Five Rounds are Sufficient and Necessary for the Indifferentiability of Iterated Even-Mansour

Yuanxi Dai*, Yannick Seurin**, John Steinberger***, and Aishwarya Thiruvengadam†

January 18, 2017

Abstract. We prove that the 5-round iterated Even-Mansour (IEM) construction (which captures the high-level structure of the class of key-alternating ciphers) with a non-idealized key-schedule (such as the trivial key-schedule, where all round keys are equal) is indifferentiable from an ideal cipher. In a separate result, we also prove that five rounds are necessary by describing an attack against the corresponding 4-round construction. This closes the gap regarding the exact number of rounds for which the IEM construction with a non-idealized key-schedule is indifferentiable from an ideal cipher, which was previously only known to lie between four and twelve.

1 Introduction

Background. A large number of block ciphers are so-called *key-alternating ciphers*, i.e., they alternatively apply two types of transformations to the current state: the (usually bitwise) addition of a *secret* round key, and the application of a *public* permutation. This is in particular the case of virtually all SPNs (*Substitution Permutation Networks*) such as the AES. Hence, in general, for an r-round key-alternating cipher with message space $\{0,1\}^n$, the ciphertext y is computed from the plaintext x as

$$y = k_r \oplus P_r(k_{r-1} \oplus P_{r-1}(\cdots P_2(k_1 \oplus P_1(k_0 \oplus x))\cdots)), \tag{1}$$

where $(k_0, \ldots, k_r)$ are n-bit round keys usually derived from a master key k of size close to n, and $P_1, \ldots, P_r$ are n-bit public permutations.

A recent trend has been to analyze this class of block ciphers in the so-called Random Permutation Model (RPM), which consists in modeling permutations $P_1, \ldots, P_r$ as oracles that the adversary can only query (from both sides) in a black-box way, each behaving as a perfectly random permutation. This approach allows to assert the nonexistence of *generic attacks*, i.e., attacks not exploiting the particular structure of "concrete" (i.e., fully specified by a short description)

* Tsinghua University, P.R. China. E-mail: dyx13@mails.tsinghua.edu.cn
** ANSSI, Paris, France. E-mail: yannick.seurin@m4x.org
*** Tsinghua University, P.R. China. E-mail: jpsteinb@gmail.com
† University of Maryland, United States. E-mail: aish@cs.umd.edu

permutations. This approach dates back to Even and Mansour [25] who studied the case $r = 1$. For this reason, construction (1), once seen as a way to define a block cipher from an arbitrary tuple of permutations $\mathbf{P} = (P_1, \ldots, P_r)$, is often called the *iterated Even-Mansour (IEM) construction*. The general case of $r \geq 2$ rounds was only considered more than 20 years later in a series of papers [11, 45, 37, 14, 13, 31], primarily focusing on the standard security notion for block ciphers, namely pseudorandomness, which requires that no computationally bounded adversary with (usually two-sided) black-box access to a permutation can distinguish whether it is interacting with the block cipher under a random key or a perfectly random permutation. Pseudorandomness of the IEM construction with independent round keys is by now well understood, the security bound increasing beyond the "birthday bound" (the original bound proved for the 1-round Even-Mansour construction [25, 24]) as the number of rounds increases [14, 31].

The Ideal Cipher Model. Even though pseudorandomness has been the primary security requirement any block cipher should satisfy, in some cases this property is not enough to establish the security of higher-level cryptosystems where the block cipher is used. For example, the security of some real-world authenticated encryption protocols such as 3GPP confidentiality and integrity protocols f8 and f9 [33] rely on the stronger block cipher security notion of *indistinguishability under related-key attacks* [7, 3]. Another context where this problem shows up is for block-cipher based hash functions [36, 42]. In such cases, the adversary controls both the message and the key of the block cipher, and hence can exploit "known-key" or "chosen-key" attacks [35, 8] in order to break the collision- or preimage-resistance of the hash function.

For this reason, cryptographers have come to view a good block cipher as something close to an *ideal cipher (IC)*, i.e., a family of 2^κ uniformly random and independent permutations, where κ is the key-length of the block cipher. Perhaps not surprisingly, this view turned out to be very fruitful for proving the security of constructions based on a block cipher when the PRP assumption is not enough [46, 41, 10, 34, 4, 22, 28, 6], an approach often called the *ideal cipher model (ICM)*. On the other hand, similarly to the random oracle model [27, 5], this ultimately remains a heuristic approach, as one can construct (artificial) schemes that are secure in the ICM, but insecure for any concrete instantiation of the block cipher [9]. This means that there is no hope to formalize (let alone prove) what it means for a concrete block cipher to "behave" as an ideal cipher and that the strength of a concrete block cipher in this respect should ultimately be evaluated through cryptanalysis (once settled the preliminary question of what should be considered as a harmful, undesirable, "non-random" property for a block cipher!).

Indifferentiability. This does not mean that the provable security approach has nothing to offer regarding how to design something close to an ideal cipher. Indeed, the indifferentiability framework, introduced by Maurer *et al.* [40] and popularized by Coron *et al.* [18], allows to assess whether a construction of some

target primitive A (e.g., a block cipher) from some lower-level ideal primitive B (e.g., for the IEM construction, a small number of random permutations $(P_1, \ldots, P_r)$) is "structurally close" to the ideal version of A (e.g., an ideal cipher). The power of indifferentiability stems from a powerful composition theorem [40] which ensures that a large class of protocols (see [43, 21] for restrictions) provably secure when used with the ideal-A primitive remain secure (in the ideal-B model) when used with a construction of A from B which is indifferentiable from ideal-A.

