^{\mathrm{indiff}}_{\mathsf{Dup},\mathsf{p},\mathsf{ORO},\mathsf{Sim}}(\mathscr{D})$ for the distinguisher $\mathscr{D}$ consisting of the security game running $\mathcal{A}^{\mathsf{Sim}}$. The KDM-AEAD security of SpongeWrap in the online random oracle model is stated formally in Theorem 7.2 below, and its proof can be found in Appendix C.1.

Theorem 7.2 (SpongeWrap is KDM-AEAD Secure in the ORO-model) *Let* $\mathsf{nSW}[\mathsf{ORO}] = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be the AEAD scheme described in Fig. 6 in the online random oracle model having key size* k*, nonce size* n*, and tag size* t*. Further, let* $\mathcal{A}$ *be a* KDAE *adversary initialising* w *keys and making* q_e *encryption queries,* q_d *decryption queries, and* q_o *queries to the* ORO*. Let* q_ϕ *denote the number of queries to the* ORO *that all key derivation functions* ϕ *can make in total. Then for any such adversary querying key-derivation functions of the form* ϕ^{ORO}*, its corresponding* KDAE *advantage is bounded by:*

$$\mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW}[\mathsf{ORO}]}(\mathcal{A}) \leq \frac{q_e^2 + q_e q_\phi}{2^n} + \frac{w q_o + w^2}{2^k} + \frac{2q_d}{2^t}\,.$$

Translating the KDM Bound from the ORO-model to the p-model. We now apply the composition theorem [BF18, Theorem 1] to show the security of SpongeWrap ($\mathsf{nSW}[\mathsf{Dup}]$). By combining the result in Theorem 7.2 with the composition theorem re-stated in Theorem 4.2, we obtain the following result, whose proof can be found in Appendix C.2.

Theorem 7.3 (SpongeWrap is KDM-AEAD Secure in the p-model) *Let* $\mathsf{nSW}[\mathsf{Dup}] = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be the AEAD scheme described in Fig. 6 in the* p*-model having key size* k*, nonce size* n*, and tag size* t*. Further, let* $\mathcal{A}$ *be a* KDAE *adversary initialising* w *keys and making* q_e *encryption queries,* q_d *decryption queries,* q_p *primitive queries. Let* q_ϕ *denote the maximum number of construction queries that all key derivation functions* ϕ *can make in total. Then for any such adversary querying key-derivation functions of the form* ϕ^{Dup}*, its corresponding* KDAE *advantage is bounded by:*

$$\mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW}[\mathsf{Dup}]}(\mathcal{A}) \leq \frac{q_e^2 + q_e q_\phi}{2^n} + \frac{w\ell q_p + w^2}{2^k} + \frac{2q_d}{2^t} + \frac{(q_p + \ell(q_e + q_d) + q_\phi + 1)^2}{2^c} + \frac{6q_p(\ell(q_e + q_d) + q_\phi)}{2^c}\,.$$

where $\ell \cdot r$ *is a bound on the maximum input bit length made to the duplex construction during an encryption/decryption query (including the ones inside the key-derivation functions* ϕ*). We further require* q_p *and* $\ell(q_e + q_d) + q_\phi$ *to be strictly less than* 2^{c-2}*.*

7.4 RKA-AEAD Security

Related-key attacks were first introduced as a cryptanalysis tool for block cipher [Bih94, Knu92]. Motivated by real attacks, related-key security was then formally studied by Bellare and Kohno [BK03] for pseudorandom permutations and functions. From then, the notion was extended to other primitives such as encryption schemes [AHI11] and MACs [BR14]. For authenticated encryption, Lu et al. [LLJ15] defined RKA security for probabilistic schemes as a combination of two security notions: indistinguishability security against related-key attacks (IND-RKA) and integrity security against related-key attacks (INT-RKA). Later, Faust et al. [FKOS22] defined an all-in-one RKA security notion for nonce-based AEAD schemes, which we reproduce in Definition 7.4 and denote RKA-AEAD security. This security notion implies the classical all-in-one AEAD security notion from [RS06] by considering the identity function as the only related-key-deriving (RKD) function allowed.

Definition 7.4 (RKA-AEAD Security) *Let* $\mathsf{SE} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be an AEAD scheme,* $\Phi \subset \mathsf{Func}(\mathsf{K}, \mathsf{K})$ *and the* RKAE *game be defined as in Fig. 10. Further, let* $\mathcal{A}$ *be any adversary whose queries are such that* $\varphi \in \Phi$ *and* (φ, N) *never repeats across their encryption queries. Then its corresponding* RKAE *advantage is given by:*

$$\mathbf{Adv}^{\mathsf{rkae}}_{\mathsf{SE}}(\mathcal{A}, \Phi) = 2\Pr\left[\mathsf{RKAE}^{\mathcal{A}} \Rightarrow \mathsf{true}\right] - 1\,.$$

proc INITIALIZE	proc FINALIZE(b')
$K \twoheadleftarrow \mathsf{K};\ S \leftarrow \emptyset$	**return** $(b' = b)$
$b \twoheadleftarrow \{0,1\}$	
proc ENC(φ, N, A, M)	proc DEC(φ, N, A, C)
if $(b = 1)$	**if** $(\varphi, N, A, C) \in S$, **return** $\lightning$
$C \leftarrow \mathsf{E}(\varphi(K), N, A, M)$	**if** $(b = 1)$
else	$M \leftarrow \mathsf{D}(\varphi(K), N, A, C)$
$c \leftarrow \mathsf{cl}(\lvert A\rvert, \lvert M\rvert);\ C \twoheadleftarrow \{0,1\}^c$	**else**
$S \leftarrow S \cup \{(\varphi, N, A, C)\}$	$M \leftarrow \perp$
return C	**return** M

Figure 10: The RKAE game for defining AEAD security under related keys.

In the previous definition, the RKAE advantage depends on a set Φ of related-key deriving functions. This restriction is necessary, as Bellare and Kohno [BK03] showed that RKA security is only achievable for restricted sets of related-key deriving functions. For the ideal cipher to be RKA secure, they showed that output-unpredictability (Definition 7.5) and collision resistance (Definition 7.6) are sufficient conditions on the set of related-key deriving functions.

Definition 7.5 (Output-unpredictability for Φ.) *Let* K *be a set of keys and* $\Phi \subset \mathsf{Func}(\mathsf{K}, \mathsf{K})$*. Let* r*,* r' *be positive integers. Then*

$$\boldsymbol{InSec}^{up}_{\Phi}(r, r') = \max_{F \subseteq \Phi, X \subseteq \mathsf{K}, |F| \leq r, |X| \leq r'} \Pr_{K \twoheadleftarrow \mathsf{K}}[\,\{\varphi(K) : \varphi \in F\} \cap X \neq \emptyset]$$

is defined as the (r*,*r'*)-output-unpredictability of* Φ*.*

In the previous definition, the maximum is over all multisets F of at most r elements of Φ.

Definition 7.6 (Collision resistance for Φ.) *Let* K *be a set of keys and* $\Phi \subset \mathsf{Func}(\mathsf{K}, \mathsf{K})$*. Let* r *be a positive integer. Then*

$$\boldsymbol{InSec}^{cr}_{\Phi}(r) = \max_{F \subseteq \Phi, |F| \leq r} \Pr_{K \twoheadleftarrow \mathsf{K}}[\,|\{\varphi(K) : \varphi \in F\}| < |F|]$$

is defined as the r*-collision resistance of* Φ*.*

Similarly as in the KDM case, when working with idealised schemes such as the Duplex with an ideal permutation p, the key-derivation functions φ, chosen by the adversary in each encryption/decryption query, can call the idealised primitive. We again write $\varphi^{(\cdot)}$ to denote such functions with access to an oracle $(\cdot)$ and $\Phi^{(\cdot)}$ to denote a set of them. To be able to leverage the indifferentiability of the Duplex from ORO and apply the composition theorem, we restrict our analysis to RKD functions that can make only calls to the construction and cannot make calls to the primitive. As shown by Albrecht et al. [AFPW11], new restrictions must be added on the set of related key-derivation functions when the key derivation function is dependent on the cipher. In the case of RKA security of the ideal cipher, they introduced the notion of oracle-independence for a set of RKD functions. We adapt this notion to the duplex construction in Definition 7.7 and call it query independence.

Definition 7.7 (Query independence for Φ^{Dup}.) *Let* K *be a set of keys and* Φ^{Dup} *be a set of related-key-deriving functions on the key space* K*. Then* $\boldsymbol{InSec}^{qi}_{\Phi^{\mathsf{Dup}}}(r, r')$ *is defined as the maximum probability that for any multiset* F *of at most* r *elements of* Φ^{Dup} *and making at most* r' *queries to the duplex construction* Dup*, when running successively all elements* φ^{Dup} *of* F *over a random input key* K*, one of the keys derived by a* φ^{Dup} *hits one of the paths queried to* Dup *by another or the same* φ^{Dup}*, i.e.,* $\boldsymbol{InSec}^{qi}_{\Phi^{\mathsf{Dup}}}(r, r') =$

$$\max_{F \subseteq \Phi^{\mathsf{Dup}}, |F| \leq r, \sum_{\varphi \in F} \left|\mathrm{Qry}[\varphi^{\mathsf{Dup}}(K)]\right| \leq r'} \Pr_{K \leftarrow \mathsf{K}}[\exists \varphi_1^{\mathsf{Dup}}, \varphi_2^{\mathsf{Dup}} \in F, \exists P \in \mathrm{Qry}[\varphi_2^{\mathsf{Dup}}(K)],\ \varphi_1^{\mathsf{Dup}}(K) = [P]^K],$$

where $\mathrm{Qry}[\varphi^{\mathsf{Dup}}(K)]$ *denotes the set of queries placed to* Dup *by* φ^{Dup} *when run on input* K*. We call* $\boldsymbol{InSec}^{qi}_{\Phi^{\mathsf{Dup}}}(r, r')$ *the* (r, r')*-query independence of* Φ^{Dup}*.*

In the previous definition, the maximum is over all multisets F of at most r elements of Φ^{Dup}. Note that any set Φ of oracle-free related-key-deriving functions is query independent.

Similarly as in previous works analysing the RKA security of a scheme with RKD functions dependent on the cipher, when restricting the set of RKD functions to be output-unpredictable, collision resistant and query independent, we can show that SpongeWrap is RKA-AEAD secure. The RKA-AEAD security of SpongeWrap in the online random oracle model is stated formally in Theorem 7.8 below, and its proof can be found in Appendix C.3.

Theorem 7.8 (SpongeWrap is RKA-AEAD Secure in the ORO-model) *Let* $\mathsf{nSW}[\mathsf{ORO}] = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be the AEAD scheme described in Fig. 6 in the online random oracle model having tag size* t *and* Φ^{ORO} *be a set of related-key-deriving functions on the key space* K*. Further, let* $\mathcal{A}$ *be a* RKAE *adversary making* q_e *encryption queries,* q_d *decryption queries, and* q_o *queries to the* ORO*. Let* q_φ *denote the number of queries to the* ORO *that all key derivation functions* φ *can make in total. Then for any such adversary querying key-derivation functions of the form* $\varphi^{\mathsf{ORO}} \in \Phi^{\mathsf{ORO}}$ *and never repeating a pair* $(\varphi^{\mathsf{ORO}}, N)$ *across their encryption queries, its corresponding* RKAE *advantage is bounded by:*

$$\mathbf{Adv}^{\mathsf{rkae}}_{\mathsf{nSW}[\mathsf{ORO}]}\left(\mathcal{A}, \Phi^{\mathsf{ORO}}\right) \leq \boldsymbol{InSec}^{up}_{\Phi^{\mathsf{ORO}}}(q_e + q_d, q_o) + \boldsymbol{InSec}^{cr}_{\Phi^{\mathsf{ORO}}}(q_e + q_d) + \boldsymbol{InSec}^{qi}_{\Phi^{\mathsf{ORO}}}(q_e, q_\varphi) + \frac{2q_d}{2^t}\,.$$

Similarly as done in Theorem 7.3 for the KDM case, combining the composition Theorem 4.2 with the result from Theorem 7.8 yields the RKA-AEAD security of SpongeWrap in the p-model, where the related-key-deriving functions have only access to the Duplex construction.

7.5 CMT-AEAD Security

Following previous works on commitment of authenticated encryption schemes [FOR17, GLR17, DGRW18, ADG+22], Bellare and Hoang [BH22] proposed multiple security notions for committing AEAD. The strongest notion for tidy AEAD schemes, denoted CMT-3 originally, and called context-committing here, is reproduced in Definition 7.9 with its associated game in Fig. 11. It says that the adversary wins if it creates distinct inputs to the AEAD scheme resulting in the same ciphertext. Since the adversary has full control over the input to the AEAD scheme, including the key, the security game only consists of the FINALIZE procedure checking the adversary's choice for a collision.

Definition 7.9 (CMT-AEAD Security) *Let* $\mathsf{SE} = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be an AEAD scheme and the* CMT *game be as defined in Fig. 11. Then the* CMT *advantage of an adversary* $\mathcal{A}$ *is given by:*

$$\mathbf{Adv}^{\mathsf{cmt}}_{\mathsf{SE}}(\mathcal{A}) = \Pr\left[\mathsf{CMT}^{\mathcal{A}} \Rightarrow \mathsf{true}\right].$$

proc FINALIZE$((K_1, N_1, A_1, M_1), (K_2, N_2, A_2, M_2))$
return $(((K_1, N_1, A_1) \neq (K_2, N_2, A_2)) \wedge (\mathsf{E}(K_1, N_1, A_1, M_1) = \mathsf{E}(K_2, N_2, A_2, M_2)))$

Figure 11: The CMT game defining context-committing AEAD security.

CMT-AEAD security notably implies r-BIND security [GLR17], making CMT-AEAD secure schemes suitable for message franking applications. The CMT-AEAD security of SpongeWrap in the online random oracle model stated in Theorem 7.10, follows immediately from the collision resistance of ORO, assuming a sufficiently long tag. The proof of the following theorem can be found in Appendix C.4.

Theorem 7.10 (SpongeWrap is CMT-AEAD Secure in the ORO-model) *Let* $\mathsf{nSW}[\mathsf{ORO}] = (\mathsf{K}, \mathsf{E}, \mathsf{D})$ *be the AEAD scheme described in Fig. 6 in the online random oracle model and having tag size* t*. Further, let* $\mathcal{A}$ *be a* CMT *adversary making* q_o *oracle queries to the* ORO*. Then for any such adversary, its corresponding* CMT *advantage is bounded by:*

$$\mathbf{Adv}^{\mathsf{cmt}}_{\mathsf{nSW}[\mathsf{ORO}]}(\mathcal{A}) \leq \frac{q_o^2 + 2}{2^{t+1}}\,.$$

As CMT-AEAD security is modelled as a single-stage game, simply combining the composition Theorem 4.2 with the result from Theorem 7.10 yields the CMT-AEAD security of SpongeWrap in the p-model.

Acknowledgments

We thank Aishwarya Thiruvengadam for her input during earlier stages of this work. We are grateful to the anonymous ASIACRYPT 2023 reviewers for their constructive comments. This research was supported by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

[ADG+22] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to abuse and fix authenticated encryption without key commitment. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 3291–3308, Boston, MA, USA, August 10–12, 2022. USENIX Association. (Cited on pages 4 and 23.)

[AFPW11] Martin R. Albrecht, Pooya Farshim, Kenneth G. Paterson, and Gaven J. Watson. On cipher-dependent related-key attacks in the ideal-cipher model. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 128–145, Lyngby, Denmark, February 13–16, 2011. Springer Berlin Heidelberg, Germany. (Cited on page 22.)

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In *Innovations in Computer Science - ICS 2011*, pages 45–60, January 2011. (Cited on page 21.)

[AJN14] Jean-Philippe Aumasson, Philipp Jovanovic, and Samuel Neves. NORX: Parallel and scalable AEAD. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *ESORICS 2014: 19th European Symposium on Research in Computer Security, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 19–36, Wroclaw, Poland, September 7–11, 2014. Springer, Cham, Switzerland. (Cited on page 9.)

[BDP+16] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Ketje v2. *Submission to the CAESAR Competition*, 2016. `https://keccak.team/files/Ketjev2-doc2.0.pdf`. (Cited on pages 3 and 7.)

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the Sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197, Istanbul, Turkey, April 13–17, 2008. Springer Berlin Heidelberg, Germany. (Cited on pages 1, 13, and 14.)

[BDPV12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337, Toronto, Ontario, Canada, August 11–12, 2012. Springer Berlin Heidelberg, Germany. (Cited on pages 2, 3, 6, 7, 8, 16, and 17.)

[BF18] Manuel Barbosa and Pooya Farshim. Indifferentiable authenticated encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 187–220, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. (Cited on pages 4, 7, 10, 11, 16, 18, 19, 20, and 21.)

[BH22] Mihir Bellare and Viet Tung Hoang. Efficient schemes for committing authenticated encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 845–875, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland. (Cited on pages 23, 48, and 56.)

[Bih94] Eli Biham. New types of cryptanalytic attacks using related keys (extended abstract). In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409, Lofthus, Norway, May 23–27, 1994. Springer Berlin Heidelberg, Germany. (Cited on page 21.)

[BK03] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 491–506, Warsaw, Poland, May 4–8, 2003. Springer Berlin Heidelberg, Germany. (Cited on pages 21 and 22.)

[BK11] Mihir Bellare and Sriram Keelveedhi. Authenticated and misuse-resistant encryption of key-dependent data. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 610–629, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany. (Cited on pages 4, 18, 19, and 20.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications*

Security, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. (Cited on page 1.)

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer Berlin Heidelberg, Germany. (Cited on page 5.)

[BR14] Rishiraj Bhattacharyya and Arnab Roy. Secure message authentication against related-key attack. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 305–324, Singapore, March 11–13, 2014. Springer Berlin Heidelberg, Germany. (Cited on page 21.)

[BRS03] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002: 9th Annual International Workshop on Selected Areas in Cryptography*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75, St. John's, Newfoundland, Canada, August 15–16, 2003. Springer Berlin Heidelberg, Germany. (Cited on page 19.)

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448, Santa Barbara, CA, USA, August 14–18, 2005. Springer Berlin Heidelberg, Germany. (Cited on page 9.)

[CFN+12] Anne Canteaut, Thomas Fuhr, María Naya-Plasencia, Pascal Paillier, Jean-René Reinhard, and Marion Videau. A unified indifferentiability proof for permutation- or block cipher-based hash functions. Cryptology ePrint Archive, Report 2012/363, 2012. (Cited on page 14.)

[CR22] John Chan and Phillip Rogaway. On committing authenticated-encryption. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022: 27th European Symposium on Research in Computer Security, Part II*, volume 13555 of *Lecture Notes in Computer Science*, pages 275–294, Copenhagen, Denmark, September 26–30, 2022. Springer, Cham, Switzerland. (Cited on page 18.)

[CS14] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany. (Cited on pages 28 and 29.)

[DEMS16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. *Submission to the CAESAR Competition*, 2016. `http://competitions.cr.yp.to/round3/asconv12.pdf`. (Cited on pages 2, 3, and 16.)

[DGRW18] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 155–186, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. (Cited on page 23.)

[DM19] Christoph Dobraunig and Bart Mennink. Leakage resilience of the duplex construction. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 225–255, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland. (Cited on pages 3, 7, and 11.)

[DMV17] Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-state keyed duplex with built-in multi-user support. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 606–637, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland. (Cited on pages 3, 7, 8, and 11.)

[DPR+13] Yevgeniy Dodis, David Pointcheval, Sylvain Ruhault, Damien Vergnaud, and Daniel Wichs. Security analysis of pseudo-random number generators with input: /dev/random is not robust. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 647–658, Berlin, Germany, November 4–8, 2013. ACM Press. (Cited on page 3.)

[FKOS22] Sebastian Faust, Juliane Krämer, Maximilian Orlt, and Patrick Struck. On the related-key attack security of authenticated encryption schemes. In *Security and Cryptography for Networks: 13th International Conference, SCN 2022, Amalfi (SA), Italy, September 12–14, 2022, Proceedings*, pages 362–386. Springer, 2022. (Cited on pages 18 and 21.)

[FOR17] Pooya Farshim, Claudio Orlandi, and Răzvan Roşie. Security of symmetric primitives under incorrect usage of keys. *IACR Transactions on Symmetric Cryptology*, 2017(1):449–473, 2017. (Cited on page 23.)

[FP15] Pooya Farshim and Gordon Procter. The related-key security of iterated Even-Mansour ciphers. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*, pages 342–363, Istanbul, Turkey, March 8–11, 2015. Springer Berlin Heidelberg, Germany. (Cited on page 10.)

[GLR17] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 66–97, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland. (Cited on pages 4, 23, and 24.)

[Ham17] Mike Hamburg. The STROBE protocol framework. Cryptology ePrint Archive, Report 2017/003, 2017. (Cited on page 2.)

[KCP16] John Kelsey, Shu-Jen Chang, and Ray Perlner. SHA-3 derived functions: cSHAKE, KMAC, TupleHash, and ParallelHash. *NIST SP 800-185*, 2016. `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf`. (Cited on page 1.)

[Knu92] Lars R. Knudsen. Cryptanalysis of loki91. In *Advances in Cryptology - AUSCRYPT '92*, volume 718, pages 196–208, 1992. (Cited on page 21.)

[LGR21] Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning oracle attacks. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 195–212. USENIX Association, August 11–13, 2021. (Cited on page 4.)

[LLJ15] Xianhui Lu, Bao Li, and Dingding Jia. KDM-CCA security from RKA secure authenticated encryption. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 559–583, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany. (Cited on page 21.)

[Men22] Bart Mennink. Understanding the duplex and its security. Cryptology ePrint Archive, Report 2022/1340, 2022. (Cited on pages 3, 7, 8, and 11.)

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39, Cambridge, MA, USA, February 19–21, 2004. Springer Berlin Heidelberg, Germany. (Cited on pages 2, 9, and 11.)

[MRV15] Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of full-state keyed Sponge and duplex: Applications to authenticated encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 465–489, Auckland, New Zealand, November 30 – December 3, 2015. Springer Berlin Heidelberg, Germany. (Cited on pages 3, 7, and 11.)

[Pat90] Jacques Patarin. Pseudorandom permutations based on the DES scheme. In *EUROCODE*, volume 514 of *Lecture Notes in Computer Science*, pages 193–204. Springer, 1990. (Cited on page 28.)

[Pat09] Jacques Patarin. The "coefficients H" technique (invited talk). In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008: 15th Annual International Workshop on Selected Areas in Cryptography*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345, Sackville, New Brunswick, Canada, August 14–15, 2009. Springer Berlin Heidelberg, Germany. (Cited on page 28.)

[Per18] Trevor Perrin. Stateful hash objects: Api and constructions, 2018. `https://github.com/noiseprotocol/sho_spec/blob/master/output/sho.pdf`. (Cited on page 2.)

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390, St. Petersburg, Russia, May 28 – June 1, 2006. Springer Berlin Heidelberg, Germany. (Cited on pages 7, 19, and 21.)

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506, Tallinn, Estonia, May 15–19, 2011. Springer Berlin Heidelberg, Germany. (Cited on pages 3, 10, 11, and 20.)

A H-coefficient technique

The H-coefficient technique [Pat90, Pat09, CS14] is a method for bounding the advantage of a computationally unbounded adversary $\mathscr{A}$, which wlog can be assumed to be deterministic, attempting to distinguish between a real and an ideal game. The technique focuses on the transcripts generated when $\mathscr{A}$ interacts with the oracles in these games, namely, the sequence of input-output pairs $\tau = ((x_1, y_1), (x_2, y_2), \ldots, (x_q, y_q))$.