Previous Results. Two papers have previously explored the indifferentiability of the IEM construction from an ideal cipher. In [1], Andreeva *et al.* showed that the 5-round IEM construction with an idealized key-schedule (i.e., the function(s) mapping the master key onto the round key(s) are modeled as random oracles) is indifferentiable from an ideal cipher. Subsequently, Lampe and Seurin [38] showed that the 12-round IEM construction with the trivial key-schedule, i.e., in which all round keys are equal, is also indifferentiable from an ideal cipher. Moreover, both papers included impossibility results for the indifferentiability of the 3-round IEM construction with a trivial key-schedule showing that at least four rounds must be necessary in that contexts. In both settings, the question of the exact number of rounds needed to make the IEM construction indifferentiable from an ideal cipher remained open. In this work, we settle the question for the case of non-idealized key-schedules.

Our Results. We improve both the positive and negative results for the indifferentiability of the IEM construction with the trivial (and more generally, non-idealized) key-schedule. Specifically, we show an attack on the 4-round IEM construction, and prove that the 5-round IEM construction is indifferentiable from an ideal cipher, in both cases for the trivial key-schedule.[1] Hence, our work resolves the question of the exact number of rounds needed for the IEM construction with a non-idealized key-schedule to achieve indifferentiability from an IC.

Our 4-round impossibility result improves on the afore-mentioned 3-round impossibility results of [1, 38]. It can be seen as an extension of the attack against the 3-round IEM with the trivial key-schedule given in [38]. But unlike this 3-round attack, our new 4-round attack does not merely consist in finding a tuple of key/plaintext/ciphertext triples for the construction satisfying a so-called "evasive" relation (i.e., a relation which is hard to find with only black-box access to an ideal cipher, e.g. a triple (k, x, y) such that $x \oplus y = 0$). Instead, it relies on relations on the "internal" variables of the construction (which makes the attack harder to analyze rigorously). We note that a simple "evasive-relation-finding" attack against four rounds had previously been excluded by Cogliati and Seurin [15] (in technical terms, they proved that the 4-round IEM construction is *sequentially*-indifferentiable from an IC, see Remark 2 in Section 3) so the extra complexity of our 4-round attack is in a sense inevitable.

[1] Actually we consider a slight variant of the trivial key-schedule where the first and last round keys are omitted, but both our negative and positive results are straightforward to extend to the "standard" trivial key-schedule. See Section 2 for a discussion.

Our 5-round feasibility result improves *both* on the 5-round result of [1] for the IEM construction with idealized key-schedules and on the 12-round feasibility result of [38] for the IEM construction with the trivial key-schedule. Our simulator runs in time $O(q^5)$ and makes $O(q^5)$ ideal cipher queries. The security bound is quite poor, but we believe it can be improved with a more fined-grained analysis of "bad events".

A Glimpse at the Simulator. Our 5-round simulator follows the traditional "chain detection/completion" paradigm, pioneered by Coron *et al.* [19, 32, 17] for proving indifferentiability of the Feistel construction, which has since then been used for the IEM construction as well [1, 38]. However, it is, in a sense, conceptually simpler and more "systematic" than previous simulators for the IEM construction (something we pay by a more complex "termination" proof). In a nutshell, our new 5-round simulator detects and completes *any* path of length three, where a path is a sequence of adjacent permutation queries "chained" by the same key (and which might "wrap around" the ideal cipher). In contrast, the 12-round simulator of [38] used a much more parsimonious chain detection strategy (inherited from [19, 44, 32, 17]) which allowed a much simpler termination argument.

Once a tentative simulator has been determined, the indifferentiability proof usually entails two technical challenges: on one hand, proving that the simulator works hard enough to ensure that it will never be trapped into an inconsistency, and on the other hand, proving that it does not work in more than polynomial time (except maybe with some negligible probability). Finding the right balance between these two requirements is at the heart of the design of a suitable simulator.

The proof that our new 5-round simulator remains consistent with the IC roughly follows the same ideas as in previous indifferentiability proofs. In short, since the simulator completes all paths of length three, this means that at the moment the distinguisher makes a permutation query, only incomplete paths of length at most two can exist. Hence any incomplete path has three "free" adjacent positions, two of which (the ones on the edge) will be sampled at random, while the middle one will be adapted to match the IC. The most delicate part consists in proving that no path of length three can appear "unexpectedly" and remain unnoticed by the simulator (which will therefore not complete it), which ultimately relies on excluding a (large) number of "bad events".

We feel the most innovative part of our new proof lies in the "termination argument", i.e., proving that the simulator is efficient and that the recursive chain detection/completion process does not "chain react" beyond a fixed polynomial bound. As in many previous termination arguments [19, 44, 32, 17] the proof is "bootstrapped" by proving that certain types of paths (namely those that wrap around the IC) will ever be detected and completed only if the distinguisher made the corresponding IC query. Hence, assuming the distinguisher makes at most q queries, at most q such paths will be completed. In virtually all previous indifferentiability proofs, this fact easily allows to upper bound the size of permutations "history" for all other "detect zones" used by the simulator,

and hence to upper bound the total number of paths that will ever be detected and completed. This is not the case for our 5-round simulator, since the "at most q wrapping paths" trick only allows to upper bound the size of the middle permutation P_3, which by itself is not sufficient to upper bound the number of other detected paths. In order to push the argument further, we need to prove a structural property of the history of adjacent permutations P_2 and P_4. In more detail, this property is that the table maintaining answers of the simulator for P_2 (resp. P_4) never contains four distinct input/output pairs $(x^{(i)}, y^{(i)})$, $1 \leq i \leq 4$, such that $\bigoplus_{1 \leq i \leq 4}(x^{(i)} \oplus y^{(i)}) = 0$. It is rather straightforward to prove that such input/output pairs are unlikely to exist if the simulator sets answers at random, but answers are sometimes "adapted" when completing a path, which makes the proof much more complicated.

Related Work and Open Problems. Several papers have studied security properties of the IEM construction that are stronger than pseudorandomness yet weaker than indifferentiability, such as resistance to related-key [26, 15], known-key [2, 16], or chosen-key attacks [15, 29]. A recent preprint shows that the 3-round IEM construction with a (non-invertible) idealized key-schedule is indifferentiable from an ideal cipher [30]. This complements our work by settling the problem analogous to ours in the case of idealized key-schedules. In both cases, the main open question is whether the concrete indifferentiability bounds (which are typically dramatically poor) can be improved.

Organization of the Paper. We start with some preliminary definitions in Section 2. The attack against the 4-round IEM construction is given and analyzed in Section 3. Our 5-round simulator is described in Section 4, while the indifferentiability proof is in Section 5.

2 Preliminaries

General Definitions and Notation. Throughout the paper, n will denote the block length of permutations $P_1, \ldots, P_r$ of the IEM construction and will play the role of security parameter for asymptotic statements. Given a finite non-empty set S, we write $s \leftarrow_{\$} S$ to mean that an element is drawn uniformly at random from S and assigned to s.

A *distinguisher* is an oracle algorithm $\mathcal{D}$ with oracle access to a finite list of oracles $(\mathcal{O}_1, \mathcal{O}_2, \ldots)$ and that outputs a single bit b, which we denote $\mathcal{D}^{\mathcal{O}_1, \mathcal{O}_2, \cdots} = b$ or $\mathcal{D}[\mathcal{O}_1, \mathcal{O}_2, \ldots] = b$.

A block cipher with key space $\{0,1\}^\kappa$ and message space $\{0,1\}^n$ is a mapping $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ such that for any key $k \in \{0,1\}^\kappa$, $x \mapsto E(k,x)$ is a permutation. An ideal cipher with block length n and key length κ is a block cipher drawn uniformly at random from the set of all block ciphers with block length n and key length κ.

The Iterated Even-Mansour Construction. Fix integers $n, r \geq 1$. Let $\mathbf{f} = (f_0, \ldots, f_r)$ be a $(r+1)$-tuple of functions from $\{0,1\}^n$ to $\{0,1\}^n$. The r-round iterated Even-Mansour construction $\mathrm{EM}[n, r, \mathbf{f}]$ specifies, from any r-tuple $\mathbf{P} = (P_1, \ldots, P_r)$ of permutations of $\{0,1\}^n$, a block cipher with n-bit keys and n-bit messages, simply denoted $\mathrm{EM}^{\mathbf{P}}$ in all the following (parameters $[n, r, \mathbf{f}]$ will always be clear from the context), which maps a plaintext $x \in \{0,1\}^n$ and a key $k \in \{0,1\}^n$ to the ciphertext defined by

$$\mathrm{EM}^{\mathbf{P}}(k, x) = f_r(k) \oplus P_r(f_{r-1}(k) \oplus P_{r-1}(\cdots P_2(f_1(k) \oplus P_1(f_0(k) \oplus x)) \cdots)).$$

We say that the key-schedule is *trivial* when all f_i's are the identity.

In this work, we make the observation that the first and the last key additions do not play any role for indifferentiability. Indeed, in this setting, the key is just a "public" input to the construction, much like the plaintext/ciphertext. What provides security are the random permutations, that remain secret for inputs that have not been queried by the attacker. Hence, we will focus on the slight variant of the trivial key-schedule where $f_0 = f_r = 0$ (see Fig. 3), but our results carry over directly to the "standard' trivial key-schedule (and more generally to any non-idealized key-schedule where the f_i's are permutations of $\{0,1\}^n$).

Indifferentiability. We recall the standard definition of indifferentiability (for clarity, we state it directly for the IEM construction as opposed to the more general context of the instantiation of a primitive A from a primitive B).

Definition 1. The construction $\mathrm{EM}^{\mathbf{P}}$ with access to an r-tuple $\mathbf{P} = (P_1, \ldots, P_r)$ of random permutations is $(t_{\mathcal{S}}, q_{\mathcal{S}}, \varepsilon)$-*indifferentiable* from an ideal cipher IC if there exists a simulator $\mathcal{S} = \mathcal{S}(q)$ such that $\mathcal{S}$ runs in total time $t_{\mathcal{S}}$ and makes at most $q_{\mathcal{S}}$ queries to IC, and such that

$$\left| \Pr[D^{\mathrm{EM}^{\mathbf{P}}, \mathbf{P}} = 1] - \Pr[D^{\mathrm{IC}, \mathcal{S}^{\mathrm{IC}}} = 1] \right| \leq \varepsilon$$

for every (information-theoretic) distinguisher D making at most q queries in total.

We simply say that the r-round IEM construction is indifferentiable from an ideal cipher if for any q polynomial in n, it is $(t_{\mathcal{S}}, q_{\mathcal{S}}, \varepsilon)$-indifferentiable from an ideal cipher with $t_{\mathcal{S}}, q_{\mathcal{S}}$ polynomial in n and ε negligible in n.

Remark 1. Note that Definition 1 allows the simulator $\mathcal{S}$ to depend on the number of queries q. In fact, the simulator that we show in the pseudocode (cf. Figs. 4 and 5) does not depend on q, but this simulator is efficient only with high probability, as will become clear in the proof. In Theorem 42 we discuss an optimized implementation of this simulator that, among others, uses knowledge of q to abort whenever its runtime exceeds the limit of a "good" execution, thus ensuring that the simulator is efficient with probability 1.

3 Attack Against 4-Round Simulators

In this section, we describe an attack against the 4-round IEM construction[2], improving previous attacks against three rounds given in [1, 38]. Consider the distinguisher $\mathcal{D}$ whose pseudocode is given in Fig. 1 (see also Fig. 2 for an illustration of the attack). This distinguisher can query the permutations/simulator through the interface $\mathrm{Query}(i, \delta, z)$, and the EM construction/ideal cipher through interfaces $\mathrm{Enc}(k, x)$ and $\mathrm{Dec}(k, y)$.