$\mathsf{p}(U,+)$		$\mathsf{p}(V,-)$	
1 :	**if** $U \notin \mathsf{G}$	1 :	**if** $V \notin \mathsf{G}$
2 :	$\quad \mathsf{G.add}(U)$	2 :	$\quad \mathsf{G.add}(V)$
3 :	**if** $\mathsf{G.out}(U) = \perp$	3 :	**if** $\mathsf{G.in}(V) = \perp$
4 :	$\quad V \twoheadleftarrow \{0,1\}^b$	4 :	$\quad U \twoheadleftarrow \{0,1\}^b$
5 :	$\quad$ **if** $V \in \mathsf{G} \wedge \mathsf{G.in}(V) \neq \perp$	5 :	$\quad$ **if** $U \in \mathsf{G} \wedge \mathsf{G.out}(U) \neq \perp$
6 :	$\quad\quad V \twoheadleftarrow \{0,1\}^b \setminus \mathsf{G}^-$	6 :	$\quad\quad U \twoheadleftarrow \{0,1\}^b \setminus \mathsf{G}^+$
7 :	$\quad$ **if** $V \notin \mathsf{G}$	7 :	$\quad$ **if** $U \notin \mathsf{G}$
8 :	$\quad\quad \mathsf{G.add}(V)$	8 :	$\quad\quad \mathsf{G.add}(U)$
9 :	$\quad \mathsf{G.out}(U) \leftarrow V,\ \mathsf{G.in}(V) \leftarrow U$	9 :	$\quad \mathsf{G.in}(V) \leftarrow U,\ \mathsf{G.out}(U) \leftarrow V$
10 :	**return** $\mathsf{G.out}(U)$	10 :	**return** $\mathsf{G.in}(V)$

Figure 12: Lazy-sampled instantiation of the random permutation $\mathsf{p}(\cdot,\cdot)$ through the graph-based data structure G for Game $\mathbf{G}1$.

We use $\mathcal{T}_{\text{ideal}}$ to denote the random variable corresponding to the transcript generated by $\mathscr{A}$ in the ideal game. Then, $\mathsf{P}_{\text{ideal}}(\tau)$ and $\mathsf{P}_{\text{real}}(\tau)$ denote the probabilities that a given transcript τ is generated in the corresponding game when interacting with $\mathscr{A}$. A transcript τ is said to be attainable if there exists an adversary such that the probability of generating τ in the ideal game is strictly greater than 0.

The H-coefficient technique relies on identifying a suitable partition of attainable transcripts, applying the following theorem, and then calculating ϵ_1 and ϵ_2 (for a proof, see [CS14]):

Theorem A.1 (H-coefficient Technique) *Let $\mathscr{A}$ be a computationally unbounded adversary trying to distinguish between a real game G^{Real} and an ideal game G^{Ideal}. Let $\mathbb{T} = \mathbb{T}_{\text{good}} \sqcup \mathbb{T}_{\text{bad}}$ be a partition of the set of attainable transcripts. If there exist $\epsilon_1, \epsilon_2 \geq 0$ such that*

$$\forall \tau \in \mathbb{T}_{\text{good}}, \frac{\mathsf{P}_{\text{real}}(\tau)}{\mathsf{P}_{\text{ideal}}(\tau)} \geq 1 - \epsilon_1 \qquad \textit{and} \qquad \Pr[\mathcal{T}_{\text{ideal}} \in \mathbb{T}_{\text{bad}}] \leq \epsilon_2,$$

then

$$\left|\Pr\left[\mathscr{A}^{G^{\text{Real}}} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{G^{\text{Ideal}}} \Rightarrow 1\right]\right| \leq \epsilon_1 + \epsilon_2.$$

B Proofs of Indifferentiability

B.1 Proof of Theorem 6.1

We now flesh out the proof in full detail. Note that in every game hop, except for the last, we only modify the primitive oracle and accordingly we only list the corresponding code in the game description. Except for game $\mathbf{G}7$, the construction oracle is always the Duplex construction described in Fig. 1 on page 5. While each game hop works for any (computationally unbounded) distinguisher, it simplifies our exposition to consider some fixed distinguisher $\mathscr{D}$. Accordingly, $(\mathbf{G}i = 1)$ denotes the event that $\mathscr{D}$ outputs 1 in game $\mathbf{G}i$, when interacting with the corresponding oracles. Moreover, q_c and q_p denote the number of queries that this distinguisher makes to the construction and permutation oracles, respectively.

Game $\mathbf{G}1$. This corresponds to the real world where the distinguisher interacts with the Duplex construction Dup and its underlying random permutation p. However the random permutation is realised via a lazy sampling procedure as shown in Fig. 12.

We view the lazy sampling of the random permutation p as the (dynamic) sampling of a directed graph. We store this graph in a data structure G indexed by the nodes in the graph. For each node $U \in \mathsf{G}$ we

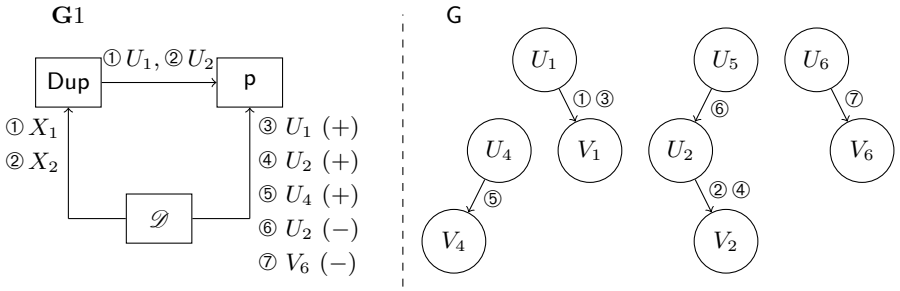

Figure 13: A pictorial example relating the distinguisher's queries to the graph G in $\mathbf{G}1$. The distinguisher queries $\mathsf{Dup.init}(X_1)$ and $\mathsf{Dup.next}(X_2)$, resulting in the corresponding forward queries to the primitive, $U_1 = IV \oplus \mathsf{encode}_0(X_1)$ and $U_2 = V_1 \oplus \mathsf{encode}_1(X_2)$, where $V_1 = \mathsf{p}(U_1)$. The distinguisher then queries the primitive directly with three forward queries, U_1, U_2 and $U_4 = V_1 \oplus \mathsf{encode}_1(X_4)$ for $X_4 \neq X_2$, and two backward queries U_2 and V_6, thereby creating nodes V_4, U_5, and U_6 in the resulting graph.

additionally store its predecessor $\mathsf{G.in}(U)$, i.e., the node from which there is an incoming edge, as well as its successor $\mathsf{G.out}(U)$, i.e., the node to which there is an outgoing edge. An edge from U to V represents the mapping $\mathsf{p} : U \to V$, and thus for every node the successor represents the image, and its predecessor represents its preimage. As p is a permutation, each node has at most one successor and one predecessor. For convenience, G is initialised to $\{IV\}$ but its predecessor and successor are not initially set.

Referring now to Fig. 12, we see that the procedures for handling forward and backward queries are almost identical. First they check whether the queried node is already in G, and if not they add it via the call $\mathsf{G.add}(\,)$. Whenever a node is added, its successor and predecessor are initialised to $\perp$. Then, if the successor (or predecessor) of the queried node is not defined it is sampled and added to G. The sampling is consistent with that of a random permutation, where it is first sampled uniformly at random and if the sampled node happens to already be the successor (or predecessor) of another node (Line 5), it is resampled from the set which excludes such nodes. Here G^- and G^+ denote respectively the sets of nodes in G with in-degree and out-degree one, i.e., $\mathsf{G}^- = \{U \in \mathsf{G} : \mathsf{G.in}(U) \neq \perp\}$ and similarly $\mathsf{G}^+ = \{U \in \mathsf{G} : \mathsf{G.out}(U) \neq \perp\}$. A pictorial example relating the distinguisher's queries to the graph G in game $\mathbf{G}1$ is shown in Figure 13.

Game G2. This is a reformulation of $\mathbf{G}1$ in preparation for the transition to $\boxed{\mathbf{G}2}$. We have replaced the random permutation in $\mathbf{G}1$ with the simulator $\mathsf{Sim}_{\mathsf{fc}}$ described in Fig. 14 (boxed code excluded). The construction oracle remains to be the Duplex construction as in $\mathbf{G}1$. The main differences between $\mathbf{G}2$ and $\mathbf{G}1$ are highlighted in gray. Namely, we introduced some bad flags, added if statements whose effect is only to set these flags, and introduced additional bookkeeping with respect to the nodes in G. These are described below in more detail.

Every node that is added to G is now labeled to keep track of how it was generated—whether it was sampled internally by $\mathsf{Sim}_{\mathsf{fc}}$ (marked by '\$') or specified by the distinguisher in a query (marked by '$*$'). More concretely, this can be observed at Lines 2 and 13 where the $\mathsf{G.add}(\cdot, \cdot)$ calls are augmented with '$*$' and '\$' respectively. For reasons that will become clear later on we label the IV node in G with \$. The other piece of bookkeeping we have added is that we now also store the capacity component of every node in G in a separate data structure C. The data structure C is automatically updated by the $\mathsf{G.add}(\cdot, \cdot)$ procedure and is initialised with the capacity component of IV, i.e., $\mathsf{C} = \{V_C : V \in \mathsf{G}\} \cup \{IV_C\}$.

The flags bad_1 and bad_2 get set in $\mathbf{G}2$ at Lines 6 and 10 respectively. The flag bad_2 is set in Line 10 due to a new check performed at Line 9 (and at Line 9 for the inverse case). This checks whether the capacity component V_C of a sampled node V is already contained in C. Note that the new labels and bad flags do not alter the functionality as they are never checked. The only exception is Line 9 which checks for membership in C but in $\mathbf{G}2$ it results in no action besides setting bad_2. Accordingly $\mathbf{G}2$ and $\mathbf{G}1$ remain

$\mathsf{Sim}_{\mathsf{fc}}(U,+)$		$\mathsf{Sim}_{\mathsf{fc}}(V,-)$	
1 :	**if** $U \notin \mathsf{G}$	1 :	**if** $V \notin \mathsf{G}$
2 :	$\mathsf{G.add}(U,*)$	2 :	$\mathsf{G.add}(V,*)$
3 :	**if** $\mathsf{G.out}(U) = \perp$	3 :	**if** $\mathsf{G.in}(V) = \perp$
4 :	$V \twoheadleftarrow \{0,1\}^b$	4 :	$U \twoheadleftarrow \{0,1\}^b$
5 :	**if** $(V \in \mathsf{G} \wedge \mathsf{G.in}(V) \neq \perp)$	5 :	**if** $U \in \mathsf{G} \wedge \mathsf{G.out}(U) \neq \perp$
6 :	$\mathsf{bad}_1 \leftarrow \mathsf{true}$	6 :	$\mathsf{bad}_1 \leftarrow \mathsf{true}$
7 :	$V \twoheadleftarrow \{0,1\}^b \setminus \mathsf{G}^-$	7 :	$U \twoheadleftarrow \{0,1\}^b \setminus \mathsf{G}^+$
8 :	$\boxed{V_R \twoheadleftarrow \{0,1\}^r, V_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}}$	8 :	$\boxed{U_R \twoheadleftarrow \{0,1\}^r, U_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}}$
9 :	**if** $V_C \in \mathsf{C}$	9 :	**if** $U_C \in \mathsf{C}$
10 :	$\mathsf{bad}_2 \leftarrow \mathsf{true}$	10 :	$\mathsf{bad}_2 \leftarrow \mathsf{true}$
11 :	$\boxed{V_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}}$	11 :	$\boxed{U_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}}$
12 :	**if** $V \notin \mathsf{G}$ // always true now	12 :	**if** $U \notin \mathsf{G}$ // always true now
13 :	$\mathsf{G.add}(V,\$)$	13 :	$\mathsf{G.add}(U,\$)$
14 :	$\mathsf{G.out}(U) \leftarrow V$, $\mathsf{G.in}(V) \leftarrow U$	14 :	$\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
15 :	**return** $\mathsf{G.out}(U)$	15 :	**return** $\mathsf{G.in}(V)$

Figure 14: The simulator in games $\mathbf{G}2$ and $\boxed{\mathbf{G}2}$. Game $\boxed{\mathbf{G}2}$ includes the boxed code and $\mathbf{G}2$ does not. In $\boxed{\mathbf{G}2}$ $\mathsf{Sim}_{\mathsf{fc}}$ ensures that sampled nodes have 'fresh' capacity components by resampling whereas the rate components are always uniformly distributed.

functionally equivalent and therefore:

$$\Pr[\,\mathbf{G}2 = 1] = \Pr[\,\mathbf{G}1 = 1]\ .$$

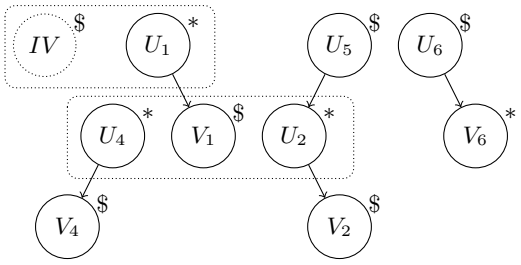

Figure 15: Recurring example of the graph in games $\mathbf{G}2$ and $\boxed{\mathbf{G}2}$ with the implicit node IV and the added labels displayed ($ or $*$). The dotted rectangles indicate nodes with the same capacity. Game $\boxed{\mathbf{G}2}$ ensures that the capacity component of a sampled node (label $) is new, as a result there will be at most one node labelled $ (a.k.a. the representative) among every group of nodes that share the same capacity component.

Game $\boxed{\mathbf{G}2}$. We now alter the sampling procedure so that the capacity component of every sampled node is guaranteed to be 'fresh' and not contained in C. The simulator $\mathsf{Sim}_{\mathsf{fc}}$ for $\boxed{\mathbf{G}2}$ is described in Fig. 14 where the boxed code is now included, whereas the construction oracle is still the Duplex construction. Note that in the new code in $\boxed{\mathbf{G}2}$, the capacity and rate components of a node are sampled separately and independently. Games $\mathbf{G}2$ and $\boxed{\mathbf{G}2}$ behave differently only when the internal flags bad_1 and bad_2 are set.

When bad_1 is set in $\boxed{\mathbf{G}2}$ at Line 6, the simulator overwrites V at Line 8 by resampling the rate component V_R uniformly at random and the capacity component V_C from the set $\{0,1\}^c \setminus \mathsf{C}$. Furthermore, if bad_2 gets set in $\boxed{\mathbf{G}2}$ at Line 10, then only the capacity component V_C of V gets resampled from $\{0,1\}^c \setminus \mathsf{C}$ and overwritten. Lemma B.1 bounds the distinguisher's ability to discern $\boxed{\mathbf{G}2}$ from $\mathbf{G}2$.

Game $\boxed{\mathbf{G2}}$ now ensures that every sampled node that is added to the graph G, marked '\$', has a unique capacity component. In contrast, note that no such restriction exists for the nodes in G marked with '$*$' as they can be chosen freely by the distinguisher. Nevertheless, we have now endowed the graph G with some useful structure that will come in handy later on. Now for every capacity component in C there exists only one node in G marked with '\$'. We call this node the *representative* of that capacity component and store it in C when it is added to G. The representative of any capacity component U_C in C can then be accessed via the call $\mathsf{C}.\mathsf{rep}(U_C)$. The other important feature is that since every edge involves at least one sampled node, no two adjacent nodes in G will share the same capacity component. Accordingly the capacity graph extracted from G is now free from any loops. A depiction of the prior graph example updated with the amendments of Games $\mathbf{G}2$ and $\boxed{\mathbf{G2}}$ is shown in Figure 15.

Lemma B.1 *Let q_p and q_c denote the number of queries made by $\mathscr{D}$ to its permutation and construction oracles, respectively, and let c denote the capacity. Then*

$$|\Pr[\boxed{\mathbf{G2}} = 1] - \Pr[\mathbf{G}2 = 1]| \leq \frac{(q_p + q_c + 1)^2}{2^c}.$$

Proof. The code in Fig. 14 shows that games $\mathbf{G}2$ and $\boxed{\mathbf{G2}}$ are *identical-until-bad*. It then follows that:

$$|\Pr[\boxed{\mathbf{G2}} = 1] - \Pr[\mathbf{G}2 = 1]| \leq \Pr[\mathsf{bad}_1 \vee \mathsf{bad}_2].$$

The flags bad_1 and bad_2 are set respectively at Lines 6 and 10 when either $(V \in \mathsf{G} \wedge \mathsf{G}.\mathsf{in}(V) \neq \perp)$ or $V_C \in \mathsf{C}$. Since $V \in \mathsf{G}$ implies $V_C \in \mathsf{C}$, we can then bound $\Pr[\mathsf{bad}_1 \vee \mathsf{bad}_2]$ by the probability of the single event $V_C \in \mathsf{C}$. The set C is initialized to $\{IV_C\}$ and at most $2(q_p + q_c)$ entries may be subsequently added. As every node is sampled at Line 4 uniformly at random from the set $\{0,1\}^b$, the probability of $V_C \in \mathsf{C}$ is then bounded by $\sum_{i=1}^{q_p+q_c} (2i)/2^c \leq (q_p + q_c + 1)^2/2^c$. □

Before transitioning to the next game it is important to note that $\mathsf{Sim}_{\mathsf{fc}}(\cdot, +)$ in Game $\boxed{\mathbf{G2}}$ can be simplified significantly by replacing the convoluted sampling procedure on Lines 4–11 with Line 8. This is because after sampling V uniformly at random, whenever $V_C \in \mathsf{C}$ then either bad_1 or bad_2 will occur, which results in V_C being resampled from $\{0,1\}^c \setminus \mathsf{C}$. However this sampling procedure is equivalent to directly sampling V_R uniformly at random and V_C from $\{0,1\}^c \setminus \mathsf{C}$. In addition Line 12 is now redundant since the capacity component of V will be new thereby guaranteeing that V is also new. The same argument holds also for $\mathsf{Sim}_{\mathsf{fc}}(\cdot, -)$.

Game G3. In this game the construction oracle remains to be the Duplex whereas the simulator's behaviour is modified to take into account the interface from which it receives a query. Nevertheless, the changes to the simulator will only affect the labelling applied to the nodes in the graph and not the distribution of its outputs. Accordingly we will show that $\mathbf{G}3$ is indistinguishable from $\boxed{\mathbf{G2}}$.

The simulator used in $\mathbf{G}3$ is shown in Fig. 16, which is obtained from a simplified form of the simulator in $\boxed{\mathbf{G2}}$ with the forward direction replicated into two separate algorithms and then applying the alterations indicated by the highlighted lines of code. Algorithm $\mathsf{Sim}_{\mathsf{p}}$ handles direct primitive queries from the distinguisher and $\mathsf{Sim}_{\mathsf{c}}$ handles internal calls from the construction. Under certain conditions, Lines 1–2, $\mathsf{Sim}_{\mathsf{p}}$ may call $\mathsf{Sim}_{\mathsf{c}}$ but this is mainly done to avoid code repetition and present the code of $\mathsf{Sim}_{\mathsf{p}}$ more concisely. We also introduce new labelling and in this game the interface-dependent behaviour of the simulator differs only in how this labelling is applied. Specifically, every node that is added to G is additionally labelled with 'd' or 't'. This labelling is extended to the corresponding entries in C, effectively subdividing it into two disjoint sets which we denote by C_{d} and C_{t}.

The labelling strategy is as follows: the node IV is initialised with the label 't', nodes added by $\mathsf{Sim}_{\mathsf{c}}$ are always labelled with 't', whereas nodes added by $\mathsf{Sim}_{\mathsf{p}}$ are labelled with 'd' except when 1)

$\mathsf{Sim_p}(U,+)$

1 : **if** $U_C \in \mathsf{C_t}$
2 : **return** $\mathsf{Sim_c}(U,+)$
3 : **if** $U \notin \mathsf{G}$
4 : $\mathsf{G.add}(U,*,\mathsf{d})$
5 : **if** $\mathsf{G.out}(U) = \perp$
6 : $V_R \twoheadleftarrow \{0,1\}^r$, $V_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}$
7 : $\mathsf{G.add}(V,\$,\mathsf{d})$
8 : $\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
9 : **return** $\mathsf{G.out}(U)$

$\mathsf{Sim_c}(U,+)$

1 : **if** $U \notin \mathsf{G}$ // It must be that $U_C \in \mathsf{C_t}$
2 : $\mathsf{G.add}(U,*,\mathsf{t})$
3 : **if** $\mathsf{G.out}(U) = \perp$
4 : $V_R \twoheadleftarrow \{0,1\}^r$, $V_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}$
5 : $\mathsf{G.add}(V,\$,\mathsf{t})$
6 : $\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
7 : **return** $\mathsf{G.out}(U)$

$\mathsf{Sim_p}(V,-)$

1 : **if** $V \notin \mathsf{G}$
2 : **if** $V_C \in \mathsf{C_t}$
3 : $\mathsf{G.add}(V,*,\mathsf{t})$
4 : **else**
5 : $\mathsf{G.add}(V,*,\mathsf{d})$
6 : **if** $\mathsf{G.in}(V) = \perp$
7 : $U_R \twoheadleftarrow \{0,1\}^r$, $U_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}$
8 : $\mathsf{G.add}(U,\$,\mathsf{d})$
9 : $\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
10 : **return** $\mathsf{G.in}(V)$

Figure 16: The simulator in Game $\mathbf{G}3$ with separate interfaces. The adversary is given access to $\mathsf{Sim_p}$ and $\mathsf{Sim_c}$ is the interface given to the construction.

there already exists another node in G with the same capacity component labelled 't' or 2) the node's predecessor is labelled 't'. The first exception ensures consistency in that any two nodes with the same capacity component will have the same label thereby ensuring that the labelling in C is well-defined. This is checked for and executed at Lines 2 and 5 in $\mathsf{Sim_p}(\cdot,-)$. The second exception essentially detects when the distinguisher's queries are emulating a Duplex session or extending a prior one and labels the resulting freshly sampled nodes with 't' as if it were created by a query from the construction oracle. This is detected in $\mathsf{Sim_p}(\cdot,+)$ at Line 1 by checking whether the capacity component of the queried node is in $\mathsf{C_t}$, and if true, it forwards the query to $\mathsf{Sim_c}$ at Line 2. Note that after Line 2, $\mathsf{Sim_p}(\cdot,+)$ is identical to $\mathsf{Sim_c}(\cdot,+)$ except that the 't' label is applied instead of 'd'. A depiction of this labelling strategy with respect to our recurring example appears in Figure 17.

The net effect of this labelling is that $\mathsf{C_t}$ identifies a subgraph within the capacity graph, where this subgraph is a *directed tree* (hence the label 't') with IV_C as its root. Moreover, every Duplex session—whether

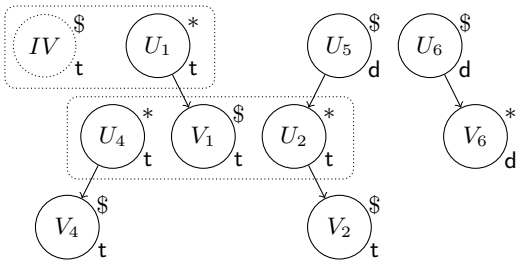

Figure 17: Recurring example of the graph in game $\mathbf{G}3$ with its additional labelling displayed (d or t). Note that $\mathscr{D}$'s query U_4 to the primitive interface gets labeled with t since it branches the path IV, U_1, U_2 (on the capacity graph) corresponding to a Duplex session. In contrast, the inverse query at U_2 yields a node U_5 with label d.

evaluated through the constructions oracle, the primitive oracle, or a combination of both—corresponds to a unique path on this tree from IV_C to the Duplex's current internal state S_C. The full Duplex state S at any point on this path corresponds to the representative $S = \mathsf{C.rep}(S_C)$ of that capacity component, and every edge (U_C, V_C) on this tree corresponds to the Duplex input $X = (\mathsf{C.rep}(U_C) \oplus \mathsf{G.in}(\mathsf{C.rep}(V_C)))_R$. Conversely, every path from the root corresponds to a unique sequence of Duplex queries. On the other hand, nodes resulting from queries to the primitive oracle that are unrelated to a Duplex session will be labelled 'd'—denoting that they are *disconnected* from the root as no path exists in the capacity graph from the root to these nodes. Note however that the capacity graph may still contain edges directed from $\mathsf{C_d}$ to $\mathsf{C_t}$ (but not vice versa). These observations follow directly from the Duplex construction, the freshness of the capacity components of sampled nodes (representatives), and the labelling strategy applied by our simulator.