$y_3 \leftarrow_{\$} \{0,1\}^n$
$x_4 \leftarrow_{\$} \{0,1\}^n$
$x'_4 \leftarrow_{\$} \{0,1\}^n \setminus \{x_4\}$
$k \coloneqq y_3 \oplus x_4$
$k' \coloneqq y_3 \oplus x'_4$
$y_4 \coloneqq \mathrm{Query}(4, +, x_4)$
$y'_4 \coloneqq \mathrm{Query}(4, +, x'_4)$
$x_1 \coloneqq \mathrm{Dec}(k, y_4)$
$x'_1 \coloneqq \mathrm{Dec}(k', y'_4)$
if $x_1 = x'_1$ **then**
  return 0
$y_1 \coloneqq \mathrm{Query}(1, +, x_1)$
$y'_1 \coloneqq \mathrm{Query}(1, +, x'_1)$
$x_2 \coloneqq y_1 \oplus k$
$x'_2 \coloneqq y'_1 \oplus k'$
$k'' \coloneqq y_1 \oplus x'_2$
$k''' \coloneqq k'' \oplus k \oplus k'$
$y''_4 \coloneqq \mathrm{Enc}(k'', x_1)$
$y'''_4 \coloneqq \mathrm{Enc}(k''', x'_1)$
if y_4, y'_4, y''_4, y'''_4 are not distinct **then**
  return 0
draw $b \leftarrow_{\$} \{0,1\}$
if $b = 1$ **then**
  $y''_4 \leftarrow_{\$} \{0,1\}^n \setminus \{y_4, y'_4\}$
  $y'''_4 \leftarrow_{\$} \{0,1\}^n \setminus \{y_4, y'_4, y''_4\}$
$x''_4 \coloneqq \mathrm{Query}(4, -, y''_4)$
$x'''_4 \coloneqq \mathrm{Query}(4, -, y'''_4)$
if $b = 0$ **then**
  return $x''_4 \oplus x'''_4 = x_4 \oplus x'_4$
else $(b = 1)$
  return $x''_4 \oplus x'''_4 \neq x_4 \oplus x'_4$

Fig. 1. Pseudocode of the attack against the 4-round IEM construction.

We prove that $\mathcal{D}$ has advantage close to $1/2$ against any simulator making a polynomial number of queries to the IC. More formally, we have the following result.

Theorem 1. *Let $\mathcal{S}$ be any simulator making at most σ IC queries when interacting with $\mathcal{D}$. Then the advantage of $\mathcal{D}$ in distinguishing $(\mathrm{EM}^{\mathbf{P}}, \mathbf{P})$ and $(\mathrm{IC}, \mathcal{S}^{\mathrm{IC}})$ is at least*

$$\frac{1}{2} - \frac{4\sigma}{2^n} - \frac{7}{2^n}.$$

[2] For simplicity, we consider the construction without the first and the last round keys, but the attack can be straightforwardly extended to the construction with first and last round keys.

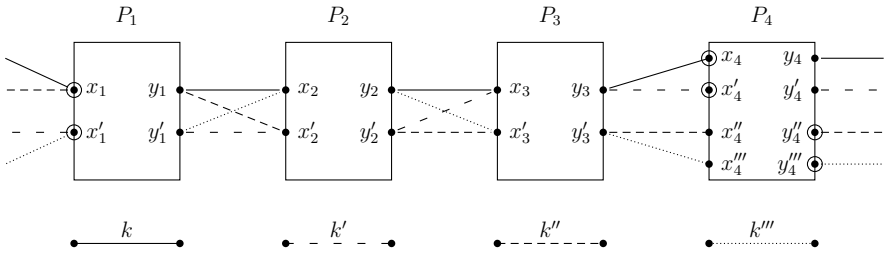

Fig. 2. Illustration of the attack against the 4-round IEM construction. The circled dots correspond to queries made by the distinguisher to the permutations/simulator.

Proof. We start by showing that the distinguisher returns 1 with probability close to one when interacting with the real world $(\mathrm{EM}^{\mathbf{P}}, \mathbf{P})$. For this, we denote

$$\begin{aligned} x_3 &= P_3^{-1}(y_3), \\ y_2 &= x_3 \oplus k, \\ y'_2 &= x_3 \oplus k'. \end{aligned}$$

Then, it is easy to check that $P_2^{-1}(y_2) = x_2$ and $P_2^{-1}(y'_2) = x'_2$, where x_2 and x'_2 are the values obtained by the distinguisher at lines 14 and 15 respectively.

We first observe that the probability that $\mathcal{D}$ returns 0 at line 11 or 21 is negligible. Consider line 11 first. One has

$$x_1 = x'_1 \Leftrightarrow y_1 = y'_1 \Leftrightarrow x_2 \oplus x'_2 = k \oplus k' \Leftrightarrow P_2^{-1}(y_2) \oplus P_2^{-1}(y'_2) = k \oplus k'.$$

By construction, y_2 and y'_2 are distinct and $k \oplus k' \neq 0$, so that the last equality above is satisfied with probability exactly 2^{-n} over the randomness of P_2. Next, we observe that, assuming $x_1 \neq x'_1$, then the four keys (k, k', k'', k''') are distinct. Indeed, by construction $k \neq k'$, which implies that $k'' \neq k'''$. Hence, the only possibilities of equality are $k'' = k$ (which is equivalent to $k''' = k'$) or $k'' = k'$ (which is equivalent to $k''' = k$). Note that

$$k'' = y_1 \oplus x'_2 = x_2 \oplus k \oplus x'_2 = k \oplus P_2^{-1}(y_2) \oplus P_2^{-1}(y'_2).$$

Since $y_2 \neq y'_2$ by construction, we see that $k'' = k$ is impossible, while $k'' = k'$ *iff* $P_2^{-1}(y_2) \oplus P_2^{-1}(y'_2) = k \oplus k'$, which turns out to be equivalent to $x_1 = x'_1$, as seen above. Hence the claim. Consider now line 21. By construction, we have $y_4 \neq y'_4$. Moreover, since the four keys (k, k', k'', k''') are distinct, y''_4 and y'''_4 are uniformly random, so that $y''_4 \in \{y_4, y'_4\}$ with probability $2/2^n$, and $y'''_4 \in \{y_4, y'_4, y''_4\}$ with probability $3/2^n$. All in all,we have

$$\Pr\left[\mathcal{D} \text{ outputs 0 at line 11 or 21}\right] \leq \frac{6}{2^n}. \tag{2}$$