As already noted, in Game $\mathbf{G}3$ we have only split the forward direction of the simulator $\mathsf{Sim_{fc}}$ into two separate interfaces—$\mathsf{Sim_p}(\cdot,+)$ and $\mathsf{Sim_c}(\cdot,+)$—which preserve its functionality, and added an extra label to the nodes in the graph. Further note that when the labelling of a node is checked, it is only to decide which label to assign to some other new node. Hence, at this point, the newly introduced labelling does not affect any other aspect of the simulator except for the labelling itself, and in particular it does not affect the functionality of the simulator or the distribution of its outputs. It then follows that:

$$\Pr[\,\mathbf{G}3 = 1] = \Pr\left[\,\boxed{\mathbf{G}2} = 1\right] .$$

Game G4. In this game we introduce an ORO instance that will be used by the simulator to compute the rate component of the sampled nodes with the label 't', instead of sampling it uniformly at random. The sampling of the capacity component of the added nodes remains unchanged. Further note that in this game the ORO is only accessible to the simulator and not the distinguisher, and in particular, the construction oracle continues to be the Duplex. Intuitively, we are adjusting the *representative* nodes corresponding to the capacity components in the directed tree to have their rate components match the ORO output. As can be observed in Fig. 18, V_R is no longer sampled at Line 12, where V_C is sampled, but is instead set within Lines 3–10 which in turn make use of a new subprocedure createNewID and a new data structure ID. Throughout Lines 3–10 and createNewID a number of calls are made to the ORO. Note that this change implicitly affects $\mathsf{Sim_p}(\cdot,+)$ as well, since it makes an internal call to $\mathsf{Sim_c}(\cdot,+)$ whenever it needs to create a node with label 't'.

At a high level, for the simulator to align the tree's representative nodes with the ORO, it needs to map these primitive queries to ORO queries. Towards this goal, it first identifies the path in the tree on which the capacity component of the primitive query is located, and then map that path to an ORO session. Note that nodes labelled 't' can only be created in response to forward queries whose capacity component is already in $\mathsf{C_t}$, and therefore it is always possible to locate the primitive query on the tree. In addition, a primitive query will either extend a new path from the root, extend an existing path at a leaf node, or branch off an existing path from an inner node. The first two cases are relatively easily to handle whereas the third case is a bit more involved and is where the createNewID procedure comes into play. For example, the second case is easily handled via a lookup to the data structure ID which maps a leaf in the capacity tree to its respective ORO session identifier id and the representative node S of that leaf. The data structure ID is initially empty and gradually populated by the simulator and createNewID. Once the session identifier is obtained it remains to recover the Duplex query which would have resulted in this same primitive query and submit this to the ORO. This is simply the XOR of the rate components of the leaf's representative and the primitive query.

We now explain the operation of the new code in more detail. Lines 3–5 handle the case where the query is extending a new path from the root. This case is identified by checking whether the capacity component of the queried node U is equal to IV_C. In that case, the simulator makes an initialisation query to the ORO

$\mathsf{Sim_p}(U, +)$

1 : **if** $U_C \in \mathsf{C_t}$
2 : **return** $\mathsf{Sim_c}(U, +)$
3 : **if** $U \notin \mathsf{G}$
4 : $\mathsf{G.add}(U, *, \mathsf{d})$
5 : **if** $\mathsf{G.out}(U) = \perp$
6 : $V_R \twoheadleftarrow \{0,1\}^r$, $V_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}$
7 : $\mathsf{G.add}(V, \$, \mathsf{d})$
8 : $\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
9 : **return** $\mathsf{G.out}(U)$

$\mathsf{Sim_c}(U, +)$

1 : **if** $U \notin \mathsf{G}$ // It must be that $U_C \in \mathsf{C_t}$
2 : $\mathsf{G.add}(U, *, \mathsf{t})$
3 : **if** $U_C = IV_C$
4 : $(V_R, id) \leftarrow \mathsf{ORO.init}(U_R \oplus IV_R)$
5 : $\mathsf{ID}[U_C] \leftarrow (id, U)$
6 : **else**
7 : $(id, S) \leftarrow \mathsf{ID}[U_C]$
8 : **if** $id = \perp$
9 : $(id, S) \leftarrow \mathsf{createNewID}(U_C)$
10 : $V_R \leftarrow \mathsf{ORO.next}(U_R \oplus S_R, id)$
11 : **if** $\mathsf{G.out}(U) = \perp$
12 : $V_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}$
13 : $\mathsf{G.add}(V, \$, \mathsf{t})$
14 : $\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
15 : **else**
16 : $V \leftarrow \mathsf{G.out}(U)$
17 : $\mathsf{ID.update}(U_C, V)$
18 : **return** $\mathsf{G.out}(U)$

$\mathsf{Sim_p}(V, -)$

1 : **if** $V \notin \mathsf{G}$
2 : **if** $V_C \in \mathsf{C_t}$
3 : $\mathsf{G.add}(V, *, \mathsf{t})$
4 : **else**
5 : $\mathsf{G.add}(V, *, \mathsf{d})$
6 : **if** $\mathsf{G.in}(V) = \perp$
7 : $U_R \twoheadleftarrow \{0,1\}^r$, $U_C \twoheadleftarrow \{0,1\}^c \setminus \mathsf{C}$
8 : $\mathsf{G.add}(U, \$, \mathsf{d})$
9 : $\mathsf{G.in}(V) \leftarrow U, \mathsf{G.out}(U) \leftarrow V$
10 : **return** $\mathsf{G.in}(V)$

$\mathsf{createNewID}(\bar{V}_C)$

1 : $i \leftarrow 0$
2 : $S \leftarrow \mathsf{C.rep}(\bar{V}_C)$, $\bar{V} \leftarrow S$
3 : **while** $\bar{V} \neq IV$ // executes at least once
4 : $\bar{U} \leftarrow \mathsf{G.in}(\bar{V})$
5 : $\bar{V} \leftarrow \mathsf{C.rep}(\bar{U}_C)$
6 : $X_i \leftarrow \bar{V}_R \oplus \bar{U}_R$
7 : $i \leftarrow i + 1$
8 : $i \leftarrow i - 1$
9 : $(Z, id) \leftarrow \mathsf{ORO.init}(X_i)$
10 : **while** $i > 0$
11 : $i \leftarrow i - 1$
12 : $Z \leftarrow \mathsf{ORO.next}(X_i, id)$
13 : $\mathsf{ID}[S_C] \leftarrow (id, S)$
14 : **return** (id, S)

Figure 18: The simulator used in Game **G**4 which aligns the rate components of the representative nodes of the set $\mathsf{C_t}$ with ORO outputs.

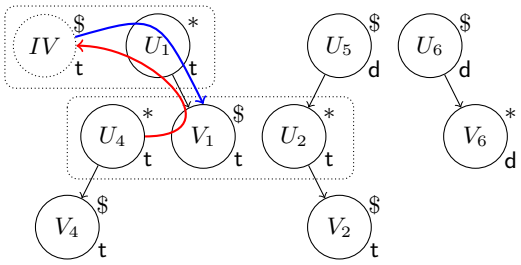

Figure 19: Recurring example of the graph in game **G**4 depicting the operation of createNewID. The figure illustrates the point at which U_4 is queried and createNewID(U_4) is invoked to create a new ORO session and add a matching entry in ID. The red arrow indicates the phase in which createNewID traces the path back to the IV and recovers the Duplex query $X_1 = (IV \oplus U_1)_R$. The blue arrow indicates the second phase, in which createNewID initiates a new ORO session and traces the path in forward direction by reproducing the recovered queries to the ORO, in this case X_1, and stops at V_1 (the representative for the capacity component of its input U_4). Once createNewID(U_4) has terminated, the simulator extends that ORO session by querying $(U_4 \oplus V_1)_R$ and sets the rate component of V_4 equal to the returned value.

on the value $U_R \oplus IV_R$ and stores the returned session identifier id in ID together (temporarily) with the primitive query U. The value V_R returned by ORO will comprise the rate component of the representative node V extending the new path from IV_C. Its capacity component is then sampled on Line 12 as before. Now that the representative node has been determined, the temporary entry in ID is updated on Line 17 where the operation ID.update(U_C, V) removes the entry ID$[U_C] = (id, U)$ and adds ID$[V_C] = (id, V)$.

Alternatively if U_C is different from IV_C (Line 6) the simulator consults ID to check if there is an entry for U_C. Now if the primitive query is extending a path there will be an entry in ID which will return the representative S of U_C and the session identifier id. The simulator uses this information to make a next query of the form $(U_R \oplus S_R, id)$ to the ORO (Line 10). Then if U was not queried before, a new node is constructed and set as its successor from the rate component returned by the ORO and an internally-sampled capacity component. Again prior to returning the successor of U as output, the entry in ID is updated to point to this node. It should be noted that not all entries in ID correspond to leafs in the tree, as there may be Duplex sessions which replicate part of a prior session, in which case, there will be an entry in ID pointing to the last node in the corresponding subpath—which is not a leaf node. However it also possible that the distinguisher first emulates a Duplex session through its primitive oracle and then directly makes a primitive query that branches off the corresponding path at some point in the middle. In the example graph portrayed in Figure 19 query U_4 is of this form: the ORO session has already progressed from U_2 to V_2, where the capacity components of U_2 and U_4 are equal, but no entry in ID points to the capacity component of U_4 anymore. Such a case will trigger the condition $id = \perp$ on Line 8 and call the subprocedure createNewID to create a new ORO session that points to U_C.

Given any element in C the subprocedure createNewID traces back the path from that point to the root while simultaneously working out (in reverse order) the sequence of Duplex queries corresponding to that path. Then it starts an ORO session and reproduces the Duplex queries (in correct order) to the ORO and adds a corresponding entry in ID. More specifically, given the input $\bar{V}_C$ it first recovers the full Duplex state by looking up in C for its representative $\bar{V}$ and then looks up its predecessor $\bar{U}$ in G. At this point it has worked out the input to the permutation but it still needs to recover the Duplex query that led to this input. For this it looks up the representative of $\bar{U}_C$ (corresponds to the output of the prior permutation), assigns it to $\bar{V}$, and recovers the Duplex query $X_i = \bar{U}_R \oplus \bar{V}_R$. It then repeats this procedure with the new value of $\bar{V}_C$ until it recovers a $\bar{V}$ equal to IV, at which point it has recovered all Duplex queries and can reproduce them to the ORO. It then adds an entry for this session and the representative of U_C in ID and returns these values to the simulator. At this point the simulator continues on Line 10 exactly as in the other cases when an entry for U_C is found in ID—it extends the session with another ORO query, creates

a new node V with a matching rate component, and updates the corresponding entry in ID with this new node on Line 17.

In this game we have only changed the sampling procedure for the rate component of the representative nodes in the tree identified by $\mathsf{C_t}$. Now each time a representative node is sampled by the simulator a new node and a new edge are added to this tree, resulting in a new path from the root to this new node. As we established in **G**3, each path identifies a unique sequence of Duplex queries, and accordingly a new path corresponds to a new sequence of Duplex queries. These are the Duplex queries that get forwarded to the ORO in order to compute the rate component of the new representative node. Then, since the sequence of queries is new the ORO will return a freshly-sampled output that is uniformly distributed. As such, the sampling distribution of the graph nodes in **G**4 is identical to that in **G**3, and hence

$$\Pr[\,\mathbf{G}4 = 1] = \Pr[\,\mathbf{G}3 = 1]\ .$$

Game G5. At this point $\mathsf{Sim_p}$ and $\mathsf{Sim_c}$ are still fairly intertwined. We can easily replace the call to $\mathsf{Sim_c}$ in $\mathsf{Sim_p}$ with its code, so that there are no internal calls between the two, but they still share access to the ORO and the graph data structure G. Our goal now is to remove the influence of $\mathsf{Sim_c}$ on $\mathsf{Sim_p}$. In particular we want that the operation of $\mathsf{Sim_p}$ is not affected by queries to $\mathsf{Sim_c}$, or rather, that construction calls to the simulator bear no influence on the simulator's response to direct primitive calls by the distinguisher. The source of this dependency lies in the shared data structure G, which is populated by both algorithms and in turn affects the operation of both algorithms. There are three operations in $\mathsf{Sim_p}$ that are affected by $\mathsf{Sim_c}$ in this way: when $\mathsf{Sim_p}$ retrieves a node labelled 't' that was previously sampled by $\mathsf{Sim_c}$; when $\mathsf{Sim_p}$ checks for membership in $\mathsf{C_t}$; and when $\mathsf{Sim_p}$ samples from the set $\{0,1\}^c \setminus \mathsf{C}$. The game hop we consider next, from **G**5 to $\boxed{\mathbf{G}5}$, makes progress towards lifting the first dependency and removes the second one, whereas the third dependency will be removed in the game hop from $\boxed{\mathbf{G}6}$ to **G**6. However, before discussing further these game transitions we first describe how **G**5 is derived from **G**4.

Game **G**5 is a reformulation of **G**4 with $\mathsf{Sim_p}$ additionally applying an extra label 'p' to some of the nodes it creates. In addition a new data structure $\mathsf{L_p}$, initialised to $\{IV_C\}$, is introduced which stores the capacity components of the nodes marked 'p'. Only nodes marked 't' can be marked 'p' and therefore $\mathsf{L_p} \subseteq \mathsf{C_t}$. Indeed, in future games $\mathsf{L_p}$ will serve as a local copy of $\mathsf{C_t}$. A description of $\mathsf{Sim_p}$ in game **G**5 appears in Fig. 20, where $\mathsf{Sim_c}$ remains as in **G**4 and the construction oracle remains the Duplex. Here the check for membership in $\mathsf{C_t}$ on Line 1 in **G**4 has been split between Lines 1 and 24 as membership in $\mathsf{L_p}$ and $\mathsf{C_t} \setminus \mathsf{L_p}$ respectively. Whenever the conditions on Line 24 in $\mathsf{Sim_p}(\cdot,+)$ and Line 4 in $\mathsf{Sim_p}(\cdot,-)$ are satisfied the flag bad is set, but this change does not alter the functionality of $\mathsf{Sim_p}$. In addition the call to $\mathsf{Sim_c}$ on Line 2, when $U_C \in \mathsf{L_p}$, has been replaced with the code of $\mathsf{Sim_c}$. The other shaded lines of code highlight the points at which the label 'p' is applied, where in $\mathsf{Sim_p}(V,-)$ this is determined by whether the capacity component of the node in question is already in $\mathsf{L_p}$ or not. Thus the contents of $\mathsf{L_p}$ only affect which other nodes get assigned this new label, but have no effect on the output distribution of $\mathsf{Sim_p}$, and therefore

$$\Pr[\,\mathbf{G}5 = 1] = \Pr[\,\mathbf{G}4 = 1]\ .$$

Game $\boxed{\mathbf{G}5}$. The most direct way in which the distinguisher can cause a node that was sampled by $\mathsf{Sim_c}$ to be returned by $\mathsf{Sim_p}$, is by reproducing the internal Duplex calls corresponding to prior construction queries. Such nodes are marked (\$,t) and their rate components are predetermined by the ORO, irrespective of whether they are set by $\mathsf{Sim_c}$ or $\mathsf{Sim_p}$. Accordingly, only the capacity components of these nodes are sampled by $\mathsf{Sim_c}$ or $\mathsf{Sim_p}$. However, when a node is sampled by $\mathsf{Sim_c}$ as a result of a construction query the capacity component of that node remains hidden from the adversary. The modification that we introduce in $\boxed{\mathbf{G}5}$ is that, in the most likely case just described, $\mathsf{Sim_p}$ will now overwrite the capacity components, previously set by $\mathsf{Sim_c}$, by resampling them anew. As a result, the capacity components sampled by

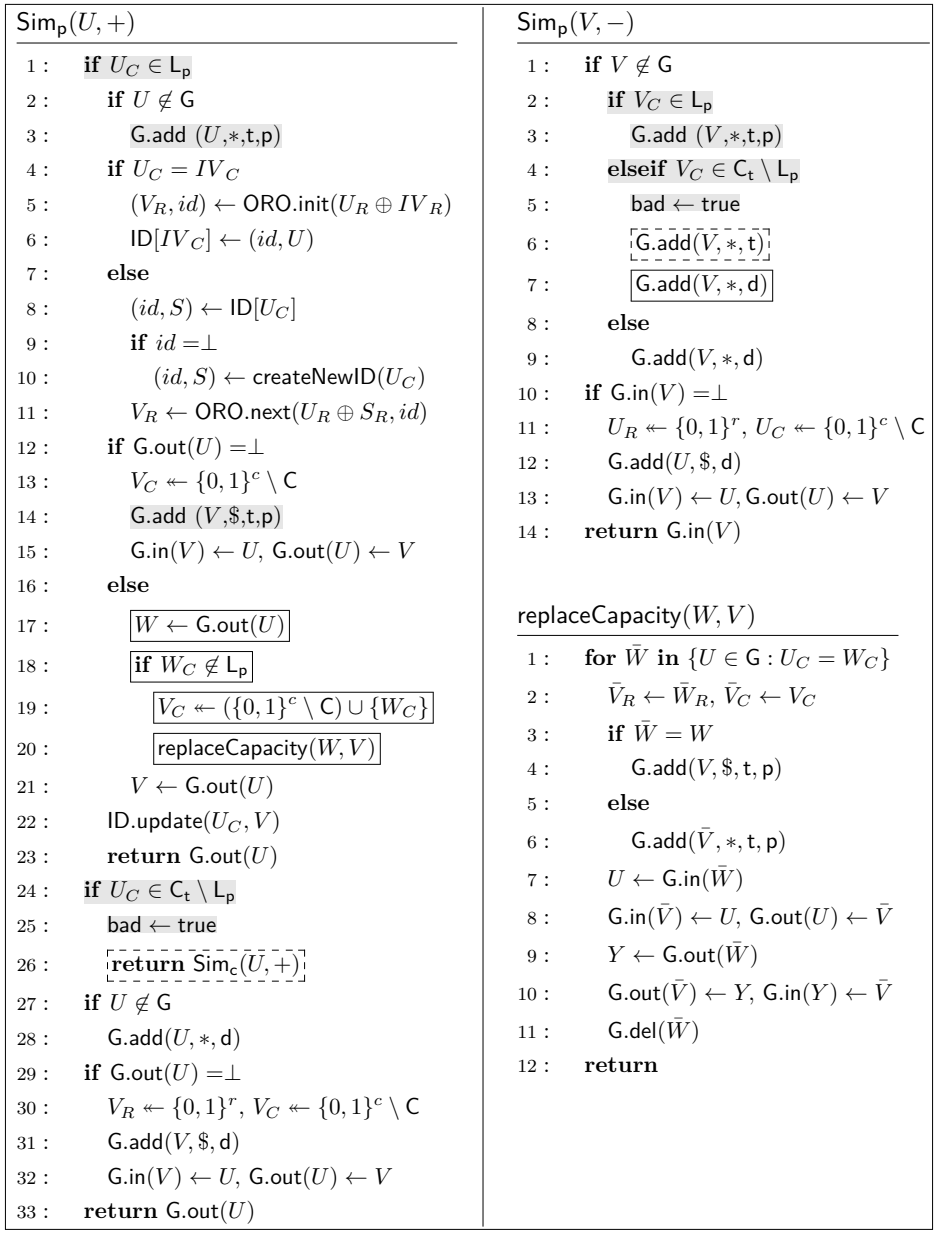

Figure 20: Description of $\mathsf{Sim_p}$ in games $\mathbf{G}5$ and $\boxed{\mathbf{G}5}$. Here $\mathbf{G}5$ includes code in the dashed boxes but not the code in solid boxes, whereas $\boxed{\mathbf{G}5}$ includes the code in solid boxes but not the code in the dashed boxes. In both games $\mathsf{Sim_c}$ remains unchanged.

$\mathsf{Sim_c}$ will only be *temporary* and will be reassigned and *finalised* when they are retrieved, labelled 'p', and returned by $\mathsf{Sim_p}$. We show that this resampling leaves the distribution of the affected capacity components unaltered, and the two games are indistinguishable unless $\mathsf{Sim_p}$ returns a temporary capacity component without resampling it—which will only happen with small probability.

This resampling happens between Lines 17 and 20 in $\mathsf{Sim_p}(U, +)$ and it is triggered when the queried node U has its capacity component in $\mathsf{L_p}$ but the capacity component of its successor is not, meaning that it is still temporary and needs to be resampled. Note that the capacity components sampled by $\mathsf{Sim_c}$ are contained in $\mathsf{C_t}$ but only get added to $\mathsf{L_p}$ by $\mathsf{Sim_p}$ once they are finalised; thus the set $\mathsf{C_t} \setminus \mathsf{L_p}$ identifies the set of temporary capacity components. Then, a new node V is sampled to replace W as the successor of U. The rate component V_R has already been sampled in Lines 5–11 (which will be identical to W_R) so only V_C needs to be sampled. The resampling is done with replacement, where V_C can, with some small probability, get reassigned the value W_C. The fact that we sample with replacement will be central to our argument for showing that this resampling does not alter the distribution of the capacity components. The procedure $\mathsf{replaceCapacity}(W, V)$ then updates the graph G by replacing W with V while preserving its structure. Essentially it searches through all nodes $\bar{W}$ with the same capacity component as W, creates a new node $\bar{V}$ with rate component equal to $\bar{W}_R$ and a capacity component equal to V_C, updates the neighbours of $\bar{W}$ to point to $\bar{V}$ instead, and then deletes $\bar{W}$. Note that when the newly sampled node V is added to G, the labels used are ($, t, p), meaning that V_C is added to $\mathsf{L_p}$ and V takes over the role of the representative from W.

The main observation behind this modification is that from the perspective of $\mathsf{Sim_c}$ the capacity components only serve to locate nodes (primitive queries) on a path, as in the $\mathsf{createNewID}$ procedure, but their actual value is not important as long as they are unique and the graph structure is preserved. This is because the output of the Duplex depends only on the path and the value of the rate components it traverses, whereas the capacity components eventually get discarded. The values of the capacity components are only exposed when they are returned to the adversary via $\mathsf{Sim_p}$. However, temporary values may still leak when the queried node is in $\mathsf{C_t} \setminus \mathsf{L_p}$ in which case no resampling takes place. However, we will show that this can only occur with negligible probability. We introduce another modification in $\boxed{\mathbf{G}5}$ in that we drop parts of the code that are triggered by queries on a node in $\mathsf{C_t} \setminus \mathsf{L_p}$. The immediate effect of this change is that $\mathsf{Sim_p}$ will no longer call $\mathsf{Sim_c}$ and if a node is queried to $\mathsf{Sim_p}(\cdot, -)$ whose capacity is already in $\mathsf{C_t} \setminus \mathsf{L_p}$ it will now be added with a label 'd'. We next show that the resampling introduced in $\boxed{\mathbf{G}5}$ does not alter the distribution of the capacity components returned by $\mathsf{Sim_p}$ as long as it is not queried on temporary capacity components. This is stated more formally in the following lemma.

Lemma B.2 *For any distinguisher making q_c construction queries and $q_p \leq 2^{c-2}$ primitive queries, its ability to distinguish Games $\mathbf{G}5$ and $\boxed{\mathbf{G}5}$ is bounded by:*

$$\left|\Pr\left[\mathbf{G}5 = 1\right] - \Pr\left[\boxed{\mathbf{G}5} = 1\right]\right| \leq \frac{2q_c q_p}{2^c}.$$

Proof. We prove the above lemma by applying the H-coefficient technique (cf. Appendix A) were games $\mathbf{G}5$ and $\boxed{\mathbf{G}5}$ will correspond to the Real world and Ideal world respectively. The transcript τ consists of the construction queries and the queries to $\mathsf{Sim_p}(\cdot, \cdot)$ that the distinguisher makes. We say that a transcript τ is bad if it contains a query to $\mathsf{Sim_p}$ whose capacity component is contained in $\mathsf{C_t} \setminus \mathsf{L_p}$. In the game description such an event will set the bad flag. Accordingly, the set of attainable transcripts is partitioned into good and bad transcript, i.e., $\mathbb{T} = \mathbb{T}_{\text{good}} \sqcup \mathbb{T}_{\text{bad}}$. It then follows that:

$$\left|\Pr\left[\mathbf{G}5 = 1\right] - \Pr\left[\boxed{\mathbf{G}5} = 1\right]\right| \leq \epsilon_1 + \epsilon_2,$$

where ϵ_1 and ϵ_2 are positive real numbers such that

$$\forall \tau \in \mathbb{T}_{\text{good}}, \frac{\mathsf{P}_{\text{real}}(\tau)}{\mathsf{P}_{\text{ideal}}(\tau)} \geq 1 - \epsilon_1 \qquad \text{and} \qquad \Pr[\,\mathcal{T}_{\text{ideal}} \in \mathbb{T}_{\text{bad}}] \leq \epsilon_2.$$

We will show that $\epsilon_1 = 0$ and $\epsilon_2 = \frac{q_c q_p}{2^{c-1}}$. We start by bounding the probability of a bad transcript in the ideal world, i.e., $\boxed{\mathbf{G5}}$. Let bad^i be the event that bad is set during the first i queries to $\mathsf{Sim_p}$, so that $\Pr[\,\mathsf{bad}] = \Pr\left[\bigvee_{i=1}^{q_p} \mathsf{bad}^i\right]$ and

$$\Pr[\,\mathsf{bad}] \leq \sum_{i=1}^{q_p} \Pr\left[\,\mathsf{bad}^i \;\middle|\; \neg\mathsf{bad}^{i-1}\right]. \tag{1}$$

Recall that the set C of capacity components is partitioned into three mutually exclusive sets: $\mathsf{C_d}$, $\mathsf{L_p}$, and $\mathsf{C_t} \setminus \mathsf{L_p}$, where $\mathsf{L_p} \subseteq \mathsf{C_t}$. In the ideal world, values in $\mathsf{C_t} \setminus \mathsf{L_p}$ can only be added by $\mathsf{Sim_c}$ in response to construction queries, which are always sampled at random and never revealed in the output. On the other hand, values in $\mathsf{C_d}$ and $\mathsf{L_p}$ can only be added by $\mathsf{Sim_p}$ during primitive queries. These are known to the distinguisher, since they were either part of the input to $\mathsf{Sim_p}$, and thus chosen by the distinguisher, or returned by $\mathsf{Sim_p}$ as part of its output. Then, the important point to note is that conditioned on bad not occurring, $\mathsf{Sim_p}$ will not have revealed any of the capacity components contained in $\mathsf{C_t} \setminus \mathsf{L_p}$ to the distinguisher.

By construction, the nodes with capacity components in $\mathsf{C_t} \setminus \mathsf{L_p}$ can only have outgoing edges to nodes in that same set. On the other hand, incoming edges are possible from all three sets $\mathsf{C_d}$, $\mathsf{L_p}$, and $\mathsf{C_t} \setminus \mathsf{L_p}$. However, edges from $\mathsf{C_d}$ to $\mathsf{C_t} \setminus \mathsf{L_p}$ can only be created when a node is queried to $\mathsf{Sim_p}(\cdot, -)$ that has a capacity component in $\mathsf{C_t} \setminus \mathsf{L_p}$. Thus, conditioned on bad not occurring, such edges will not be present. Then, based on which edges are possible, the only queries that can return a node in $\mathsf{C_t} \setminus \mathsf{L_p}$ are queries to $\mathsf{Sim_p}(\cdot, +)$ or $\mathsf{Sim_p}(\cdot, -)$ with a capacity component in $\mathsf{C_t} \setminus \mathsf{L_p}$, and queries to $\mathsf{Sim_p}(\cdot, +)$ contained in $\mathsf{L_p}$. Queries of the former type are again excluded by the assumption that bad is not set. Finally, queries of the latter type on a node with an outgoing edge to a node with a capacity component in $\mathsf{C_t} \setminus \mathsf{L_p}$, will invoke Lines 17 to 20. As a result, the capacity component of the output node will be resampled and overwritten, where the old output node is purged from G and replaced with this new node whose capacity component is assigned to $\mathsf{L_p}$.

Thus, from the perspective of the distinguisher, at primitive query i and conditioned on bad^{i-1} being false, the set $\mathsf{C_t} \setminus \mathsf{L_p}$ is a random subset of at least $2^c - 2i$ possible string values, since each prior query eliminated at most two values. It then follows that

$$\Pr\left[\,\mathsf{bad}^i \;\middle|\; \neg\mathsf{bad}^{i-1}\right] \leq \frac{|\mathsf{C_t} \setminus \mathsf{L_p}|}{2^c - 2i}.$$

In addition, the size of $\mathsf{C_t} \setminus \mathsf{L_p}$ is bounded above by q_c and i is bounded above by q_p, yielding

$$\leq \frac{q_c}{2^c - 2q_p}.$$

Further imposing the constraint $q_p \leq 2^{c-2}$ and summing up we obtain the claimed bound

$$\Pr[\,\mathsf{bad}] \leq \frac{q_c q_p}{2^{c-1}}. \tag{2}$$

We next establish that $\epsilon_1 = 0$ by showing that the interpolation probability of any good transcript is identical in both worlds. To prove this, we focus on how and when the node values are sampled. In both worlds the rate components of nodes are sampled independently of the capacity components and in

an identical manner. Accordingly we restrict our attention to how the capacity components are sampled. The capacity components are only revealed in the transcript through simulator queries but may have been sampled at a prior point in time through a construction query. Now, for a given transcript the queries are fixed and correspondingly the order in which these capacity components are sampled is also fixed with respect to each world. However, at any point in time not all capacity components that have been sampled will have been revealed in the transcript. Accordingly the set of capacity components can be subdivided into revealed and hidden. Now, in the ideal world, some capacity components will be resampled as they are revealed, whereas in the real world this resampling never happens. Thus, in the real world, if all capacity components are eventually revealed a given transcript can only correspond to a single sequence of sampled capacity components. On the other hand, if not all capacity components are eventually revealed, more than one sequence of sampled capacity components can give rise to the same transcript. In the ideal world, the effect of resampling is that there are even more sequences of sampled capacity components that will give rise to the same transcript. Nevertheless, as we show next, the probability of a good transcript occurring is unaffected by this resampling.

Fix a transcript and consider the point where the first hidden sample Q_i gets revealed in the transcript. Further let $j-1$ be the number of sampled capacity components up to that point, i.e., $i < j$. Furthermore, let C_r denote the set C just before the value Q_r is sampled and added to C. Consider then the probability of the transcript up to, and including, the point where Q_i is revealed and augmented with the remaining hidden samples fixed to some value. Denote this *augmented transcript* up to point j as τ_j. In the real world, there exist only one sequence of capacity components that can yield τ_j and thus its probability is given by:

$$\mathsf{P}_{\text{real}}(\tau_j) = \left(\frac{1}{2^c - |\mathsf{C}_1|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_i|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_{j-1}|}\right).$$

On the other hand, in the ideal world, there are multiple ways that Q_i can be sampled and result in such a sequence. More specifically, there are $2^c - |\mathsf{C}_i|$ values that have not yet been assigned and Q_i can take any of these values, except for the $|\mathsf{C}_j| - |\mathsf{C}_i| - 1$ values that are assigned in the interval right after the sampling of Q_i and before its resampling. Thus there are $2^c - |\mathsf{C}_i| - (|\mathsf{C}_j| - |\mathsf{C}_i| - 1)$ elements that Q_i can take. Then when resampling Q_i there are $2^c - |\mathsf{C}_j| + 1$ values to sample from since its pre-assigned value is replaced back. This probability is given by:

$$\mathsf{P}_{\text{ideal}}(\tau_j) = \left(\frac{1}{2^c - |\mathsf{C}_1|}\right) \cdots \left(\frac{2^c - |\mathsf{C}_j| + 1}{2^c - |\mathsf{C}_i|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_j| + 1}\right),$$

and canceling terms yields:

$$= \left(\frac{1}{2^c - |\mathsf{C}_1|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_i|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_{j-1}|}\right).$$

Thus the probability of τ_j is identical in both cases. Furthermore the set C_j corresponding to τ_j is identical in both worlds.

We can now extend this argument to the next point $j' > j$ in the transcript where a hidden sample $Q_{i'}$ gets revealed in the transcript. Let $\bar{\tau}_{j'}$ denote the transcript augmented with the hidden samples fixed to some value up to, *but excluding*, the point where $Q_{i'}$ is revealed. Then, using the above result it follows that

$$\begin{aligned}\mathsf{P}_{\text{ideal}}(\bar{\tau}_{j'}) &= \mathsf{P}_{\text{ideal}}(\tau_j) \cdot \left(\frac{1}{2^c - |\mathsf{C}_j|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_{j'-1}|}\right) \\ &= \left(\frac{1}{2^c - |\mathsf{C}_1|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_{j-1}|}\right) \left(\frac{1}{2^c - |\mathsf{C}_j|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_{j'-1}|}\right).\end{aligned}$$

The probability of $\tau_{j'}$ is obtained by summing $\mathsf{P}_{\text{ideal}}(\bar{\tau}_{j'})$ over all possible values of $Q_{i'}$ which are consistent with $\tau_{j'}$ and multiplying by the probability that $Q_{i'}$ matches the right value on resampling. This yields

$$\mathsf{P}_{\text{ideal}}(\tau_{j'}) = \left(\frac{1}{2^c - |\mathsf{C}_1|}\right) \cdots \left(\frac{2^c - |\mathsf{C}_{j'}| + 1}{2^c - |\mathsf{C}_{i'}|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_{j'}| + 1}\right),$$

and cancelling terms we obtain

$$= \left(\frac{1}{2^c - |\mathsf{C}_1|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_{i'}|}\right) \cdots \left(\frac{1}{2^c - |\mathsf{C}_{j'-1}|}\right) = \mathsf{P}_{\text{real}}(\tau_{j'}) \,.$$

This argument can be repeated iteratively until the last resampled capacity component in the transcript to show that the probabilities of the full augmented transcripts is the same in both worlds. It then follows that the probability of the unmodified transcripts, obtained by summing over all consistent values of the remaining hidden capacity components, is also the same in both worlds, i.e., $\mathsf{P}_{\text{real}}(\tau) = \mathsf{P}_{\text{ideal}}(\tau)$.

□

Game $\boxed{\mathbf{G6}}$ and G6. In **G**6 we now modify $\mathsf{Sim_p}$ to sample from the (larger) set $\{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{L_p})$, when sampling capacity components, instead of $\{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{C_t})$. This further removes another dependency that $\mathsf{Sim_p}$ has on $\mathsf{Sim_c}$. In this game hop, our starting point is Game $\boxed{\mathbf{G6}}$ which is itself based on Game $\boxed{\mathbf{G5}}$. The only changes from $\boxed{\mathbf{G5}}$ to $\boxed{\mathbf{G6}}$ are the shaded lines of code that appears in Fig. 21. Namely, we first sample from $\{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{L_p})$ and if the flag bad is set we discard it and sample once more from the original set $\{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{C_t})$ or $(\{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{C_t})) \cup \{W_C\}$. Note that this is equivalent to sampling from the original set directly, and thus

$$\Pr\left[\boxed{\mathbf{G6}} = 1\right] = \Pr\left[\boxed{\mathbf{G5}} = 1\right] .$$

Then in Game **G**6 we obtain the desired functionality by dropping the boxed lines of code. We account for this modifications by using the fact that Games $\boxed{\mathbf{G6}}$ and **G**6 are identical until bad. This is formalised in Lemma B.3. Note that unlike prior game hops, here we transition from $\boxed{\mathbf{G5}}$ to $\boxed{\mathbf{G6}}$ and then to Game **G**6.

Lemma B.3 *Let q_p and q_c denote the number of queries made by $\mathscr{D}$ to its permutation and construction oracles, respectively, and let c denote the capacity. Under the condition that $q_p < 2^{c-2}$ it holds that*

$$|\Pr[\mathbf{G}6 = 1] - \Pr\left[\boxed{\mathbf{G6}} = 1\right]| \leq \frac{2q_pq_c}{2^c} \,.$$

Proof. The flag bad is set whenever a randomly chosen capacity, sampled from $\{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{L_p})$, is contained in $\mathsf{C_t} \setminus \mathsf{L_p}$. This can happen at four places in the code of $\mathsf{Sim_p}$, but they are mutually exclusive. Hence we only need to account for its probability once for each query. Consider the i-th query to $\mathsf{Sim_p}$ and the case that bad gets set. Note that elements in $\mathsf{C_d} \cup \mathsf{L_p}$ are only added by $\mathsf{Sim_p}$ and each query adds at most two values (corresponding to the input and output of a query). On the other hand elements in $\mathsf{C_t} \setminus \mathsf{L_p}$ are added solely by $\mathsf{Sim_c}$ and each query adds only one element to this set. Note also that IV_C is not contained in this set either. Hence V_C is chosen uniformly from a set of size at least $2^c - |\mathsf{C_d} \cup \mathsf{L_p}|$, and bad is set if it matches one of the elements in $\mathsf{C_t} \setminus \mathsf{L_p}$. Applying the union bound and summing over all q_p queries we get

$$\Pr[\mathsf{bad}] \leq \sum_{i=1}^{q_p} \frac{|\mathsf{C_t} \setminus \mathsf{L_p}|}{2^c - |\mathsf{C_d} \cup \mathsf{L_p}|} \leq \frac{q_cq_p}{2^c - 2q_p - 1} \leq \frac{q_cq_p}{2^{c-1}},$$

where the last inequality follows from the assumption that $q_p < 2^{c-2}$. □

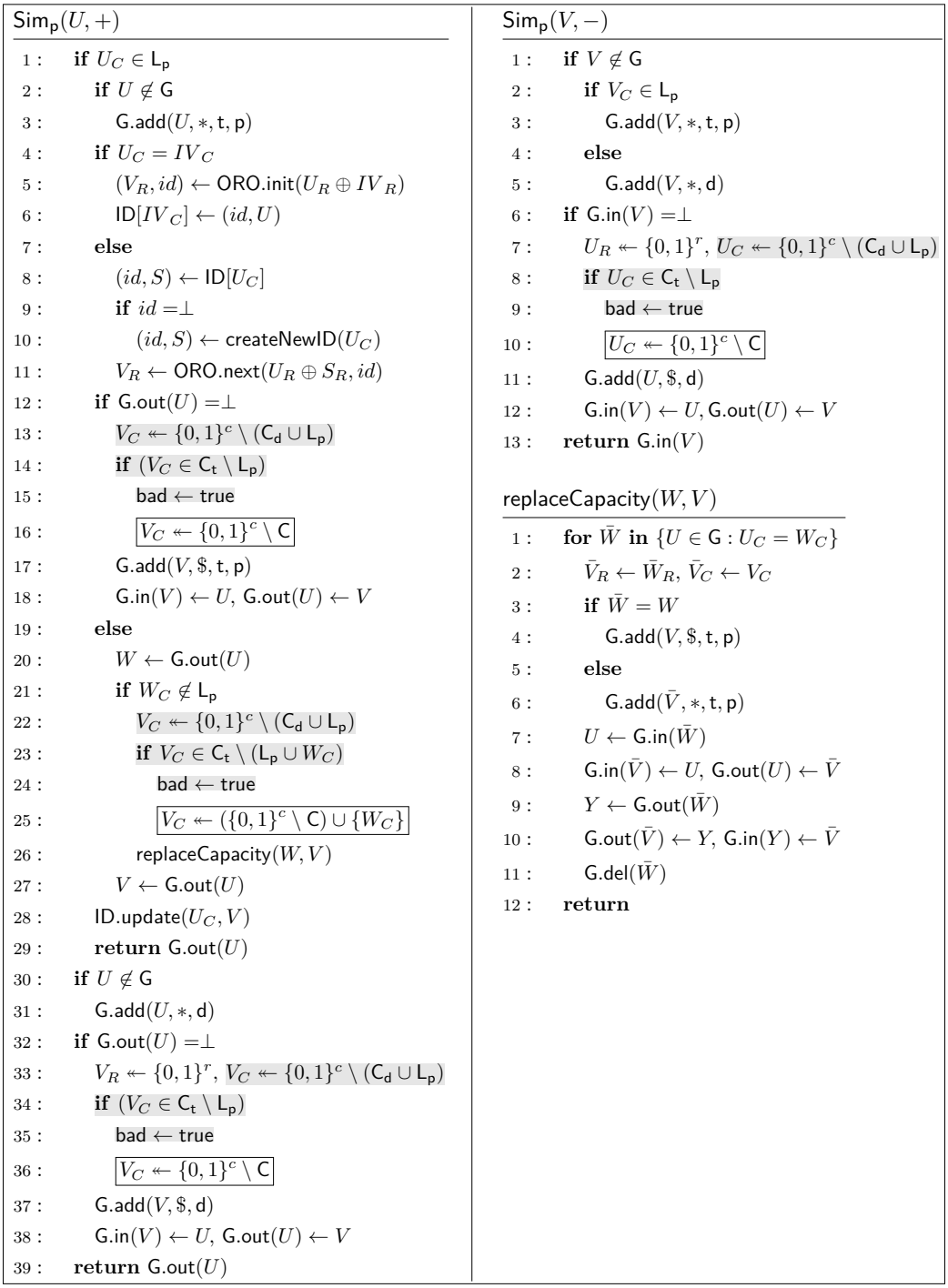

Figure 21: Description of $\mathsf{Sim_p}$ in games $\boxed{\mathbf{G}6}$ and $\mathbf{G}6$. Game $\boxed{\mathbf{G}6}$ includes the boxed code whereas $\mathbf{G}6$ does not.

Game G7. Having sufficiently decoupled $\mathsf{Sim_p}$ and $\mathsf{Sim_c}$, we are finally ready to transition to **G**7 which corresponds to the ideal world as defined by the indifferentiability notion. Here we replace the construction with the ORO and let $\mathsf{Sim_p}$ be the simulator Sim defined in Fig. 22. First note that already in **G**6 all of the distinguisher's queries to the construction oracle are being handled indirectly by the online random oracle. For every query $\mathsf{Sim_c}$ would recover a session equivalent to the one referenced by the distinguisher, obtain the rate part from the ORO, combine it with a locally sampled capacity component and forward it to the construction. In turn the construction, i.e., the Duplex, would strip out the capacity component and return the rate part that was originally computed by the ORO. Put simply, in **G**6, the combined effect of $\mathsf{Sim_c}$ and the Duplex is equivalent to a wire that forwards queries back and forth between itself and the ORO. However there is still one side effect of $\mathsf{Sim_c}$ that affects the operation of $\mathsf{Sim_p}$, which is that it adds nodes into the data structure G. This affects the operation of $\mathsf{Sim_p}$ when checking whether a node already exists in G. The resampling procedure ensures that the operation of $\mathsf{Sim_p}$ is the same whether the test on Line 12 is satisfied or not. Then the only way that $\mathsf{Sim_p}$ will behave differently in **G**7 than in **G**6 is when the adversary queries a node with a capacity component in $\mathsf{C_t} \setminus \mathsf{L_p}$ since these are the only nodes that will no longer be present in **G**7. Thus the two games are indistinguishable as long as no such query is made. This event corresponds to the bad event defined in $\boxed{\mathbf{G}5}$ and using a similar argument it follows that:

$$|\Pr[\mathbf{G}7 = 1] - \Pr[\mathbf{G}6 = 1]| \leq \frac{2q_c q_p}{2^c}.$$

Collecting the probabilities in all game hops yields the overall bound for the theorem.

B.2 Adaptations for General Encoding Functions

In this section, we discuss how to generalise the theorem to more general, round-dependent encoding functions. The more general approach is to allow the rate part to be arbitrarily distributed among the b bits, and to allow more complex functions on the rate part for both encode and decode. While the shuffling of the parts needs to be identical for each round, our proof supports round-dependent modifications on the rate.

More precisely, to define supported encoding functions $\mathsf{encode}_i : \{0,1\}^r \to \{0,1\}^b$ and decoding functions $\mathsf{decode}_i : \{0,1\}^b \to \{0,1\}^r$, we consider bit shuffles $\mathsf{Sh} : \{0,1\}^b \to \{0,1\}^b$ which permute the bits at a fixed set of positions. Then we define the encoding function for each round to be of the form $\mathsf{encode}_i(X) = \mathsf{Sh}(\pi_i(X) \parallel 0^c)$ where π_i is some round-dependent bijection, e.g., $\pi_i(X) = X \oplus 01^{r-1}$. For the decoding function we assume that $\mathsf{decode}_i(Y) = \tau_i(\lfloor \mathsf{Sh}^{-1}(Y) \rfloor_r)$, i.e., decode applies the inverse shuffle Sh^{-1} to Sh, then truncates all but the first r bits, and finally invokes a bijective function τ_i on the bits. Note that the round functions use the same bit re-arrangement Sh in all rounds, such that they affect the same bit positions in each round. In compliance with before, we call the positions to which Sh maps the extra 0-bits 0^c the *capacity* part, and the other r bits the *rate* part. We call such encoding and decoding functions *admissible*.

With such admissible functions the general indifferentiability theorem reads:

Theorem B.4 *Let* Dup *be the Duplex construction described in Fig. 1 with admissible encoding and decoding functions, composed from a random permutation* p *over b-bit strings, and let c denote its capacity. Further, let* ORO *be the online random oracle with rate $r = b - c$. Then there exists a simulator* Sim *such that for every distinguisher $\mathscr{D}$ making q_p queries to its permutation oracle and q_c queries to its construction oracle, where $q_p, q_c < 2^{c-2}$, its advantage in differentiating* Dup *from* ORO *is bounded by*

$$\mathbf{Adv}^{\mathit{indiff}}_{\mathsf{Dup},\mathsf{p},\mathsf{ORO},\mathsf{Sim}}(\mathscr{D}) \leq \frac{(q_p + q_c + 1)^2 + 6q_p q_c}{2^c}.$$

$\mathsf{Sim}(U,+)$

1 : **if** $U_C \in \mathsf{L_p}$
2 : **if** $U \notin \mathsf{G}$
3 : $\mathsf{G.add}(U,*,\mathsf{t},\mathsf{p})$
4 : **if** $U_C = IV_C$
5 : $(V_R, id) \leftarrow \mathsf{ORO.init}(U_R \oplus IV_R)$
6 : $\mathsf{ID}[IV_C] \leftarrow (id, U)$
7 : **else**
8 : $(id, S) \leftarrow \mathsf{ID}[U_C]$
9 : **if** $id = \perp$
10 : $(id, S) \leftarrow \mathsf{createNewID}(U_C)$
11 : $V_R \leftarrow \mathsf{ORO.next}(U_R \oplus S_R, id)$
12 : **if** $\mathsf{G.out}(U) = \perp$
13 : $V_C \twoheadleftarrow \{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{L_p})$
14 : $\mathsf{G.add}(V,\$,\mathsf{t},\mathsf{p})$
15 : $\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
16 : **else**
17 : $V \leftarrow \mathsf{G.out}(U)$
18 : $\mathsf{ID.update}(U_C, V)$
19 : **return** $\mathsf{G.out}(U)$
20 : **if** $U \notin \mathsf{G}$
21 : $\mathsf{G.add}(U,*,\mathsf{d})$
22 : **if** $\mathsf{G.out}(U) = \perp$
23 : $V_R \twoheadleftarrow \{0,1\}^r$, $V_C \twoheadleftarrow \{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{L_p})$
24 : $\mathsf{G.add}(V,\$,\mathsf{d})$
25 : $\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
26 : **return** $\mathsf{G.out}(U)$

$\mathsf{Sim}(V,-)$

1 : **if** $V \notin \mathsf{G}$
2 : **if** $V_C \in \mathsf{L_p}$
3 : $\mathsf{G.add}(V,*,\mathsf{t},\mathsf{p})$
4 : **else**
5 : $\mathsf{G.add}(V,*,\mathsf{d})$
6 : **if** $\mathsf{G.in}(V) = \perp$
7 : $U_R \twoheadleftarrow \{0,1\}^r$, $U_C \twoheadleftarrow \{0,1\}^c \setminus (\mathsf{C_d} \cup \mathsf{L_p})$
8 : $\mathsf{G.add}(U,\$,\mathsf{d})$
9 : $\mathsf{G.in}(V) \leftarrow U$, $\mathsf{G.out}(U) \leftarrow V$
10 : **return** $\mathsf{G.in}(V)$

$\mathsf{createNewID}(\bar{V}_C)$

1 : $i \leftarrow 0$
2 : $S \leftarrow \mathsf{C.rep}(\bar{V}_C)$, $\bar{V} \leftarrow S$
3 : **while** $\bar{V} \neq IV$ // executes at least once
4 : $\bar{U} \leftarrow \mathsf{G.in}(\bar{V})$
5 : $\bar{V} \leftarrow \mathsf{C.rep}(\bar{U}_C)$
6 : $X_i \leftarrow \bar{V}_R \oplus \bar{U}_R$
7 : $i \leftarrow i + 1$
8 : $i \leftarrow i - 1$
9 : $(Z, id) \leftarrow \mathsf{ORO.init}(X_i)$
10 : **while** $i > 0$
11 : $i \leftarrow i - 1$
12 : $Z \leftarrow \mathsf{ORO.next}(X_i, id)$
13 : $\mathsf{ID}[S_C] \leftarrow (id, S)$
14 : **return** (id, S)

Figure 22: Final simulator $\mathsf{Sim} = \mathsf{Sim_p}$ for game $\mathbf{G7}$ (ideal game).