Then, we show that conditioned on $\mathcal{D}$ not returning 0 at line 11 or 21, it returns 1 with probability very close to one. Consider first the case where $b = 0$,

which corresponds to $y_4'' = \mathrm{Enc}(k'', x_1)$ and $y_4''' = \mathrm{Enc}(k''', x_1')$. Then x_4'' and x_4''' are the input values to P_4 when encrypting respectively x_1 with key k'' and x_1' with key k'''. It is easy to check that these two encryptions share the common input $x_3' \coloneqq y_2' \oplus k'' = y_2 \oplus k'''$ to P_3. Let $y_3' = P_3(x_3')$. Then $x_4'' = y_3' \oplus k''$ and $x_4''' = y_3' \oplus k'''$, which implies

$$x_4'' \oplus x_4''' = k'' \oplus k''' = k \oplus k' = x_4 \oplus x_4'.$$

Hence, the distinguisher always outputs 1 when $b = 0$.

Consider now the case $b = 1$. Then y_4'' is sampled uniformly at random from $\{0,1\}^n \setminus \{y_4, y_4'\}$, and y_4'' is sampled uniformly at random from $\{0,1\}^n \setminus \{y_4, y_4', y_4''\}$. This is equivalent to sampling x_4'' uniformly at random from $\{0,1\}^n \setminus \{x_4, x_4'\}$ and x_4''' from $\{0,1\}^n \setminus \{x_4, x_4', x_4''\}$. Hence, $x_4'' \oplus x_4''' = x_4 \oplus x_4'$ with probability $1/(2^n - 3)$ and the distinguisher returns 1 with probability at least $1 - 2/2^n$ (assuming $n \geq 3$). All in all, we have

$$\Pr\left[\mathcal{D}[\mathrm{EM}^{\mathbf{P}}, \mathbf{P}] = 1 \,\middle|\, \neg(\mathcal{D} \text{ outputs 0 at line 11 or 21})\right] \geq 1 - \frac{1}{2^n}. \tag{3}$$

Gathering (2) and (3), we obtain

$$\Pr\left[\mathcal{D}[\mathrm{EM}^{\mathbf{P}}, \mathbf{P}] = 1\right] \geq \left(1 - \frac{6}{2^n}\right)\left(1 - \frac{1}{2^n}\right) \geq 1 - \frac{7}{2^n}.$$

We now consider what happens in the ideal world $(\mathrm{IC}, \mathcal{S}^{\mathrm{IC}})$. Intuitively, unless it makes an IC query with one of the four keys (k, k', k'', k''') used by the distinguisher, the simulator is unable to guess the bit b drawn by the distinguisher at line 22, and without knowing b the simulator has chance $\sim 1/2$ of causing $\mathcal{D}$ to output 1. We make this idea formal in what follows. The main technical ingredient here is a "domain separation lemma" by Boneh and Shoup [12]. For completeness, we also provide a simple proof (based on Patarin's H-coefficient technique) of this lemma in Appendix A.

Let $\mathcal{D}_z$, $z = 0, 1$, be obtained from $\mathcal{D}$ by hard-wiring $b = z$ at line 22. We consider $\mathcal{D}_0$ and $\mathcal{D}_1$ as two separate worlds for $\mathcal{S}$ to interact with; for $\star \in \{=, \neq\}$ and $z \in \{0,1\}$ let

$$\Pr[\mathcal{S}^{\mathcal{D}_z} \to \star]$$

denote the probability that while interacting with $\mathcal{D}_z$, the game reaches line 27 (i.e., that $\mathcal{D}_z$ does not return early at line 11 or 21) and that $\mathcal{S}$ outputs a value x_4''' at line 27 such that $x_4'' \oplus x_4''' \star x_4 \oplus x_4'$. (E.g.,

$$\Pr[\mathcal{S}^{\mathcal{D}_0} \to =]$$

is the probability that while interacting with $\mathcal{D}_0$, the game reaches line 27 and $\mathcal{S}$ outputs $x_4''' = x_4'' \oplus x_4 \oplus x_4'$ at that line.) Then $\mathcal{S}$'s chance of making $\mathcal{D}$ output 1 is

$$\frac{1}{2}\Pr[\mathcal{S}^{\mathcal{D}_0} \to =] + \frac{1}{2}\Pr[\mathcal{S}^{\mathcal{D}_1} \to \neq]$$

since b is selected uniformly at random on line 22. Since

$$\Pr[\mathcal{S}^{\mathcal{D}_1} \to \neq] \leq 1 - \Pr[\mathcal{S}^{\mathcal{D}_1} \to =]$$

(the inequality could be strict, as line 27 might not even be reached), $\mathcal{S}$'s probability of making $\mathcal{D}$ output 1 is at most

$$\frac{1}{2} + \frac{1}{2}(\Pr[\mathcal{S}^{\mathcal{D}_0} \to =] - \Pr[\mathcal{S}^{\mathcal{D}_1} \to =]) = \frac{1}{2} + \frac{1}{2}\Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_1)$$

where

$$\Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_1) := \Pr[\mathcal{S}^{\mathcal{D}_0} \to =] - \Pr[\mathcal{S}^{\mathcal{D}_1} \to =]$$

is $\mathcal{S}$'s "distinguishing advantage" at telling $\mathcal{D}_0$ from $\mathcal{D}_1$.