The simulator Sim *is described in Fig. 23. Let* $(r \cdot \ell)$ *be a bound on the maximum number of bits queried in an* ORO *session. Then for each of the* q_p *queries it receives,* Sim *runs in time proportional to* ℓ *and makes at most* ℓ *calls to the* ORO.

Proof. We only explain the necessary changes in the proof of the simpler case presented in Section B.1. Recall that in the more general case we have $\mathsf{encode}_i(X_i) = \mathsf{Sh}(\pi_i(X_i) \parallel 0^c)$ and $\mathsf{decode}_i(Y) = \tau_i(\lfloor \mathsf{Sh}^{-1}(Y) \rfloor_r)$ for round-dependent bijections π_i, τ_i, whereas before we assumed that the shuffling is the identity function and that $\pi_i(X) = \tau_i(X) = X$ in each round. We first describe the changes required to accommodate the shuffling, and then those needed to handle the round-dependent bijections π_i, τ_i.

The rate part in the simpler case were the leading r bits and the capacity component the remaining c bits. In the more general case, these parts may be distributed among the b-bit string but we can still identify the parts via the shuffle function Sh, and this is independent of the round number. Hence, when speaking of the capacity component U_C of a node we think of the bits at the corresponding positions. We can also easily lift our assignment notions for the two parts from the easier case. That is, when assigning or sampling the rate part or capacity component, written $U_R \leftarrow s$ for $s \in \{0,1\}^r$ or $U_C \twoheadleftarrow \{0,1\}^c$, it is understood that one only sets the bits in the positions corresponding to the part in question and leaves the bits in the other part unchanged. By this we can for example re-assign one part without affecting the

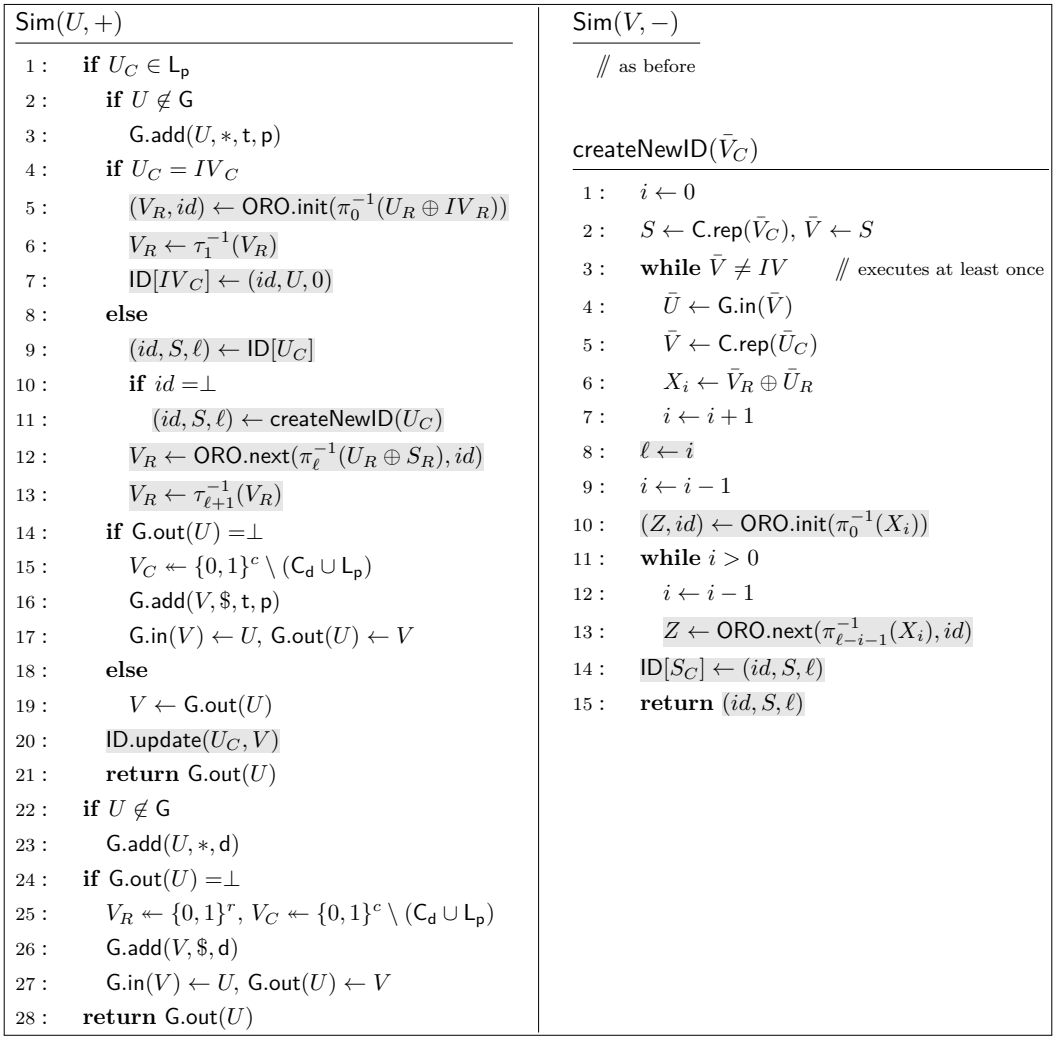

Figure 23: Final simulator Sim for game **G7** (ideal game) for round-dependent encoding functions.

other part. All statements about the two parts in the previous proof thus remain valid when one uses this implicit shuffling underneath.

Next, we justify which modifications to the proof are required to allow for round dependent bijections π_i, τ_i applied on the rate part. Without these changes, the arguments in the steps that involve the ORO no longer apply. We first explain the source of the problem and then outline the necessary modifications. In our proof, the graph G records the inputs and outputs to the permutation oracle; because the duplex construction is built from the permutation, queries to the construction oracle also induce vertices and edges in G. Since the ORO represents the ideal-world construction oracle, via a sequence of games, we gradually transform the real duplex construction into the ORO. Our current introduction of ORO in game **G**4, however, ignores the shifts introduced by π_i, τ_i between construction-level inputs/outputs and the corresponding permutation inputs/outputs represented in G. In particular, in the final hop from **G**6 to **G**7, we can no longer replace the composition (Duplex, $\mathsf{Sim_c}$, ORO) by the ORO alone, because the duplex applies π_i, τ_i to the ORO inputs/outputs forwarded by $\mathsf{Sim_c}$. Consequently, the mapping between construction-oracle paths and the graph G is misaligned and must be adjusted to reflect π_i, τ_i, i.e., the ORO inputs and outputs in the current proof must be shifted accordingly. The required changes begin at

game **G**4, where the ORO and the sub procedure createNewID are first introduced.

To support admissible round-dependent encoding functions, we need to make some changes regarding the steps where we involve ORO iterations, and where we derive its actual input. In the simpler case, we could derive the input value X_i of the ORO by the exclusive-or of the rate part of the previous output node and the input node. Here we need to take π_i into account, and apply its inverse to derive at the actual inputs. We let createNewID return the path length ℓ (equal to the number of edges and thus the number of permutation invocations) such that the simulator knows which encoding to apply, and also store the length of the path now in the table ID. In particular, ID.update(U_C, V) now also increments the stored value for ℓ in the new entry. The resulting final simulator is given in Fig. 23. The other change is to apply the correct inverse to π_i before calling ORO in games **G**4 to **G**7. This results in Fig. 23, to applying π_i^{-1} for the right index i to $U_R \oplus IV_R$ (Line 5), to $U_R \oplus S_R$ (Line 12), and to $\bar{V}_R \oplus \bar{U}_R$ (Line 10 and 13) before calling ORO. Note that this implies that if we encode such a value $\pi_i^{-1}(X)$ again and add the state to it, then the rate part of this derived value corresponds exactly to the input of the ORO.

Finally, we also need to make a minor modification to the proof when using admissible round-dependent decoding functions. We need to apply the inverse of τ_i to the ORO outputs in **G**4 to **G**7 so that the actual construction query outputs match the ORO outputs. These changes result in the new lines 6 and 13 in the final simulator given in Fig. 23.

With all these changes, in the last game hop from **G**6 to **G**7, we can now replace the combination of (Duplex, $\mathsf{Sim_c}$, ORO) with only the ORO. □

C SpongeWrap Security Proofs

In this section, we prove all the theorems stated in Section 7 on the security of SpongeWrap. In Appendix C.1 and Appendix C.2, we prove the KDM-AEAD security of SpongeWrap in the ORO- and the p-model. In Appendix C.3 and Appendix C.4, we prove the RKA-AEAD and CMT-AEAD security of SpongeWrap in the ORO-model. We first start by giving some additional notation specific to SpongeWrap that we will use throughout the proofs.

Notation specific to SpongeWrap. For any path P, we use $[P]^K$ (resp. $[P]^N$) to denote the k-bit (resp. n-bit) substring of P corresponding to the key portion (resp. nonce portion) if P was the path generated during an encryption query of SpongeWrap, i.e.,

$$[P]^K = P[1, r-1] \| P[r+1, 2r-1] \| \cdots \| P[ar+1, k+a],$$
$$[P]^N = P[k+a+1, (a+1)r-1] \| \cdots \| P[br+1, k+n+b]$$

where $a, b \in \mathbb{N}$ are such that $a \cdot (r-1) < k \leq (a+1) \cdot (r-1)$ and $b \cdot (r-1) < k+n \leq (b+1) \cdot (r-1)$. If $|P| < k+a$, then $[P]^K = \varepsilon$ and if $|P| < k+n+b$ then $[P]^N = \varepsilon$. We say that a key K (resp. nonce N) hits a path P, if $[P]^K = K$ (resp. $[P]^N = N$).

For any path P, we denote by prefix(P) the set of paths that are a prefix of P and of length ar for some integer a, i.e.,

$$\text{prefix}(P) = \{P[1, ar] : 1 \leq a \leq \lfloor |P| / r \rfloor\}.$$

C.1 Proof of Theorem 7.2 (SpongeWrap KDM-AEAD Security Proof in the ORO-model)

As a first step of the proof, we will show how we can restrict the adversary to be orderly, meaning that its decryption queries are made only after its encryption queries. Then, we will use the H-coefficient technique to bound the advantage of this simpler orderly adversary.

Let $\mathcal{A}$ be a KDAE adversary initialising w keys, making q_e encryption queries, q_d decryption queries, q_o queries to the ORO and whose queries are such that the output length $\mathsf{ol}(\phi)$ of ϕ is constant. We will construct an adversary $\mathcal{B}$ that is orderly, using a similar technique as in [BH22, Proposition 4.2] but adapted to the KDAE game. Let $\mathcal{B}$ be a KDAE adversary, that runs $\mathcal{A}$. It answers $\mathcal{A}$'s encryption and ORO queries faithfully using its own oracles $\textsc{Enc}$ and ORO. For each decryption query of $\mathcal{A}$, the reduction $\mathcal{B}$ returns $\lightning$ to $\mathcal{A}$ if it is the forwarded output of a previous encryption query, otherwise, it saves the query in a list L and returns 0 to $\mathcal{A}$. When $\mathcal{A}$ terminates with its guess b', the adversary $\mathcal{B}$ queries all the elements in L to its decryption oracle $\textsc{Dec}$. If one of these decryption queries returns 1 or $\lightning$, then $\mathcal{B}$ returns 1, otherwise it returns b'. Note that $\mathcal{B}$ makes the same number of queries as $\mathcal{A}$ and is orderly, as it makes all its decryption queries after its encryption queries.

When $b = 1$, $\mathcal{B}$ simulates exactly the game to $\mathcal{A}$ until a decryption query of $\mathcal{A}$ would return 1. If this happens, then $\mathcal{B}$ returns 1. Thus $\Pr\left[\mathcal{A}^{\mathsf{KDAE}} \Rightarrow 1 \;\middle|\; b = 1\right] \leq \Pr\left[\mathcal{B}^{\mathsf{KDAE}} \Rightarrow 1 \;\middle|\; b = 1\right]$. Moreover, when $b = 0$, $\mathcal{B}$ simulates exactly the game to $\mathcal{A}$ and has the same answer as $\mathcal{A}$ unless one of the decryption queries of $\mathcal{B}$ returns $\lightning$. This bad event happens when $\mathcal{B}$ has a decryption query $\textsc{Dec}(j, N, A, C)$ in its list L and an encryption query $\textsc{Enc}(j, A, \phi)$ returns (N, C). For a fixed decryption query $\textsc{Dec}(j, N, A, C)$, the probability that an encryption query $\textsc{Enc}(j, A, \phi)$ returns (N, C) is bounded by $\frac{1}{2^{n+t}}$. As there are at most q_d decryption queries in L and at most q_e encryption queries, using the fundamental lemma of game playing, we obtain

$$\Pr\left[\mathcal{B}^{\mathsf{KDAE}} \Rightarrow 1 \;\middle|\; b = 0\right] \leq \Pr\left[\mathcal{A}^{\mathsf{KDAE}} \Rightarrow 1 \;\middle|\; b = 0\right] + \frac{q_d q_e}{2^{n+t}}.$$

Therefore,

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW[ORO]}}(\mathcal{A}) &= \Pr\left[\mathcal{A}^{\mathsf{KDAE}} \Rightarrow 1 | b = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{KDAE}} \Rightarrow 1 | b = 0\right] \\
&\leq \Pr\left[\mathcal{B}^{\mathsf{KDAE}} \Rightarrow 1 | b = 1\right] - \Pr\left[\mathcal{B}^{\mathsf{KDAE}} \Rightarrow 1 | b = 0\right] + \frac{q_d q_e}{2^{n+t}} \\
&\leq \mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW[ORO]}}(\mathcal{B}) + \frac{q_d}{2^t},
\end{aligned}$$

where in the last inequality, we assume $q_e \leq 2^n$.

We now only have to bound $\mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW[ORO]}}(\mathcal{B})$ for the simpler orderly adversary $\mathcal{B}$ by using the H-coefficient technique (Theorem A.1). To apply the theorem, we split the KDAE game into a real game $\mathsf{realKDAE}$ (Fig. 24) and an ideal game $\mathsf{idealKDAE}$ (Fig. 25) corresponding to the KDAE game when respectively $b = 1$ and $b = 0$. Thus $\mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW[ORO]}}(\mathcal{B}) = \Pr\left[\mathcal{B}^{\mathsf{realKDAE}} \Rightarrow 1\right] - \Pr\left[\mathcal{B}^{\mathsf{idealKDAE}} \Rightarrow 1\right]$. Moreover, to facilitate our proof, we modify these games to give the adversary more information through an additional oracle $\textsc{Reveal}$ described in Fig. 26 and Fig. 27, that it must query exactly once as its last query, right before returning its output. Since these augmented games are strictly stronger than the original ones, we can bound any adversary's KDAE advantage by bounding its distinguishing advantage with respect to the augmented games.

We start by specifying the format of transcripts with respect to the augmented games, then we define four sets of bad transcripts. Subsequently, we bound the H-coefficient over good transcripts and the probability of bad transcripts in the ideal world. Finally, plugging the two bounds into Theorem A.1 yields the desired result.

Transcripts. In the H-coefficient technique, we only need to consider attainable transcripts, i.e., ones with a probability strictly greater than 0 of occurring in the ideal world. Note that here we define transcripts slightly differently, as we include additional information beyond the input-output pairs corresponding to the adversary's queries. We define them this way to facilitate the classification of good and bad transcripts

proc INITIALIZE(w)

for $j = 1$ **to** w **do**
  $K_j \twoheadleftarrow \mathsf{K}$; $S_j \leftarrow \emptyset$
$\eta_e \leftarrow 0$; $\eta_d \leftarrow 0$

proc ENC(j, A, ϕ)

$(M, V_o) \leftarrow \phi^{\mathsf{ORO}}(K_1, \ldots, K_w)$
$N \twoheadleftarrow \{0,1\}^n$; $C \leftarrow \mathsf{E}(K_j, N, A, M)$
$S_j \leftarrow S_j \cup \{(N, A, C)\}$; $\eta_e \leftarrow \eta_e + 1$
$\overline{V}_e[\eta_e] \leftarrow (j, A, M, V_o, N)$
return (N, C)

proc DEC(j, N, A, C)

if $(N, A, C) \in S_j$, **return** $\lightning$
$M \leftarrow \mathsf{D}(K_j, N, A, C)$; $\eta_d \leftarrow \eta_d + 1$
$\overline{V}_d[\eta_d] \leftarrow (j, N, A, C)$
return $(M \neq \perp)$

proc ORO.INIT(X)

return $\mathsf{ORO.init}(X)$

proc ORO.NEXT(X, id)

return $\mathsf{ORO.next}(X, id)$

proc FINALIZE(b')

return b'

Figure 24: realKDAE Game.

proc INITIALIZE(w)

for $j = 1$ **to** w **do** $S_j \leftarrow \emptyset$
$\eta_e \leftarrow 0$; $\eta_d \leftarrow 0$

proc ENC(j, A, ϕ)

$N \twoheadleftarrow \{0,1\}^n$; $C \twoheadleftarrow \{0,1\}^{\mathsf{ol}(\phi)+t}$
$S_j \leftarrow S_j \cup \{(N, A, C)\}$; $\eta_e \leftarrow \eta_e + 1$
$\overline{V}_e[\eta_e] \leftarrow (j, A, \phi, N, C)$
return (N, C)

proc DEC(j, N, A, C)

if $(N, A, C) \in S_j$, **return** $\lightning$
$\eta_d \leftarrow \eta_d + 1$
$\overline{V}_d[\eta_d] \leftarrow (j, N, A, C)$
return 0

proc ORO.INIT(X)

return $\mathsf{ORO.init}(X)$

proc ORO.NEXT(X, id)

return $\mathsf{ORO.next}(X, id)$

proc FINALIZE(b')

return b'

Figure 25: idealKDAE Game.

and other aspects of the proof. A transcript τ of an adversary interacting with an augmented game consists of the following entries:

- **Revealed key entries:** (key, j, K_j).
  These entries correspond to the w keys returned as part of the REVEAL query output. In the real world, these keys are sampled during the INITIALIZE procedure, whereas in the ideal world, they correspond to values sampled during the REVEAL procedure, independently of all the previous queries.

- **ORO entries:** $(\mathsf{oro}, X, id, Z, P_o)$.
  When $X = P_o$, an entry $(\mathsf{oro}, X, id, Z, P_o)$ corresponds to a query $\mathsf{ORO.init}(X)$ with answer (Z, id), otherwise, it corresponds to a query $\mathsf{ORO.next}(X, id)$ with answer Z where P_o is the path associated with the query (i.e., $Z = \mathsf{Tab}[P_o]$). Note that the adversary knows P_o in both worlds as it is the concatenation of the values queried to ORO for the corresponding session identifier id. For simplicity and wlog, we restrict the ORO entries to values of X that are of length r.

- **Encryption entries:** $(\mathsf{enc}, j, A, \phi, N, C, M, V_o^{(\eta)}, V_m^{(\eta)})$.
  An entry $(\mathsf{enc}, j, A, \phi, N, C, M, V_o^{(\eta)}, V_m^{(\eta)})$ corresponds to an encryption query $\textsc{Enc}(j, A, \phi)$ with answer (N, C), where M is the message encrypted and generated by ϕ.

  It additionally includes two sets $V_o^{(\eta)}$ and $V_m^{(\eta)}$ that contain in the real world all the internal ORO calls made by the encryption algorithm in the corresponding encryption query, more specifically, all

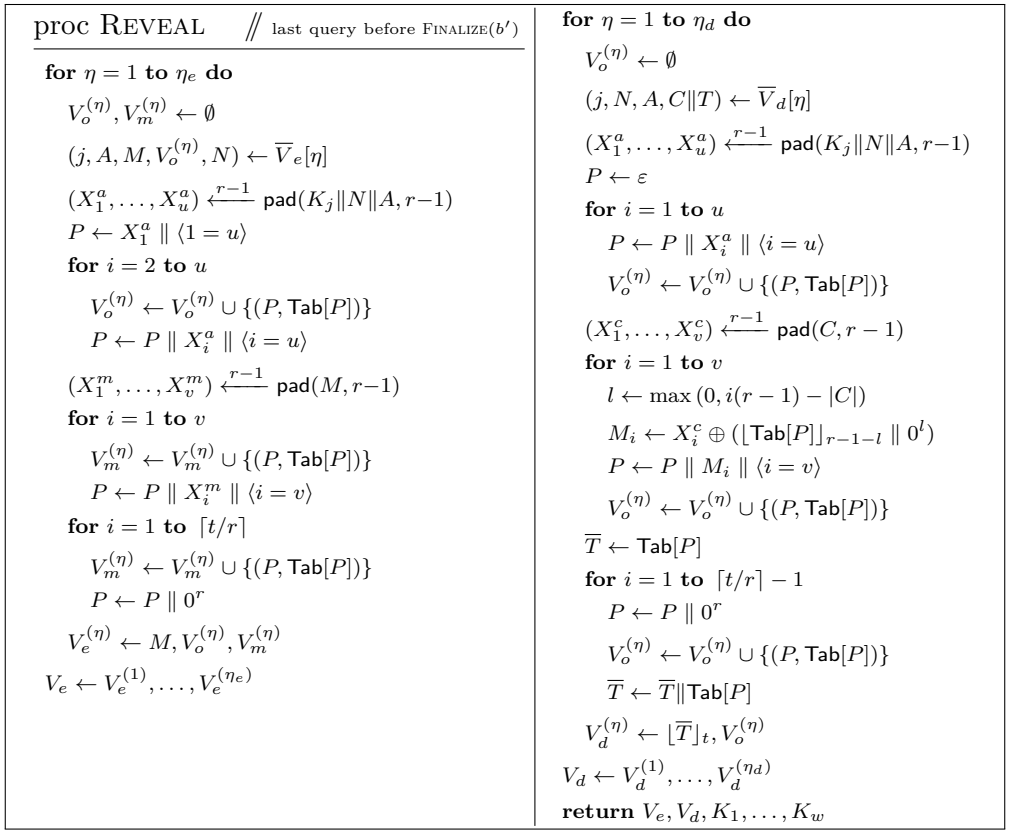

Figure 26: REVEAL oracle for the realKDAE Game.

the (path, output) pairs that were queried to ORO. The distinction between the two sets, whose generation is described in Fig. 26, is that the outputs in $V_m^{(\eta)}$ can be partially deduced by the adversary from the ciphertext either by xoring the plaintext and the ciphertext or directly through the tag, whereas the outputs in $V_o^{(\eta)}$ are generally unknown by the adversary. For simplicity in Fig. 24, we write $(M, V_o) \leftarrow \phi^{\mathsf{ORO}}(K_1, \ldots, K_w)$ to denote that when running $\phi^{\mathsf{ORO}}(K_1, \ldots, K_w)$ the message returned is M and the sets of (path, output) pairs queried to ORO by ϕ is V_o.

In the ideal world, the sets $V_o^{(\eta)}$ and $V_m^{(\eta)}$ as well as the message M are generated so that all good transcripts (defined below) have a similar probability strictly greater than 0 of occurring in the real world. We describe this generation process in Fig. 27 and Fig. 28. The core idea is to generate M and an ORO path with the same probability of being generated in the real world. As the values in $V_m^{(\eta)}$ can be partially deduced in the real world from the ciphertext and the plaintext, they need to be defined directly from M and $C\|T$. However, the inputs in $V_m^{(\eta)}$ are then not really queried to ORO in the ideal world, and the rest of the paths actually queried to ORO and stored in $V_o^{(\eta)}$ can become inconsistent. Meaning that we can obtain $(X, Z_1) \in V_m^{(\eta)}$ and $(X, Z_2) \in V_o^{(\eta)}$ such that $Z_1 \neq Z_2$, which would have a zero probability of happening in the real world. To circumvent this problem, we generate the ORO path through a slightly modified ORO procedure which we denote $\overline{\mathsf{ORO}}$ and describe in Fig. 28. The $\overline{\mathsf{ORO}}$ procedure responds faithfully using ORO to queries that are not a path in any $V_m^{(\eta)}$ and does not extend a path in any $V_m^{(\eta)}$. For queries that would correspond to a path in some $V_m^{(\eta)}$, the $\overline{\mathsf{ORO}}$ oracle responds in a consistent way by returning the first (already sampled) associated output. For queries that would extend a path in some $V_m^{(\eta)}$, the $\overline{\mathsf{ORO}}$ oracle

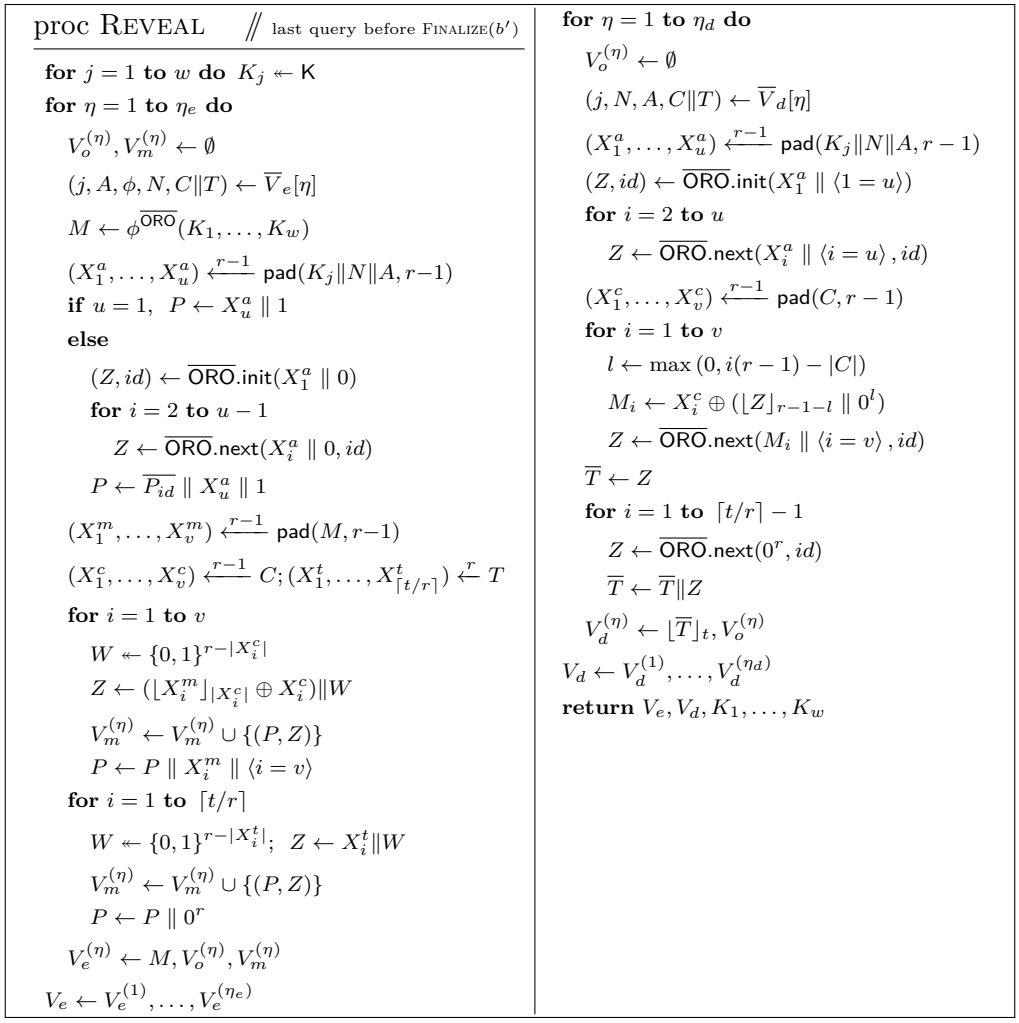

Figure 27: REVEAL oracle for the idealKDAE Game.

samples an associated output if it is the first time it is queried or responds in a consistent way if the path has already been queried (and sampled) before. Instead of queries to ORO, in the ideal world, we save the queries to $\overline{\mathsf{ORO}}$ in $V_o^{(\eta)}$.

Note that while in the transcript, for convenience, we include $M, V_o^{(\eta)}, V_m^{(\eta)}$ in the encryption entries, they are not actually returned to the adversary by the encryption oracle. In the augmented games, these values are only revealed to the adversary at the end in the REVEAL query. Also note that each ϕ and the adversary $\mathcal{B}$ are distinct algorithms, and as such, they cannot interfere with each other's ORO sessions.

- **Decryption entries:** $(\mathsf{dec}, j, N, A, C, 0, \overline{T}, V_o^{(\eta)})$.
  Entries of the type $(\mathsf{dec}, j, N, A, C, 0, \overline{T}, V_o^{(\eta)})$ correspond to decryption queries $\textsc{Dec}(j, N, A, C)$ with answer 0. For simplicity and wlog, we also restrict the decryption entries to the ones not forwarding the output from an encryption query, i.e., not returning $\lightning$, as the outputs of these queries are deterministic and the same in both worlds. Note that in the H-coefficient technique, we only need to be concerned with attainable transcripts, and in the ideal world, verification queries always return 0 as an answer. We also include in the decryption entries for the real world the associated ORO paths

$\overline{\mathsf{ORO}}.\mathsf{init}(X)$	$\overline{\mathsf{ORO}}.\mathsf{next}(X, id)$
if $\exists \overline{\eta}, \overline{Z}$ s.t. $(X, \overline{Z}) \in V_m^{(\overline{\eta})}$	**if** $\exists \overline{\eta}, \overline{Z}$ s.t. $(\overline{P_{id}} \parallel X, \overline{Z}) \in V_m^{(\overline{\eta})}$
$(Z, id) \leftarrow (\overline{Z}, \mathsf{GenID})$	$Z \leftarrow \overline{Z}$
else	**elseif** $\exists \overline{\eta}, \overline{P} \in \mathrm{prefix}(\overline{P_{id}})$ s.t. $(\overline{P}, \cdot) \in V_m^{(\overline{\eta})}$
$(Z, id) \leftarrow \mathsf{ORO}.\mathsf{init}(X)$	**if** $\exists \tilde{\eta}, \overline{Z}$ s.t. $(\overline{P_{id}} \parallel X, \overline{Z}) \in V_o^{(\tilde{\eta})}$
$\overline{P_{id}} \leftarrow X$	$Z \leftarrow \overline{Z}$
$V_o^{(\eta)} \leftarrow V_o^{(\eta)} \cup \{(\overline{P_{id}}, Z)\}$	**else**
return (Z, id)	$Z \twoheadleftarrow \{0,1\}^r$
	else
	$Z \leftarrow \mathsf{ORO}.\mathsf{next}(X, id)$
	$\overline{P_{id}} \leftarrow \overline{P_{id}} \parallel X$
	$V_o^{(\eta)} \leftarrow V_o^{(\eta)} \cup \{(\overline{P_{id}}, Z)\}$
	return Z

Figure 28: The $\overline{\mathsf{ORO}}$ procedures used by the REVEAL oracle in the idealKDAE Game (Fig. 27) and in the idealRKAE Game (Fig. 32).

and for the ideal world the associated $\overline{\mathsf{ORO}}$ paths queried, as a set $V_o^{(\eta)}$ of (path, output) pairs. This set $V_o^{(\eta)}$ is revealed to the adversary at the end in the REVEAL query. The string $\overline{T}$ represents the valid tag for C in the real world and is simply a concatenation of some of the outputs in $V_o^{(\eta)}$.

Bad Transcripts. For the H-coefficient technique, we need to lower bound the ratio of the real-world and ideal-world probabilities to a value close to one. We accomplish this by reducing the probability calculation of a transcript to a counting argument, by mainly counting the number of distinct ORO calls and random blocks generated. As already shortly described above, the induced calls in the sets $V_m^{(\eta)}$ are not queried to ORO in the ideal world and can generate inconsistent transcript with either a probability zero of happening in the real world or a different probability in each world. Similarly, with a probability 0 in the real world, the sets $V_o^{(\eta)}$ in a decryption query can correspond to a valid tag $\overline{T}$. To get a good ratio, we need to exclude and define those transcripts as bad. To facilitate our counting argument, we introduce the following sets and multi-sets (sets where elements can repeat):

$$S_1(\tau) = \bigcup_{(\mathsf{enc},j,A,\phi,N,C,M,V_o^{(\eta)},V_m^{(\eta)}) \in \tau} V_m^{(\eta)}.$$

The multi-set $S_1(\tau)$ contains the ORO (path, output) pairs that were induced through encryption queries and whose outputs can be partially deduced in the real world from the ciphertext and the plaintext. In the real world, they are generated by internal ORO queries, whereas in the ideal world, they are constructed from the corresponding ciphertext and the plaintext.

$$S_2(\tau) = \bigcup_{(\mathsf{enc},j,A,\phi,N,C,M,V_o^{(\eta)},V_m^{(\eta)}) \in \tau} V_o^{(\eta)}.$$

The set $S_2(\tau)$ contains the remaining ORO (path, output) pairs that were induced through encryption queries. In the real world, they are generated by internal ORO queries, whereas in the ideal world, they are generated through the internal $\overline{\mathsf{ORO}}$ procedure.

$$S_3(\tau) = \{(P_o, Z) \mid (\mathsf{oro}, X, id, Z, P_o) \in \tau\}.$$

The set $S_3(\tau)$ contains the ORO (path, output) pairs that were induced through direct queries to $\mathsf{ORO.init}$ or $\mathsf{ORO.next}$.

$$S_4(\tau) = \bigcup_{(\mathsf{dec},j,N,A,C,0,\overline{T},V_o^{(\eta)})\in\tau} V_o^{(\eta)}.$$

The set $S_4(\tau)$ contains the ORO (path, output) pairs induced through decryption queries. In the real world, they are generated by internal ORO queries, whereas in the ideal world, they are generated through the internal $\overline{\mathsf{ORO}}$ procedure.

$$S_5(\tau) = (S_2(\tau) \cup S_3(\tau) \cup S_4(\tau)) \setminus S_1(\tau).$$

The set $S_5(\tau)$ contains the ORO (path, output) pairs that were induced through all the queries and that are not in $S_1(\tau)$. As they are distinct from the elements in $S_1(\tau)$, they are sampled through the ORO or $\overline{\mathsf{ORO}}$ queries.

We define as bad, transcripts that result in a different number of blocks sampled in the real world compared to the ideal world and impossible transcripts in the real world. This can happen in the following ways:

Case 1 $(x_1, y_1) \in S_1(\tau)$ and $(x_2, y_2) \in S_1(\tau)$ where $x_1 = x_2$. In this case, in the real world, the same path is queried twice to ORO resulting in a single block sampled, while in the ideal world, two output blocks y_1, y_2 are randomly sampled. This case also encompasses the case where $x_1 = x_2$ and $y_1 \neq y_2$, which is impossible in the real world. From this case, we can define the following simplified bad transcript description:

$\mathsf{Bad_1}$**:** There is an entry $(\mathsf{enc}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, V_m^{(\eta)})$ and afterward a distinct entry $(\mathsf{enc}, \cdot, \cdot, \cdot, N, \cdot, \cdot, \cdot, \cdot)$ such that for $(P, \cdot) \in V_m^{(\eta)}$, $[P]^N = N$.

Case 2 $(x_1, y_1) \in S_1(\tau)$ and $(x_2, y_2) \in S_2(\tau)$ where $x_1 = x_2$. In this case, in the real world, the same path is queried twice to ORO resulting in a single block sampled, while in the ideal world, one output block is randomly sampled, and the second is returned by an internal call to $\overline{\mathsf{ORO}}$. This case also encompasses the case where $x_1 = x_2$ and $y_1 \neq y_2$, which is impossible in the real world. Note that in the ideal world, if the encryption query corresponding to the $\overline{\mathsf{ORO}}$ block y_2 happens after the encryption query corresponding to the randomly sampled block y_1, then the block returned by $\overline{\mathsf{ORO}}$ is y_1 and no sampling occurs in $\overline{\mathsf{ORO}}$. The inconsistency in the number of sampling (between the real and ideal world) happens only when the encryption query corresponding to the $\overline{\mathsf{ORO}}$ block y_2 is the first query. From this case, we can define the following simplified bad transcript description:

$\mathsf{Bad_2}$**:** There is an entry $(\mathsf{enc}, \cdot, \cdot, \phi, \cdot, \cdot, \cdot, V_o^{(\eta)}, \cdot)$ and afterward an entry $(\mathsf{enc}, \cdot, \cdot, \cdot, N, \cdot, \cdot, \cdot, \cdot)$ such that for $(P, \cdot) \in V_o^{(\eta)}$, $[P]^N = N$. When the two entries are the same, the condition $[P]^N = N$ should hold only for paths in $V_o^{(\eta)}$ called by the function ϕ associated to the entry.

Case 3 $(x_1, y_1) \in S_1(\tau)$ and $(x_2, y_2) \in S_3(\tau)$ where $x_1 = x_2$. In this case, in the real world, the same path is queried twice to ORO resulting in a single block sampled, while in the ideal world, one output block is randomly sampled, and the second is returned by ORO. This case also encompasses the case where $x_1 = x_2$ and $y_1 \neq y_2$, which is impossible in the real world. Note that when this case doesn't happen, direct queries to ORO and the internal calls to $\overline{\mathsf{ORO}}$ are never inconsistent between each other. From this case, we can define the following simplified bad transcript description:

$\mathsf{Bad_3}$**:** There are two entries $(\mathsf{oro}, X, id, Z, P_o)$ and (key, j, K_j) such that $[P_o]^K = K_j$.

Case 4 $(\mathsf{dec}, j, N, A, C \| T, 0, \overline{T}, V_o^{(\eta)}) \in \tau$ where $T = \overline{T}$. This case corresponds to an impossible transcript in the real world, as the verification query considered would have returned true. From this case, we can define the following bad transcript description:

Bad_4**:** There is an entry $(\mathsf{dec}, j, N, A, C \| T, 0, \overline{T}, V_o^{(\eta)})$ such that $T = \overline{T}$.

For $j \in \{1, \ldots, 4\}$, let Bad_j be the set of attainable transcripts satisfying the j-th bad transcript description defined above. Then the set of bad transcripts is defined as $\mathbb{T}_{\text{bad}} = \bigcup_{j=1}^{4} \mathsf{Bad}_j$.

Good Transcript Ratio. Attainable transcripts not in $\mathbb{T}_{\text{bad}}$ are called good transcripts, and the set of all good transcripts is denoted by $\mathbb{T}_{\text{good}}$. As we have excluded in $\mathbb{T}_{\text{bad}}$ all detrimental transcripts resulting in a weak H-coefficient ratio not close to one, we can now calculate this value more easily.

Let $\tau \in \mathbb{T}_{\text{good}}$ be a good transcript and let $\mathsf{P}_{\text{ideal}}(\tau)$, resp. $\mathsf{P}_{\text{real}}(\tau)$, be the probability that if we make the queries described in τ (in the same order), we receive in the ideal game, resp. the real game, the corresponding answers recorded in τ. We denote by $(M, V_o^{(\eta)}) \leftarrow \phi(K_1, \ldots, K_w, V_o^{(\eta)})$ the event that when running $\phi(K_1, \ldots, K_w)$ and responding deterministically to its ORO queries with the answers in the set $V_o^{(\eta)}$ of (path, output) pairs, its queries to ORO are the ones described in $V_o^{(\eta)}$ (in the same order), and the message returned is M.

In the ideal world, the revealed key and the blocks in $S_1(\tau)$ are sampled at random, independently from the calls saved in $S_5(\tau)$ that correspond to values sampled through ORO and $\overline{\mathsf{ORO}}$. Note that the values in $S_5(\tau)$ are constructed in a consistent way, meaning that each path in $S_5(\tau)$ is associated with a unique corresponding output. Moreover, the coins of the derivation functions ϕ queried during encryption are also independent from the coins of the system. Thus,

$$\mathsf{P}_{\text{ideal}}(\tau) \;=\; \frac{1}{2^{kw}} \cdot \prod_{i=1}^{|S_1(\tau)|} \frac{1}{2^r} \cdot \prod_{i=1}^{|S_5(\tau)|} \frac{1}{2^r} \cdot \prod_{(\mathsf{enc}, \cdot, \cdot, \phi, \cdot, \cdot, M, V_o^{(\eta)}, \cdot) \in \tau} \Pr\left[(M, V_o^{(\eta)}) \leftarrow \phi(K_1, \ldots, K_w, V_o^{(\eta)}) \right].$$

Note that for simplicity, in the above equation, we slightly abuse notation for $V_o^{(\eta)}$ to denote only the subset of (path, output) pairs queried by ϕ for the corresponding encryption query.

In the real world, as τ is a good transcript, all paths in $S_1(\tau)$ are distinct, and there are exactly $|S_1(\tau)| + |S_5(\tau)|$ distinct corresponding calls to ORO in τ. Similarly as in the ideal world, the revealed keys are sampled at random and the coins of the derivation functions ϕ queried during encryption are independent from the coins of the system. Thus $\mathsf{P}_{\text{real}}(\tau) = \mathsf{P}_{\text{ideal}}(\tau)$, which gives us the following H-coefficient:

$$\frac{\mathsf{P}_{\text{real}}(\tau)}{\mathsf{P}_{\text{ideal}}(\tau)} = 1. \tag{3}$$

Bad Transcript Probabilities. We now bound the probability that $\mathcal{T}_{\text{ideal}} \in \mathbb{T}_{\text{bad}}$, i.e., the probability that a transcript generated in the ideal world is bad.

We start by bounding the probability that $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_1 \cup \mathsf{Bad}_2$, with $\mathsf{Bad}_1 \cup \mathsf{Bad}_2$ being the set of attainable transcripts that contains an entry $(\mathsf{enc}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, V_o^{(\eta)}, V_m^{(\eta)})$ and afterward an entry $(\mathsf{enc}, \cdot, \cdot, \cdot, N, \cdot, \cdot, \cdot, \cdot)$ such that for $P \in V_o^{(\eta)} \cup V_m^{(\eta)}$, $[P]^N = N$. Thus $\mathcal{T}_{\text{ideal}}$ is in $\mathsf{Bad}_1 \cup \mathsf{Bad}_2$ if a nonce N sampled during an encryption query hits a path of a previous encryption query. While the paths in $V_o^{(\eta)}$ and $V_m^{(\eta)}$ are constructed during the REVEAL query, we perform a syntactical change to the $\mathsf{idealKDAE}$ game so that the paths in $V_o^{(\eta)}$ and $V_m^{(\eta)}$ are constructed during their corresponding encryption queries. This modification is indistinguishable for the adversary as the values in $V_o^{(\eta)}$ and $V_m^{(\eta)}$ are released only at the end of the

game. Thus whether they are constructed at the end or during the encryption queries leaves the game unchanged. In this equivalent game, we split the event $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_1 \cup \mathsf{Bad}_2$ into two cases. In the first case, the nonce hits a path called by a ϕ corresponding to a previous or current encryption query. For each encryption query $(\mathsf{enc}, \cdot, \cdot, \cdot, N, \cdot, \cdot, \cdot, \cdot)$, there are at most q_ϕ paths called by a ϕ corresponding to a previous or current encryption query. As the nonce is sampled uniformly at random during the encryption queries, summing over all encryption queries, the probability that a nonce hits one of the paths called by a ϕ in a previous encryption query is at most $\frac{q_e q_\phi}{2^n}$. In the remaining case, the nonce hits a path corresponding to a previous encryption query, but that is not a path called by a ϕ. This case is only possible when the nonce collides with the nonce of a previous encryption query. Thus, the probability of this case is bounded by $\frac{q_e^2}{2^n}$. Using a union bound, we obtain

$$\Pr[\,\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_1 \cup \mathsf{Bad}_2] \leq \frac{q_e \cdot (q_e + q_\phi)}{2^n}. \tag{4}$$

$\mathcal{T}_{\text{ideal}}$ is in Bad_3 if one of the w keys sampled during Reveal hits a path queried to ORO by the adversary. As the keys are sampled uniformly at random and independently from the at most q_o ORO queries, the probability that $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_3$ is bounded by $\frac{wq_o}{2^k}$.

$$\Pr[\,\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_3] \leq \frac{wq_o}{2^k}. \tag{5}$$

When $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_4$, it means that there is an entry $(\mathsf{dec}, j, N, A, C\|T, 0, \overline{T}, V_o^{(\eta)})$ such that $T = \overline{T}$. We will consider two cases, when the path corresponding to $\overline{T}$ has been sampled in a previous encryption query and when it is sampled during Reveal.

For any entry $(\mathsf{dec}, j, N, A, C\|T, 0, \overline{T}, V_o^{(\eta)})$, if the paths corresponding to $\overline{T}$ have been sampled before Reveal in a previous encryption query (i.e., are in a $V_m^{(\eta)}$), the only possibility is that there exists a previous query $\textsc{Enc}(j', A, \cdot)$ with answer $(N, C\|\overline{T})$ and $K_j = K_{j'}$. We split this event into two subcases, when $j \neq j'$ and when $j = j'$. For the first subcase $j \neq j'$, the probability of this event is bounded by the probability that there exists $j \neq j'$ such that $K_j = K_{j'}$. As the user keys are sampled at random, it is bounded by $\frac{w^2}{2^{k+1}}$. In the second subcase $j = j'$, as the adversary cannot forward the output of an encryption query to a decryption query, we must have that $\overline{T} \neq T$. Thus, the probability that there exists an entry $(\mathsf{dec}, j, N, A, C\|T, 0, \overline{T}, V_o^{(\eta)})$ such that $T = \overline{T}$ and the path corresponding to $\overline{T}$ has been sampled in a previous encryption query is bounded by $\frac{w^2}{2^{k+1}}$.