We will upper bound $\Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_1)$ by introducing two intermediate worlds $\mathcal{D}_{0*}$ and $\mathcal{D}_{1*}$. Briefly, $\mathcal{D}_{z*}$ is identical to $\mathcal{D}_z$ except that $\mathcal{S}$ is given a "dummy" ideal cipher oracle IC^* in $\mathcal{D}_{z*}$ that is independent from the "real" ideal cipher oracle IC (notated Enc/Dec in the pseudocode) being used by $\mathcal{D}_z$. Since

$$\Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_1) = \Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_{0*}) + \Delta_{\mathcal{S}}(\mathcal{D}_{0*}, \mathcal{D}_{1*}) + \Delta_{\mathcal{S}}(\mathcal{D}_{1*}, \mathcal{D}_1)$$

where

$$\begin{aligned}
\Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_{0*}) &:= \Pr[\mathcal{S}^{\mathcal{D}_0} \to =] - \Pr[\mathcal{S}^{\mathcal{D}_{0*}} \to =] \\
\Delta_{\mathcal{S}}(\mathcal{D}_{0*}, \mathcal{D}_{1*}) &:= \Pr[\mathcal{S}^{\mathcal{D}_{0*}} \to =] - \Pr[\mathcal{S}^{\mathcal{D}_{1*}} \to =] \\
\Delta_{\mathcal{S}}(\mathcal{D}_{1*}, \mathcal{D}_1) &:= \Pr[\mathcal{S}^{\mathcal{D}_{1*}} \to =] - \Pr[\mathcal{S}^{\mathcal{D}_1} \to =]
\end{aligned}$$

it will be sufficient to upper bound the latter three distinguishing advantages. (The probabilities $\Pr[\mathcal{S}^{\mathcal{D}_{0*}} \to =]$, $\Pr[\mathcal{S}^{\mathcal{D}_{1*}} \to =]$ have the obvious meanings.) The transition from $\mathcal{D}_{0*}$ to $\mathcal{D}_{1*}$ is quite trivial, while the Domain Separation Lemma will be used for the transitions from $\mathcal{D}_0$ to $\mathcal{D}_{0*}$ and from $\mathcal{D}_{1*}$ to $\mathcal{D}_1$.

To introduce this lemma, consider finite sets U, V and a function $f : U \to V$. An adversary A is given two-way oracle access to a family of permutations $\{\pi_u : u \in U\}$ indexed by U where $\pi_u : \{0,1\}^n \to \{0,1\}^n$ for each $u \in U$. I.e., A can make a queries of the form $(u, x, 1)$ or of the form $(u, y, -1)$, to be answered by $\pi_u(x)$ and $\pi_u^{-1}(y)$ respectively. We consider two different possible instantiations of the permutation family $\{\pi_u : u \in U\}$: in the "ideal world" each π_u is an independent random permutation, whereas in the "real world" $\pi_u := \tau_{f(u)}$ where $\{\tau_v : v \in V\}$ is a family of independent random permutations indexed by V. (Permutations that are different in the ideal world thus become collapsed in the real world according to f.)

We say that A *learns* a triple (u, x, y) if it makes the query $(u, x, 1)$ and this query is answered by y or if it makes the query $(u, y, -1)$ and this query is answered by x. We say that a *collision* occurs if A learns two distinct tuples (u, x, y), (u', x', y') such that $(x = x' \vee y = y')$ and such that $f(u) = f(u')$. (Necessarily, in this case, $u \neq u'$.) Then:

Lemma 2. *(Domain Separation Lemma) A's advantage at distinguishing the real and ideal worlds is upper bounded by A's probability of causing a collision in the ideal world.*

The Domain Separation Lemma is meant to be invoked with an adversary A that has a predetermined structure. In our case, e.g., A will be obtained as an agglomeration of $\mathcal{D}$ and $\mathcal{S}$, where $\mathcal{D}$ and $\mathcal{S}$ will be making queries to different halves of U by construction.

In more detail, we will apply the lemma by setting $V = \{0,1\}^n$, $U = \{0,1\}^n \times \{0,1\}$, $f((v,0)) = f((v,1)) = v$ for all $v \in \{0,1\}^n$. The set of random permutations $\{\tau_v : v \in V\}$ corresponds to the ideal cipher IC, while if each π_u is a random permutation (i.e., if we are in the ideal world) then $\{\pi_u : u \in U\}$ should be understood as the "original" IC from the real world, to which $\mathcal{D}$ makes queries, plus a "dummy" independent ideal cipher IC', to which $\mathcal{S}$ makes queries.

In other words, we will run the experiment $\mathcal{S}^{\mathcal{D}_z}$ (for $z = 0$ or $z = 1$) with $\mathcal{S}$ and $\mathcal{D}_z$ banding together to form the adversary A, while imposing the requirement that $\mathcal{S}$ makes IC queries with keys (values $u \in U$, more exactly) of the form $(v,1) \in \{0,1\}^n \times \{0,1\}$, while $\mathcal{D}_z$ makes IC queries with keys of the form $(v,0) \in \{0,1\}^n \times \{0,1\}$. In this case the real world precisely coincides with $\mathcal{S}^{\mathcal{D}_z}$ (because the permutations $\pi_{(v,1)}$, $\pi_{(v,0)}$ are collapsed to a single random permutation τ_v for each $v \in \{0,1\}^n$), whereas the ideal world precisely coincides with $\mathcal{S}^{\mathcal{D}_{z*}}$ (because $\mathcal{S}$ is given oracle access to its own "bogus" copy of IC).