We can now consider the case when the path corresponding to $\overline{T}$ has not been sampled in a previous encryption query. If we additionally assume $\mathcal{T}_{\text{ideal}} \notin \mathsf{Bad}_3$, then $\overline{T}$ is entirely sampled during Reveal. Indeed, $\mathcal{T}_{\text{ideal}} \notin \mathsf{Bad}_3$ means that $\overline{T}$ is also not sampled during an ORO query as otherwise, the key corresponding to the decryption query would hit the path of the ORO query. Thus if for each decryption entry $(\mathsf{dec}, j, N, A, C\|T, 0, \overline{T}, V_o^{(\eta)})$, the value $\overline{T}$ is sampled uniformly at random and independently from T at the end during Reveal, the probability that $T = \overline{T}$ is 2^{-t}. Summing over all decryption entries, we obtain as upper bound for this case $\frac{q_d}{2^t}$.

Thus using a union bound,

$$\Pr[\,(\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_4) \wedge (\mathcal{T}_{\text{ideal}} \notin \mathsf{Bad}_3)] \leq \frac{q_d}{2^t} + \frac{w^2}{2^k}. \tag{6}$$

As $\mathbb{T}_{\text{bad}} = \bigcup_{j=1}^{4} \mathsf{Bad}_j$, using a union bound and substituting terms through equations (4)–(6), we have

$$\Pr[\,\mathcal{T}_{\text{ideal}} \in \mathbb{T}_{\text{bad}}] \leq \frac{q_e^2 + q_e q_\phi}{2^n} + \frac{wq_o + w^2}{2^k} + \frac{q_d}{2^t}. \tag{7}$$

Theorem 7.2 is obtained by combining equations (3) and (7) with Theorem A.1, where equation (3) yields ϵ_1 and equation (7) yields ϵ_2.

C.2 Proof of Theorem 7.3 (SpongeWrap KDM-AEAD Security Proof in the p-model)

The bound is obtained by viewing the adversary $\mathcal{A}$ as a sequence of adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ where $\mathcal{A}_1$ alone has access to the primitive p and $\mathcal{A}_2$ accesses the primitive only via the construction when issuing encryption and decryption queries. By assumption in the theorem, $\mathcal{A}_1$ makes at most q_p queries to the primitive and $\mathcal{A}_2$ makes at most q_e encryption queries and q_d decryption queries. Each such encryption/decryption query made by $\mathcal{A}_2$ results in at most ℓ construction calls to process the encryption/decryption query and in total at most q_ϕ construction calls made by all the key derivation functions.

We translate this adversary $\mathcal{A}$ in the p-model into an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ in the ORO-model where $\mathcal{B}_1$ interacts with the ORO and $\mathcal{B}_2$ issues the encryption/decryption queries issued by $\mathcal{A}_2$. The adversary $\mathcal{B}_1$ is the combination of the adversary $\mathcal{A}_1$ and the indifferentiability simulator given in Theorem 6.1. Then, we can apply the composition theorem given in Theorem 4.2 which states that $\mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW[Dup]}}(\mathcal{A})$ is bounded by $\mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW[Dup]}}(\mathcal{A}) \leq \mathbf{Adv}^{\mathsf{kdae}}_{\mathsf{nSW[ORO]}}(\mathcal{B}) + \mathbf{Adv}^{\mathsf{indiff}}_{\mathsf{Dup,p,ORO,Sim}}(\mathscr{D})$.

The bound on the first term on the RHS in the above inequality can be derived by applying Theorem 7.2 to obtain $\frac{q_e^2+q_eq_\phi}{2^n} + \frac{w\ell q_p+w^2}{2^k} + \frac{2q_d}{2^t}$. This is obtained by substitution of the appropriate values in the bound given by Theorem 7.2. It is immediate to see that the number of encryption/decryption queries q_e and q_d remain the same for $\mathcal{A}$ and $\mathcal{B}$. By assumption in the theorem, $\mathcal{A}$ leads to at most q_ϕ construction calls by the key derivation functions. Hence, the adversary $\mathcal{B}$ in the ORO-model also leads to the key derivation functions making at most q_ϕ construction calls. Further, since the adversary $\mathcal{B}_1$ is the combination of the adversary $\mathcal{A}_1$ and the indifferentiability simulator given in Theorem 6.1, q_o is simply the number of queries that the simulator makes to the ORO when it receives a primitive query from the adversary $\mathcal{A}_1$ and thus is bounded by $\ell \cdot q_p$. Substituting these values gives us the bound.

The second term on the RHS in the composition theorem given in Theorem 4.2 is obtained by substituting the appropriate q_p and q_c terms in the indifferentiability bound $\frac{(q_p+q_c+1)^2+6q_pq_c}{2^c}$ given by Theorem 6.1. The indifferentiability distinguisher here is a combination of the adversary $\mathcal{A}$ and the KDAE game. Thus, the number of queries made to the primitive is given by exactly q_p, the number of queries to the primitive that $\mathcal{A}_1$ makes. The number of queries to the construction q_c is given by the number of calls to the construction made during each encryption/decryption query. Thus, $q_c \leq \ell \cdot (q_e + q_d) + q_\phi$. Plugging these terms in Theorem 6.1 gives us $\frac{(q_p+\ell(q_e+q_d)+q_\phi+1)^2+6q_p(\ell(q_e+q_d)+q_\phi)}{2^c}$. Putting the first term and the second term of the RHS in Theorem 4.2 together, we obtain the bound.

C.3 Proof of Theorem 7.8 (SpongeWrap RKA-AEAD Security Proof)

The general structure of the proof will follow the one from Theorem 7.2 but adapted to the RKAE game. We will first restrict the adversary to be orderly, meaning that its decryption queries are made only after its encryption queries. Then, we will use the H-coefficient technique to bound the advantage of this simpler orderly adversary.

Let $\mathcal{A}$ be a RKAE adversary making q_e encryption queries, q_d decryption queries, q_o queries to the ORO and whose queries are such that $\varphi^{\mathsf{ORO}} \in \Phi^{\mathsf{ORO}}$ and a pair $(\varphi^{\mathsf{ORO}}, N)$ never repeats across its encryption queries. We will construct an adversary $\mathcal{B}$ that is orderly, using a similar technique as in [BH22, Proposition 4.2] but adapted to the RKAE game. Let $\mathcal{B}$ be a RKAE adversary that runs $\mathcal{A}$. It answers $\mathcal{A}$'s encryption and ORO queries faithfully using its own oracles Enc and ORO. For each decryption query of $\mathcal{A}$, the reduction $\mathcal{B}$ returns $\lightning$ to $\mathcal{A}$ if it is the forwarded output of a previous encryption query, otherwise, it saves the query in a list L and returns 0 to $\mathcal{A}$. When $\mathcal{A}$ terminates with its guess b', the adversary $\mathcal{B}$ queries all the elements in L to its decryption oracle Dec. If one of these decryption queries returns a message M different from $\perp$, then $\mathcal{B}$ returns 1, otherwise it returns b'. Note that $\mathcal{B}$ makes the same number of queries as $\mathcal{A}$ and is orderly, as it makes all its decryption queries after its encryption queries.

When $b = 1$, $\mathcal{B}$ simulates exactly the game to $\mathcal{A}$ until a decryption query of $\mathcal{A}$ would return a message M different from $\perp$. If this happens, then $\mathcal{B}$ returns 1. Thus

$$\Pr\left[\mathcal{A}^{\mathsf{RKAE}} \Rightarrow 1 \;\middle|\; b = 1\right] \leq \Pr\left[\mathcal{B}^{\mathsf{RKAE}} \Rightarrow 1 \;\middle|\; b = 1\right].$$

Moreover, when $b = 0$, $\mathcal{B}$ simulates exactly the game to $\mathcal{A}$ and has the same answer as $\mathcal{A}$ unless one of the decryption queries of $\mathcal{B}$ returns $\lightning$. This bad event happens when $\mathcal{B}$ has a decryption query $\textsc{Dec}(\varphi, N, A, C)$ in its list L and an encryption query $\textsc{Enc}(\varphi, N, A, M)$ returns C. For a fixed decryption query $\textsc{Dec}(\varphi, N, A, C)$, the probability that there exists an encryption query $\textsc{Enc}(\varphi, N, A, M)$ that returns C, is bounded by $\frac{1}{2^t}$. This is because there are at most $2^{|C|-t}$ possible plaintexts M and the probability that the corresponding encryption query $\textsc{Enc}(\varphi, N, A, M)$ returns C is $2^{-|C|}$. As there are at most q_d decryption queries, using the fundamental lemma of game playing, we obtain

$$\Pr\left[\mathcal{B}^{\mathsf{RKAE}} \Rightarrow 1 \;\middle|\; b = 0\right] \leq \Pr\left[\mathcal{A}^{\mathsf{RKAE}} \Rightarrow 1 \;\middle|\; b = 0\right] + \frac{q_d}{2^t}.$$

Therefore,

$$\begin{aligned}
\mathbf{Adv}^{\mathsf{rkae}}_{\mathsf{nSW[ORO]}}\left(\mathcal{A}, \Phi^{\mathsf{ORO}}\right) &= \Pr\left[\mathcal{A}^{\mathsf{RKAE}} \Rightarrow 1 | b = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{RKAE}} \Rightarrow 1 | b = 0\right] \\
&\leq \Pr\left[\mathcal{B}^{\mathsf{RKAE}} \Rightarrow 1 | b = 1\right] - \Pr\left[\mathcal{B}^{\mathsf{RKAE}} \Rightarrow 1 | b = 0\right] + \frac{q_d}{2^t} \\
&\leq \mathbf{Adv}^{\mathsf{rkae}}_{\mathsf{nSW[ORO]}}\left(\mathcal{B}, \Phi^{\mathsf{ORO}}\right) + \frac{q_d}{2^t}.
\end{aligned}$$

We now only have to bound $\mathbf{Adv}^{\mathsf{rkae}}_{\mathsf{nSW[ORO]}}\left(\mathcal{B}, \Phi^{\mathsf{ORO}}\right)$ for the simpler orderly adversary $\mathcal{B}$ by using the H-coefficient technique (Theorem A.1). To apply the theorem, we split the RKAE game into a real game $\mathsf{realRKAE}$ (Fig. 29) and an ideal game $\mathsf{idealRKAE}$ (Fig. 30) corresponding to the RKAE game when respectively $b = 1$ and $b = 0$. Thus $\mathbf{Adv}^{\mathsf{rkae}}_{\mathsf{nSW[ORO]}}\left(\mathcal{B}, \Phi^{\mathsf{ORO}}\right) = \Pr\left[\mathcal{B}^{\mathsf{realRKAE}} \Rightarrow 1\right] - \Pr\left[\mathcal{B}^{\mathsf{idealRKAE}} \Rightarrow 1\right]$. Moreover, to facilitate our proof, we modify these games to give the adversary more information through an additional oracle $\textsc{Reveal}$ described in Fig. 31 and Fig. 32, that it must query exactly once as its last query, right before returning its output. Since these augmented games are strictly stronger than the original ones, we can bound any adversary's RKAE advantage by bounding its distinguishing advantage with respect to the augmented games.

We start by specifying the format of transcripts with respect to the augmented games, then we define six sets of bad transcripts. Subsequently, we bound the H-coefficient over good transcripts and the probability of bad transcripts in the ideal world. Finally, plugging the two bounds into Theorem A.1 yields the desired result.

Transcripts. In the H-coefficient technique, we only need to consider attainable transcripts, i.e., ones with a probability strictly greater than 0 of occurring in the ideal world. We define transcripts slightly differently here, as we include additional information beyond the input-output pairs corresponding to the adversary's queries. We define them this way to facilitate the classification of good and bad transcripts and other aspects of the proof. A transcript τ of an adversary interacting with an augmented game consists of the following entries:

- **Revealed key entry:** (key, K).
  This entry corresponds to the key K returned as part of the $\textsc{Reveal}$ query output. In the real world, this key is sampled during the $\textsc{Initialize}$ procedure, whereas in the ideal world, it corresponds to a value sampled during the $\textsc{Reveal}$ procedure, independently of all the previous queries.

proc INITIALIZE

$K \twoheadleftarrow \mathsf{K}$; $S \leftarrow \emptyset$

$\eta_e \leftarrow 0$; $\eta_d \leftarrow 0$

proc ENC(φ, N, A, M)

$(K_e, V_o) \leftarrow \varphi^{\mathsf{ORO}}(K)$

$C \leftarrow \mathsf{E}(K_e, N, A, M)$; $\eta_e \leftarrow \eta_e + 1$

$S \leftarrow S \cup \{(\varphi, N, A, C)\}$

$\overline{V}_e[\eta_e] \leftarrow (K_e, N, A, M, V_o)$

return C

proc DEC(φ, N, A, C)

if $(\varphi, N, A, C) \in S$, **return** $\lightning$

$(K_d, V_o) \leftarrow \varphi^{\mathsf{ORO}}(K)$

$M \leftarrow \mathsf{D}(K_d, N, A, C)$; $\eta_d \leftarrow \eta_d + 1$

$\overline{V}_d[\eta_d] \leftarrow (K_d, N, A, C, V_o)$

return M

proc ORO.INIT(X)

return $\mathsf{ORO.init}(X)$

proc ORO.NEXT(X, id)

return $\mathsf{ORO.next}(X, id)$

proc FINALIZE(b')

return b'

Figure 29: realRKAE Game.

proc INITIALIZE

$S \leftarrow \emptyset$

$\eta_e \leftarrow 0$; $\eta_d \leftarrow 0$

proc ENC(φ, N, A, M)

$C \twoheadleftarrow \{0,1\}^{|M|+t}$; $\eta_e \leftarrow \eta_e + 1$

$S \leftarrow S \cup \{(\varphi, N, A, C)\}$

$\overline{V}_e[\eta_e] \leftarrow (\varphi, N, A, M, C)$

return C

proc DEC(φ, N, A, C)

if $(\varphi, N, A, C) \in S$, **return** $\lightning$

$\eta_d \leftarrow \eta_d + 1$

$\overline{V}_d[\eta_d] \leftarrow (\varphi, N, A, C)$

return $\perp$

proc ORO.INIT(X)

return $\mathsf{ORO.init}(X)$

proc ORO.NEXT(X, id)

return $\mathsf{ORO.next}(X, id)$

proc FINALIZE(b')

return b'

Figure 30: idealRKAE Game.

- **ORO entries:** $(\mathsf{oro}, X, id, Z, P_o)$.
  When $X = P_o$, an entry $(\mathsf{oro}, X, id, Z, P_o)$ corresponds to a query $\mathsf{ORO.init}(X)$ with answer (Z, id), otherwise, it corresponds to a query $\mathsf{ORO.next}(X, id)$ with answer Z where P_o is the path associated with the query (i.e., $Z = \mathsf{Tab}[P_o]$). Note that in both worlds, the adversary knows P_o as it is the concatenation of the values queried to ORO for the corresponding session identifier id. For simplicity and wlog, we restrict the ORO entries to values of X that are of length r.

- **Encryption entries:** $(\mathsf{enc}, \varphi, N, A, M, C, K_e, V_o, V_m^{(\eta)})$.
  An entry $(\mathsf{enc}, \varphi, N, A, M, C, K_e, V_o, V_m^{(\eta)})$ corresponds to an encryption query ENC(φ, N, A, M) with answer C, where K_e is the key derived by φ.

  It additionally includes two sets V_o and $V_m^{(\eta)}$ that contain in the real world all the internal ORO calls made by the encryption algorithm in the corresponding encryption query, more specifically, all the (path, output) pairs that were queried to ORO. The distinction between the two sets, whose generation is described in Fig. 31, is that the outputs in $V_m^{(\eta)}$ can be partially deduced by the adversary from the ciphertext either by xoring the plaintext and the ciphertext or directly through the tag, whereas the outputs in V_o are generally unknown by the adversary. For simplicity in Fig. 29 and Fig. 32, we write $(K_e, V_o) \leftarrow \varphi^{\mathsf{ORO}}(K)$ to denote that when running $\varphi^{\mathsf{ORO}}(K)$ the key returned is K_e and the sets of (path, output) pairs queried to ORO by φ is V_o.

  In the ideal world, the sets V_o and $V_m^{(\eta)}$ and the key K_e are generated so that all good transcripts

proc REVEAL // last query before FINALIZE(b')

for $\eta = 1$ **to** η_e **do**
- $V_o, V_m^{(\eta)} \leftarrow \emptyset$
- $(K_e, N, A, M, V_o) \leftarrow \overline{V}_e[\eta]$
- $(X_1^a, \ldots, X_u^a) \xleftarrow{r-1} \mathsf{pad}(K_e \| N \| A, r-1)$
- $P \leftarrow X_1^a \parallel \langle 1 = u \rangle$
- **for** $i = 2$ **to** u
  - $V_o \leftarrow V_o \cup \{(P, \mathsf{Tab}[P])\}$
  - $P \leftarrow P \parallel X_i^a \parallel \langle i = u \rangle$
- $(X_1^m, \ldots, X_v^m) \xleftarrow{r-1} \mathsf{pad}(M, r-1)$
- **for** $i = 1$ **to** v
  - $V_m^{(\eta)} \leftarrow V_m^{(\eta)} \cup \{(P, \mathsf{Tab}[P])\}$
  - $P \leftarrow P \parallel X_i^m \parallel \langle i = v \rangle$
- **for** $i = 1$ **to** $\lceil t/r \rceil$
  - $V_m^{(\eta)} \leftarrow V_m^{(\eta)} \cup \{(P, \mathsf{Tab}[P])\}$
  - $P \leftarrow P \parallel 0^r$
- $V_e^{(\eta)} \leftarrow K_e, V_o, V_m^{(\eta)}$

$V_e \leftarrow V_e^{(1)}, \ldots, V_e^{(\eta_e)}$

for $\eta = 1$ **to** η_d **do**
- $(K_d, N, A, C \| T, V_o^{(\eta)}) \leftarrow \overline{V}_d[\eta]$
- $(X_1^a, \ldots, X_u^a) \xleftarrow{r-1} \mathsf{pad}(K_d \| N \| A, r-1)$
- $P \leftarrow \varepsilon$
- **for** $i = 1$ **to** u
  - $P \leftarrow P \parallel X_i^a \parallel \langle i = u \rangle$
  - $V_o^{(\eta)} \leftarrow V_o^{(\eta)} \cup \{(P, \mathsf{Tab}[P])\}$
- $(X_1^c, \ldots, X_v^c) \xleftarrow{r-1} \mathsf{pad}(C, r-1)$
- **for** $i = 1$ **to** v
  - $l \leftarrow \max(0, i(r-1) - |C|)$
  - $M_i \leftarrow X_i^c \oplus (\lfloor \mathsf{Tab}[P] \rfloor_{r-1-l} \parallel 0^l)$
  - $P \leftarrow P \parallel M_i \parallel \langle i = v \rangle$
  - $V_o^{(\eta)} \leftarrow V_o^{(\eta)} \cup \{(P, \mathsf{Tab}[P])\}$
- $\overline{T} \leftarrow \mathsf{Tab}[P]$
- **for** $i = 1$ **to** $\lceil t/r \rceil - 1$
  - $P \leftarrow P \parallel 0^r$
  - $V_o^{(\eta)} \leftarrow V_o^{(\eta)} \cup \{(P, \mathsf{Tab}[P])\}$
  - $\overline{T} \leftarrow \overline{T} \| \mathsf{Tab}[P]$
- $V_d^{(\eta)} \leftarrow K_d, \lfloor \overline{T} \rfloor_t, V_o^{(\eta)}$

$V_d \leftarrow V_d^{(1)}, \ldots, V_d^{(\eta_d)}$

return V_e, V_d, K

Figure 31: REVEAL oracle for the realRKAE Game.

(defined below) have a similar probability strictly greater than 0 of occurring in the real world. We describe this generation process in Fig. 32. The core idea is to generate K_e and an ORO path with the same probability of being generated in the real world. As the values in $V_m^{(\eta)}$ can be partially deduced in the real world from the ciphertext and the plaintext, they need to be defined directly from M and $C\|T$. The remaining values that would be queried in the real world to ORO are simulated in the ideal world by also querying the same path to ORO and saving the queries in V_o.

Note that while in the transcript, for convenience, we include $K_e, V_o, V_m^{(\eta)}$ in the encryption entries, they are not actually returned to the adversary by the encryption oracle. In the augmented games, these values are only revealed to the adversary at the end in the REVEAL query. Also note that each φ and the adversary $\mathcal{B}$ are distinct algorithms, and as such, they cannot interfere with each others ORO sessions.

- **Decryption entries:** $(\mathsf{dec}, \varphi, N, A, C, \perp, K_d, \overline{T}, V_o^{(\eta)})$.
  Entries of the type $(\mathsf{dec}, \varphi, N, A, C, \perp, K_d, \overline{T}, V_o^{(\eta)})$ correspond to decryption queries $\textsc{Dec}(\varphi, N, A, C)$ with answer $\perp$, where K_d is the key derived by φ. For simplicity and wlog, we also restrict the decryption entries to the ones not forwarding the output from an encryption query, i.e., not returning $\lightning$, as the outputs of these queries are deterministic and the same in both worlds. Note that in the H-coefficient technique, we only need to be concerned with attainable transcripts, and in the ideal world, decryption queries always return $\perp$ as an answer.

  In the real world, we also include in the decryption entries all the associated ORO paths queried during the corresponding decryption query as a set $V_o^{(\eta)}$ of (path, output) pairs. For simplicity in Fig. 29, we write $(K_d, V_o) \leftarrow \varphi^{\mathsf{ORO}}(K)$ to denote that when running $\varphi^{\mathsf{ORO}}(K)$ the key returned is

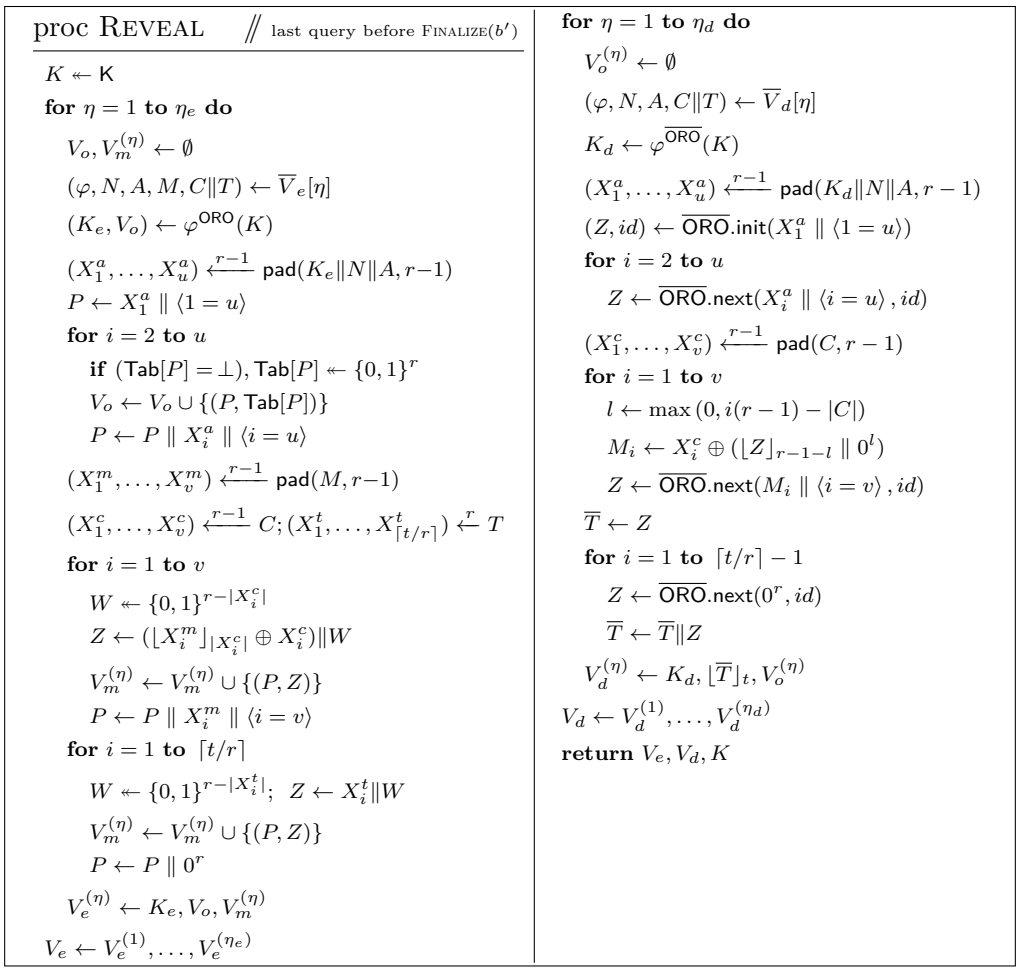

Figure 32: REVEAL oracle for the idealRKAE Game.

K_d and the sets of (path, output) pairs queried to ORO by φ is V_o.

In the ideal world, generating $V_o^{(\eta)}$ the same way as in the real world can lead to an inconsistent transcript, i.e., a transcript with probability 0 in the real world. This is due to the fact that the adversary is able to craft decryption queries which would have resulted in the real world to internal ORO calls that are already in a $V_m^{(\eta)}$, and as the inputs in $V_m^{(\eta)}$ are not really queried to ORO, we could obtain $(X, Z_1) \in V_m^{(\eta)}$ and $(X, Z_2) \in V_o^{(\eta)}$ such that $Z_1 \neq Z_2$, which would have a zero probability of happening in the real world. To circumvent this problem, we generate the ORO path through a slightly modified ORO procedure which we denote $\overline{\mathsf{ORO}}$ and describe in Fig. 28 and Fig. 32. The $\overline{\mathsf{ORO}}$ procedure responds faithfully using ORO to queries that are not a path in any $V_m^{(\eta)}$ and does not extend a path in any $V_m^{(\eta)}$. For queries that would correspond to a path in some $V_m^{(\eta)}$, the $\overline{\mathsf{ORO}}$ oracle responds in a consistent way by returning the first (already sampled) associated output. For queries that would extend a path in some $V_m^{(\eta)}$, the $\overline{\mathsf{ORO}}$ oracle samples an associated output if it is the first time it is queried or responds in a consistent way if the path has already been queried (and sampled) before. For the decryption entries in the ideal world, we save the queries to $\overline{\mathsf{ORO}}$ in $V_o^{(\eta)}$ instead of queries to ORO.

In both worlds, this set $V_o^{(\eta)}$ and the key K_d are revealed to the adversary at the end in the REVEAL

query. The string $\overline{T}$ represents, in the real world, the valid tag for C and is simply a concatenation of some of the outputs in $V_o^{(\eta)}$.

Bad Transcripts. For the H-coefficient technique, we need to lower bound the ratio of the real-world and ideal-world probabilities to a value close to one. We accomplish this by reducing the probability calculation of a transcript to a counting argument, by mainly counting the number of distinct ORO calls and random blocks generated. However, the induced calls in the sets V_m are not queried to ORO in the ideal world and can generate inconsistent transcript with either a probability zero of happening in the real world or a different probability in each world. Similarly, with a probability 0 in the real world, the sets V_o in a decryption query can correspond to a valid tag $\overline{T}$. To get a good ratio, we need to exclude and define those transcripts as bad. To facilitate our counting argument, we introduce the following sets and multi-sets (sets where elements can repeat):

$$S_1(\tau) = \bigcup_{(\mathsf{enc},\varphi,N,A,M,C,K_e,V_o,V_m^{(\eta)})\in\tau} V_m^{(\eta)}.$$

The multi-set $S_1(\tau)$ contains the ORO (path, output) pairs that were induced through encryption queries and whose outputs can be partially deduced in the real world from the ciphertext and the plaintext. In the real world, they are generated by internal ORO queries, whereas in the ideal world, they are constructed from the corresponding ciphertext and the plaintext.

$$S_2(\tau) = \bigcup_{(\mathsf{enc},\varphi,N,A,M,C,K_e,V_o,V_m^{(\eta)})\in\tau} V_o.$$

The set $S_2(\tau)$ contains the remaining ORO (path, output) pairs that were induced through encryption queries. In both worlds, they are generated by internal ORO queries.

$$S_3(\tau) = \{(P_o, Z) \mid (\mathsf{oro}, X, id, Z, P_o) \in \tau\}.$$

The set $S_3(\tau)$ contains the ORO (path, output) pairs that were induced through direct queries to ORO.init or ORO.next.

$$S_4(\tau) = \bigcup_{(\mathsf{dec},\varphi,N,A,C,\perp,K_d,\overline{T},V_o^{(\eta)})\in\tau} V_o^{(\eta)}.$$

The set $S_4(\tau)$ contains the ORO (path, output) pairs induced through decryption queries. In the real world, they are generated by internal ORO queries, whereas in the ideal world, they are generated through the internal $\overline{\mathsf{ORO}}$ procedure.

$$S_5(\tau) = (S_2(\tau) \cup S_3(\tau) \cup S_4(\tau)) \setminus S_1(\tau).$$

The set $S_5(\tau)$ contains the ORO (path, output) pairs that were induced through all the queries and that are not in $S_1(\tau)$. As they are distinct from the elements in $S_1(\tau)$, they are sampled through the ORO or $\overline{\mathsf{ORO}}$ queries.

We define as bad, transcripts that result in a different number of blocks sampled in the real world compared to the ideal world and impossible transcripts in the real world. This can happen in the following ways:

Case 1 $(x_1, y_1) \in S_1(\tau)$ and $(x_2, y_2) \in S_1(\tau)$ where $x_1 = x_2$. In this case, in the real world, the same path is queried twice to ORO resulting in a single block sampled, while in the ideal world, two output blocks y_1, y_2 are randomly sampled. This case also encompasses the case where $x_1 = x_2$ and $y_1 \neq y_2$, which is impossible in the real world. From this case, we can define the following simplified bad transcript description:

Bad_1**:** There are two distinct entries $(\mathsf{enc}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot, V_m^{(\eta)})$ and $(\mathsf{enc}, \cdot, N, \cdot, \cdot, \cdot, K_e, \cdot, \cdot)$ such that for $(P, \cdot) \in V_m^{(\eta)}$, $[P]^K = K_e$ and $[P]^N = N$.

Case 2 $(x_1, y_1) \in S_1(\tau)$ and $(x_2, y_2) \in S_2(\tau)$ where $x_1 = x_2$. In this case, in the ideal world, one output block is randomly sampled and the second is returned by an internal call to ORO, while in the real world, the same path is queried twice to ORO resulting in a single block sampled. This case also encompasses the case where $x_1 = x_2$ and $y_1 \neq y_2$, which is impossible in the real world. From this case, we can define the following simplified bad transcript description:

Bad_2**:** There are two entries $(\mathsf{enc}, \varphi, \cdot, \cdot, \cdot, \cdot, \cdot, V_o, \cdot)$ and $(\mathsf{enc}, \cdot, N, \cdot, \cdot, \cdot, K_e, \cdot, \cdot)$ such that for $(P, \cdot) \in V_o$, $[P]^K = K_e$ and $[P]^N = N$. When the two entries are the same, the conditions $[P]^K = K_e$ and $[P]^N = N$ should hold only for paths in V_o called by the function φ associated to the entry.

Case 3 $(x_1, y_1) \in S_1(\tau)$ and $(x_2, y_2) \in S_3(\tau)$ where $x_1 = x_2$. In this case, in the real world, the same path is queried twice to ORO resulting in a single block sampled, while in the ideal world, one output block is randomly sampled and the second is returned by ORO. This case also encompasses the case where $x_1 = x_2$ and $y_1 \neq y_2$, which is impossible in the real world. From this case, we can define the following simplified bad transcript description:

Bad_3**:** There are two entries $(\mathsf{oro}, X, id, Z, P_o)$ and $(\mathsf{enc}, \cdot, \cdot, \cdot, \cdot, \cdot, K_e, \cdot, \cdot)$ such that $[P_o]^K = K_e$.

Case 4 $(\mathsf{dec}, \varphi, N, A, C \| T, \bot, K_d, \overline{T}, V_o^{(\eta)}) \in \tau$ where $T = \overline{T}$. This case corresponds to an impossible transcript in the real world, as the verification query considered would have returned true. From this case, we can define the following bad transcript description:

Bad_4**:** There is an entry $(\mathsf{dec}, \varphi, N, A, C \| T, \bot, K_d, \overline{T}, V_o^{(\eta)})$ such that $T = \overline{T}$.

However, bounding the probability that $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_4$ can be problematic when the path corresponding to $\overline{T}$ has already been sampled in a previous encryption or ORO query. To assist in bounding $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_4$, we defined two more extra sets of bad transcripts that prevent these events from occurring:

Bad_5**:** There are two entries $(\mathsf{dec}, \varphi', N, A, C \| T, \bot, K_d, \overline{T}, \cdot)$ and $(\mathsf{enc}, \varphi, N, A, \cdot, C \| \overline{T}, K_e, \cdot, \cdot)$ such that $T = \overline{T}$, $K_e = K_d$ and $\varphi' \neq \varphi$.

Bad_6**:** There are two entries $(\mathsf{dec}, \cdot, \cdot, \cdot, \cdot, \cdot, K_d, \cdot, \cdot)$ and $(\mathsf{oro}, X, id, Z, P_o)$ such that $[P_o]^K = K_d$.

For $j \in \{1, \ldots, 6\}$, let Bad_j be the set of attainable transcripts satisfying the j-th bad transcript description defined above. Then the set of bad transcripts is defined as $\mathbb{T}_{\text{bad}} = \bigcup_{j=1}^{6} \mathsf{Bad}_j$.

Good Transcript Ratio. Attainable transcripts not in $\mathbb{T}_{\text{bad}}$ are called good transcripts, and the set of all good transcripts is denoted by $\mathbb{T}_{\text{good}}$. As we have excluded in $\mathbb{T}_{\text{bad}}$ all detrimental transcripts resulting in a weak H-coefficient ratio not close to one, we can now calculate this value more easily.

Let $\tau \in \mathbb{T}_{\text{good}}$ be a good transcript and let $\mathsf{P}_{\text{ideal}}(\tau)$, resp. $\mathsf{P}_{\text{real}}(\tau)$, be the probability that if we make the queries described in τ (in the same order), we receive in the ideal game, resp. the real game, the

corresponding answers recorded in τ. We denote by $(K', V_o) \leftarrow \varphi(K, V_o)$ the event that when running $\varphi(K)$ and responding deterministically to its ORO queries with the answers in the set V_o of (path, output) pairs, its queries to ORO are the ones described in V_o (in the same order) and the key returned is K'.

In the ideal world, the revealed key and the blocks in $S_1(\tau)$ are sampled at random, independently from the calls saved in $S_5(\tau)$ that correspond to values sampled through ORO and $\overline{\mathsf{ORO}}$. Note that the values in $S_5(\tau)$ are constructed in a consistent way, meaning that each path in $S_5(\tau)$ is associated with a unique corresponding output. Moreover, the coins of each derivation function φ queried during encryption and decryption are also independent from the coins of the system. Thus,

$$\mathsf{P}_{\text{ideal}}(\tau) = \frac{1}{2^k} \cdot \prod_{i=1}^{|S_1(\tau)|} \frac{1}{2^r} \cdot \prod_{i=1}^{|S_5(\tau)|} \frac{1}{2^r} \cdot \prod_{(\mathsf{enc},\varphi,\cdot,\cdot,\cdot,\cdot,K_e,V_o,\cdot)\in\tau} \Pr[\,(K_e, V_o) \leftarrow \varphi(K, V_o)] \cdot \prod_{(\mathsf{dec},\varphi,\cdot,\cdot,\cdot,\cdot,K_d,\cdot,V_o^{(\eta)})\in\tau} \Pr\left[\,(K_d, V_o^{(\eta)}) \leftarrow \varphi(K, V_o^{(\eta)})\right].$$

Note that for simplicity, in the above equation, we slightly abuse notation for V_o and $V_o^{(\eta)}$ to denote only the subset of (path, output) pairs queried by φ for the corresponding encryption/decryption query.

In the real world, as τ is a good transcript, all paths in $S_1(\tau)$ are distinct, and there are exactly $|S_1(\tau)| + |S_5(\tau)|$ distinct corresponding calls to ORO in τ. Similarly as in the ideal world, the revealed key is sampled at random, and the coins of the derivation functions φ queried during encryption are independent from the coins of the system. Thus $\mathsf{P}_{\text{real}}(\tau) = \mathsf{P}_{\text{ideal}}(\tau)$, which gives us the following H-coefficient:

$$\frac{\mathsf{P}_{\text{real}}(\tau)}{\mathsf{P}_{\text{ideal}}(\tau)} = 1. \tag{8}$$

Bad Transcript Probabilities. We now bound the probability that $\mathcal{T}_{\text{ideal}} \in \mathbb{T}_{\text{bad}}$, i.e., the probability that a transcript generated in the ideal world is bad.

We start by bounding the probability that $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_1 \cup \mathsf{Bad}_2 \cup \mathsf{Bad}_5$, by splitting the event in two cases. Note that $\mathsf{Bad}_1 \cup \mathsf{Bad}_2$ is the set of attainable transcripts that contains an entry $(\mathsf{enc}, \varphi, \cdot, \cdot, \cdot, \cdot, \cdot, V_o, V_m^{(\eta)})$ and an entry $(\mathsf{enc}, \cdot, N, \cdot, \cdot, \cdot, K_e, \cdot, \cdot)$ such that for $(P, \cdot) \in V_o \cup V_m^{(\eta)}$, $[P]^K = K_e$ and $[P]^N = N$. Thus $\mathcal{T}_{\text{ideal}}$ is in $\mathsf{Bad}_1 \cup \mathsf{Bad}_2$, if a key K_e and a nonce N corresponding to an encryption query hits a path of another encryption query.

In the first case, $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_1 \cup \mathsf{Bad}_2$ and one of the derived keys K_e hits a path called by a φ corresponding to another or the same encryption query. If we consider the following multiset $F_1 = \cup_{(\mathsf{enc},\varphi,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot,\cdot)\in\tau}\{\varphi\}$ then $F_1 \subseteq \Phi^{\mathsf{ORO}}$, $|F_1| \leq q_e$ and there is at most q_φ paths called by a $\varphi \in F_1$. Moreover, if one of the derived keys K_e hits a path called by a φ then it implies that $\exists \varphi_1, \varphi_2 \in F_1, \exists P \in \text{Qry}[\varphi_2(K)]$, $\varphi_1(K) = [P]^K$. Thus, as the user key K is sampled uniformly at random during Reveal and independently from the multiset F_1 (already fixed from the previous queries), the probability of this first case is bounded by $\Pr_{K \xleftarrow{} \mathsf{K}}[\exists \varphi_1, \varphi_2 \in F_1, \exists P \in \text{Qry}[\varphi_2(K)], \varphi_1(K) = [P]^K] \leq \mathbf{InSec}^{qi}_{\Phi^{\mathsf{ORO}}}(q_e, q_\varphi)$.

In the remaining case, either $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_5$, or $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_1 \cup \mathsf{Bad}_2$ and one of the derived keys K_e hits a path corresponding to a distinct encryption query but that is not a path called by a φ. If we consider the set $F_2 = \cup_{(\mathsf{enc},\varphi,\cdots,\cdots)\in\tau}\{\varphi\} \cup_{(\mathsf{dec},\varphi,\cdots,\cdots)\in\tau}\{\varphi\}$, then $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_5$ implies that$|\{\varphi(K) : \varphi \in F_2\}| < |F_2|$. When $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_1 \cup \mathsf{Bad}_2$ and one of the derived keys K_e hits a path corresponding to a distinct encryption query that is not a path called by a φ, then $\mathcal{T}_{\text{ideal}}$ contains two distinct entries $(\mathsf{enc}, \varphi, N, \cdot, \cdot, \cdot, K_e, V_o, V_m^{(\eta)})$ and $(\mathsf{enc}, \varphi', N', \cdot, \cdot, \cdot, K'_e, \cdot, \cdot)$ such that $K_e = K'_e$ and $N = N'$. As we consider only transcript generated by adversaries not repeating a pair (φ, N) across their encryption queries, we have that $(\varphi, N) \neq (\varphi', N')$ for the two considered entries. As $N = N'$, we obtain that $\varphi \neq \varphi'$ and the only possibility of the considered case is when two keys K_e, derived through distinct functions, collides. Again this implies that

$|\{\varphi(K) : \varphi \in F_2\}| < |F_2|$. As the two subcases imply both that $|\{\varphi(K) : \varphi \in F_2\}| < |F_2|$ and the user key K is sampled uniformly at random during REVEAL and independently from the set F_2, the probability of this second case is bounded by $\Pr_{K \leftarrow \mathsf{K}}[\,|\{\varphi(K) : \varphi \in F_2\}| < |F_2|] \leq \mathbf{InSec}^{cr}_{\Phi^{\mathsf{ORO}}}(q_e + q_d)$.

Using a union bound, we obtain

$$\Pr[\,\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_1 \cup \mathsf{Bad}_2 \cup \mathsf{Bad}_5] \leq \mathbf{InSec}^{qi}_{\Phi^{\mathsf{ORO}}}(q_e, q_\varphi) + \mathbf{InSec}^{cr}_{\Phi^{\mathsf{ORO}}}(q_e + q_d). \tag{9}$$

If we consider the multiset $F_3 = \cup_{(\mathsf{enc},\varphi,\cdots,\cdots)\in\tau}\{\varphi\} \cup_{(\mathsf{dec},\varphi,\cdots,\cdots)\in\tau} \{\varphi\}$ and the set $X = \cup_{(\mathsf{oro},\cdot,\cdot,\cdot,P_o)\in\tau}\left\{[P_o]^K : [P_o]^K \neq \varepsilon\right\}$ then $F_3 \subseteq \Phi^{\mathsf{ORO}}$, $|F_3| \leq q_e + q_d$, $X \subseteq \mathsf{K}$ and $|X| \leq q_o$. $\mathcal{T}_{\text{ideal}}$ is in $\mathsf{Bad}_3 \cup \mathsf{Bad}_6$ if one of the derived keys $\varphi(K)$ corresponding to the encryption/decryption queries and computed during REVEAL, hits a path queried to ORO by the adversary, i.e., if $\{\varphi(K) : \varphi \in F_3\} \cap X \neq \emptyset$. As the user key K is sampled uniformly at random during REVEAL and independently from the sets F_3 and X, the probability that $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_3 \cup \mathsf{Bad}_6$ is bounded by $\Pr_{K \leftarrow \mathsf{K}}[\,\{\varphi(K) : \varphi \in F_3\} \cap X \neq \emptyset]$. Thus, using the definition of output-unpredictability, we obtain

$$\Pr[\,\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_3 \cup \mathsf{Bad}_6] \leq \mathbf{InSec}^{up}_{\Phi^{\mathsf{ORO}}}(q_e + q_d, q_o). \tag{10}$$

When $\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_4$, it means that there is an entry $(\mathsf{dec}, \varphi, N, A, C\|T, \perp, K_d, \overline{T}, V_o^{(\eta)})$ such that $T = \overline{T}$. If we additionally assume $\mathcal{T}_{\text{ideal}} \notin \mathsf{Bad}_5 \cup \mathsf{Bad}_6$ then, for any entry $(\mathsf{dec}, \varphi, N, A, C\|T, \perp, K_d, \overline{T}, V_o^{(\eta)})$, the path corresponding to $\overline{T}$ is entirely sampled during REVEAL. Indeed, $\mathcal{T}_{\text{ideal}} \notin \mathsf{Bad}_6$ means that $\overline{T}$ is not sampled during an ORO query as otherwise, the key corresponding to the decryption query would hit the path of the ORO query. And $\mathcal{T}_{\text{ideal}} \notin \mathsf{Bad}_5$ means that $\overline{T}$ is not sampled during an encryption query (i.e., are in a $V_m^{(\eta)}$) when $T = \overline{T}$, as otherwise the key/nonce/ciphertext corresponding to the decryption query would hit the key/nonce/ciphertext of the encryption query. Thus, when $\mathcal{T}_{\text{ideal}} \notin \mathsf{Bad}_5 \cup \mathsf{Bad}_6$, for each decryption entry $(\mathsf{dec}, \varphi, N, A, C\|T, \perp, K_d, \overline{T}, V_o^{(\eta)})$, the value $\overline{T}$ is sampled uniformly at random and independently from T at the end during REVEAL and the probability that $T = \overline{T}$ is 2^{-t}. Summing over all decryption entries, we obtain

$$\Pr[\,(\mathcal{T}_{\text{ideal}} \in \mathsf{Bad}_4) \wedge (\mathcal{T}_{\text{ideal}} \notin \mathsf{Bad}_5 \cup \mathsf{Bad}_6)] \leq \frac{q_d}{2^t}. \tag{11}$$

As $\mathbb{T}_{\text{bad}} = \bigcup_{j=1}^{6} \mathsf{Bad}_j$, using a union bound and substituting terms through equations (9)–(11), we have

$$\Pr[\,\mathcal{T}_{\text{ideal}} \in \mathbb{T}_{\text{bad}}] \;\leq\; \mathbf{InSec}^{qi}_{\Phi^{\mathsf{ORO}}}(q_e, q_\varphi) \;+\; \mathbf{InSec}^{cr}_{\Phi^{\mathsf{ORO}}}(q_e \;+\; q_d) \;+\; \mathbf{InSec}^{up}_{\Phi^{\mathsf{ORO}}}(q_e \;+\; q_d, q_o) \;+\; \frac{q_d}{2^t}. \tag{12}$$

Theorem 7.8 is obtained by combining equations (8) and (12) with Theorem A.1, where equation (8) yields ϵ_1 and equation (12) yields ϵ_2.

C.4 Proof of Theorem 7.10 (SpongeWrap CMT-AEAD Security Proof)

Let $\mathcal{A}$ be a CMT adversary and $(K_1, N_1, A_1, M_1), (K_2, N_2, A_2, M_2)$ be its output. Let $C_1\|T_1 = \mathsf{E}(K_1, N_1, A_1, M_1)$ and $C_2\|T_2 = \mathsf{E}(K_2, N_2, A_2, M_2)$. When $(K_1, N_1, A_1) \neq (K_2, N_2, A_2)$, the associated ORO path in $\mathsf{E}(K_1, N_1, A_1, M_1)$ and $\mathsf{E}(K_2, N_2, A_2, M_2)$ are different.

If the sampling of both complete paths happens during queries to ORO by $\mathcal{A}$, when $t \leq r$, the probability that $T_1 = T_2$ is bounded by the probability that a pair of queries to ORO collides on t bits, and when $t > r$, by the probability that such a pair collides on r bits and the fixed appended path $0^r\| \cdots \|0^r$ collides on $t - r$ bits. As this probability is 2^{-t} and there is at most $\binom{q_o}{2}$ such pairs, the probability of this event happening is at most $\frac{q_o^2}{2^{t+1}}$.

If a partial sampling of the paths happens when calling ORO during FINALIZE, when $t \leq r$, the probability that $T_1 = T_2$ is 2^{-t}. When $t > r$, and T_1 or T_2 are completely sampled during FINALIZE, the probability that $T_1 = T_2$ is also 2^{-t}. Lastly, when $t > r$ and T_1, T_2 are partially sampled by $\mathcal{A}$, the probability that $T_1 = T_2$ is again bounded by the probability that a pair of queries to ORO collides on r bits and the fixed appended path $0^r \| \cdots \| 0^r$ collides on $t - r$ bits. The probability of this event is again bounded by $\frac{q_o^2}{2^{t+1}}$.
Since each of these cases are mutually exclusive, the CMT advantage of $\mathcal{A}$ is bounded by $\frac{\max(q_o^2,2)}{2^{t+1}} \leq \frac{q_o^2+2}{2^{t+1}}$.