To upper bound $\Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_{0*})$ (e.g.), it thus suffices to upper bound the probability that a collision occurs in the experiment $\mathcal{S}^{\mathcal{D}_{0*}}$, as defined by the Domain Separation Lemma. From the definition, specifically, a collision occurs if and only if $A = (\mathcal{S}, \mathcal{D}_0)$ learns a pair of tuples of the form $((v,0),x,y)$, $((v,1),x',y')$ for some $v \in \{0,1\}^n$ such that $(x = x' \vee y = y')$. Note that by construction, the tuple $((v,0),x,y)$ must be the result of a query made by $\mathcal{D}_0$ while the tuple $((v,1),x',y')$ must be the result of a query made by $\mathcal{S}$. Moreover $\mathcal{D}_0$ contributes at most four tuples, which are

$$((k,0),x_1,y_4),((k',0),x'_1,y'_4),((k'',0),x_1,y''_4),((k''',0),x'_1,y'''_4).$$

On the other hand, $\mathcal{S}$'s only information about the values k, k', k'', k''' used by $\mathcal{D}_0$ in this experiment comes from the values x_4, x'_4, x_1, x'_1 and y''_4, y'''_4 that $\mathcal{D}_0$ queries to $\mathcal{S}$ and from the values y_1, y'_1 returned by $\mathcal{S}$ to $\mathcal{D}$, since $\mathcal{S}$'s IC oracle is now completely pointless! However, it is easy to see that each of k, k', k'' and k''' maintains n bits of entropy (individually) subject to this information; hence, $\mathcal{S}$'s chance of causing a collision in $\mathcal{S}^{\mathcal{D}_{0*}}$ is at most $4\sigma/2^n$ by a union bound. Thus

$$\Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_{0*}) \leq \frac{4\sigma}{2^n}$$

by the Domain Separation Lemma. One also shows that

$$\Delta_{\mathcal{S}}(\mathcal{D}_1, \mathcal{D}_{1*}) \leq \frac{4\sigma}{2^n}$$

by an identical argument. (Indeed, y''_4, y'''_4 don't actually carry any information about k, k', k'' and k''' in either $\mathcal{S}^{\mathcal{D}_{0*}}$ or $\mathcal{S}^{\mathcal{D}_{1*}}$.)

It remains to upper bound the distinguishability of $\mathcal{D}_{0*}$ and $\mathcal{D}_{1*}$. Given that $\mathcal{S}$'s IC oracle is useless in each of these worlds, however, the argument is trivial:

from $\mathcal{S}$'s standpoint, y_4'' and y_4''' are distinct values sampled uniformly at random from $\{0,1\}^n \backslash \{y_4, y_4'\}$ in both worlds. Hence $\Delta(\mathcal{D}_{0*}, \mathcal{D}_{1*}) = 0$.

Combining these bounds, we find

$$\Delta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_1) \leq \frac{4\sigma}{2^n} + 0 + \frac{4\sigma}{2^n}$$

and

$$\Pr[\mathcal{D}[\mathrm{IC}, \mathcal{S}^{\mathrm{IC}}] = 1] \leq \frac{1}{2} + \frac{1}{2} \cdot \frac{8\sigma}{2^n} = \frac{1}{2} + \frac{4\sigma}{2^n}$$

and

$$\begin{aligned} \Pr\left[\mathcal{D}[\mathrm{EM}^{\mathbf{P}}, \mathbf{P}] = 1\right] - \Pr\left[\mathcal{D}[\mathrm{IC}, \mathcal{S}^{\mathrm{IC}}] = 1\right] &\geq \left(1 - \frac{7}{2^n}\right) - \left(\frac{1}{2} + \frac{4\sigma}{2^n}\right) \\ &= \frac{1}{2} - \frac{4\sigma}{2^n} - \frac{7}{2^n} \end{aligned}$$

as claimed. □

Remark 2. Say a distinguisher is *sequential* [39, 15] if it first queries only its right interface (random permutations/simulator), and then only its left interface (IEM construction/ideal cipher), but not its right interface anymore. Many "natural" attacks against indifferentiability are sequential (in particular, the attack against 5-round Feistel of [19] and the attack against 3-round IEM of [38]), running in two phases: first, the distinguisher looks for input/output pairs satisfying some relation which is hard to satisfy for an ideal cipher (a so-called "evasive" relation) by querying the right interface; then, it checks consistency of these input/output pairs by querying the left interface (since the relation is hard to satisfy for an ideal cipher, any polynomially-bounded simulator will fail to consistently simulate the inner permutations in the ideal world). We note that the attack described in this section is *not* sequential. This does not come as a surprise since Cogliati and Seurin [15] showed that the 4-round IEM construction is *sequentially* indifferentiable from an IC, i.e. indifferentiable from an IC by any sequential distinguisher. Hence, our new attack yields a natural separation between (full) indifferentiability and sequential indifferentiability.

4 The 5-Round Simulator

In this section, we describe our simulator for proving indifferentiability of the 5-round IEM construction from an ideal cipher.

4.1 Informal Description

We start with a high-level overview of how the simulator $\mathcal{S}$ works, deferring the formal description in pseudocode to Section 4.2. For each $i \in \{1, \ldots, 5\}$, the simulator maintains a pair of tables P_i and P_i^{-1} with 2^n entries containing either an n-bit value or a special symbol $\perp$, allowing the simulator to keep track

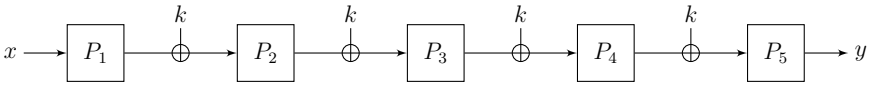

Fig. 3. The 5-round iterated Even-Mansour construction with independent permutations and identical round keys. The first and last round key additions are omitted since they do not play any role for the indifferentiability property.

of values that have already been assigned internally for the i-th permutation. Initially, these tables are empty, meaning that $P_i(x) = P_i^{-1}(y) = \perp$ for all $x, y \in \{0,1\}^n$. The simulator sets $P_i(x) \leftarrow y$, $P_i^{-1}(y) \leftarrow x$ to indicate that the i-th permutation maps x to y. The simulator never overwrites entries in P_i or P_i^{-1}, and always keeps these two tables consistent, so that P_i always encodes a "partial permutation" of $\{0,1\}^n$. We sometimes denote $x \in P_i$ (resp. $y \in P_i^{-1}$) to mean that $P_i(x) \neq \perp$ (resp. $P_i^{-1}(y) \neq \perp$).

The simulator offers a single public interface Query(i, δ, z) allowing the distinguisher to request the value $P_i(z)$ when $\delta = +$ or $P_i^{-1}(z)$ when $\delta = -$ for input $z \in \{0,1\}^n$. Upon reception of a query (i, δ, z), the simulator checks whether $P_i^{\delta}(z)$ has already been defined, and returns the corresponding value if this is the case. Otherwise, it marks query (i, δ, z) as "pending" and starts a "chain detection/completion" mechanism, called a *query cycle* in the following, in order to maintain consistency between its answers and the IC as we now explain. (We stress that some of the wording introduced here is informal and that all notions will be made rigorous in the next sections.)

We say that a triple (i, x_i, y_i) is table-defined if $P_i(x_i) = y_i$ and $P_i^{-1}(y_i) = x_i$ (that is, the simulator internally decided that x_i is mapped to y_i by permutation P_i). Let us informally call a tuple of $j - i + 1 \geq 2$ table-defined permutation queries at adjacent positions $((i, x_i, y_i), \ldots, (j, x_j, y_j))$ (indices taken mod 5) such that $x_{i+1} = y_i \oplus k$ if $i \neq 5$ and $x_{i+1} = \mathrm{IC}^{-1}(k, y_i)$ if $i = 5$ a "k-path of length $j + i - 1$" (hence, paths might "wrap around" the IC).

The very simple idea at the heart of the simulator is that, before answering any distinguisher's query to some simulated permutation, it ensures that any path of length three (or more) has been preemptively extended to a "complete" path of length five $((1, x_1, y_1), \ldots, (5, x_5, y_5))$ compatible with the ideal cipher (i.e., such that $\mathrm{IC}(k, x_1) = y_5$). For this, assume that at the moment the distinguisher makes a permutation query (i, δ, z) which is not table-defined yet (otherwise the simulator just returns the answer that was preemptively set), any path of length three is complete. This means that any existing incomplete path has length at most two. These length-2 paths will be called (table-defined[3]) *2chains* in the main body of the proof, and will play a central role. For ease of the discussion to come, let us call the pair of adjacent positions $(i, i+1)$ of the table-defined queries constituting a 2chain the *type* of the 2chain. (Note that as any path, a

[3] While the difference between a table-defined and table-undefined 2chain will be important in the formal proof, we ignore this subtlety for the moment.

2chain can "wrap around", i.e., consists of two table-defined queries $(5, x_5, y_5)$ and $(1, x_1, y_1)$ such that $\mathrm{IC}(k, x_1) = y_5$, so that possible types are $(1, 2)$, $(2, 3)$, $(3, 4)$, $(4, 5)$, and $(5, 1)$). Let us also call the direct input to permutation P_{i+2} and the inverse input to permutation P_{i-1} when extending the 2chain in the natural way the resp. right and left *endpoint* of the 2chain.[4]

The "pending" permutation query (i, δ, z) asked by the distinguisher, once answered by the simulator, might create new incomplete paths of length three when combined with adjacent 2chains, that is, 2chains at position $(i-2, i-1)$ for a direct query $(i, +, x_i)$ or 2chains at position $(i+1, i+2)$ for a inverse query $(i, -, y_i)$. Hence, just after having marked the initial query of the distinguisher as "pending", the simulator immediately detects all 2chains that will form a length-3 path with this pending query, and marks them as "triggered". Following the high-level principle of completing any length-3 path, any triggered 2chain should (by the end of the query cycle) be extended to a complete path, which might create new incomplete length-3 paths.

To ease the discussion, let us slightly change the notation and assume that the distinguisher's query that initiated the query cycle was either a direct query $(i+2, +, x_{i+2})$ or an inverse query $(i-1, -, y_{i-1})$. In both cases, adjacent 2chains that might get "triggered" are of type $(i, i+1)$. For each 2chain which was triggered by the initial query of distinguisher, the simulator computes its endpoint opposite to the initial query, and marks it as pending as well. Note that if the distinguisher's query was $(i+2, +, x_{i+2})$, then all these new pending queries are of the form $(i-1, -, \cdot)$, while if it was $(i-1, -, y_{i-1})$, all the new pending queries are of the form $(i+2, +, \cdot)$. For each of these new pending queries, the simulator recursively detects whether they form a length-3 path with other $(i, i+1)$-2chains, and triggers these 2chains. Hence, if the initiating query of the distinguisher was of the form $(i+2, +, \cdot)$ or $(i-1, -, \cdot)$, any triggered 2chain will be of type $(i, i+1)$. For this reason, we say such a query cycle is of type $(i, i+1)$. Note also that all pending queries will be of the form $(i+2, +, \cdot)$ or $(i-1, -, \cdot)$ The recursive detection process continues until there is no new $(i, i+1)$-2chain to detect and trigger. Note that the simulator can completely determine which 2chains should be triggered *before* starting the completion process itself.

Once all 2chains that must eventually be completed have been detected and "triggered", the simulator starts the completion process. First, it randomly samples all "pending" queries (which are necessarily of the form $(i+2, +, \cdot)$ or $(i-1, -, \cdot)$ for a query cycle of type $(i, i+1)$). This "all-at-the-same-time" randomness sampling is reminiscent of the simulator in [23]. Finally, for all triggered 2chains, it adapts the corresponding path by computing the corresponding input x_{i+3} and output y_{i+3} at position $i+3$ and "forcing" $P_{i+3}(x_{i+3}) = y_{i+3}$. In case some collision occurs when trying to assign a value for some permutation, the simulator aborts. All important characteristics of an $(i, i+1)$-query cycle are summarized in Table 1.

[4] Again, there is a slight subtlety for the left endpoint of a $(1, 2)$-2chain and the right endpoint of a $(4, 5)$-2chain since this involves the ideal cipher, but we ignore it here.

Table 1. The five types of $(i, i+1)$-query cycles of the simulator.

Type $(i, i+1)$	Initiating query/ReadTape call type $(i-1,-)$ and $(i+2,+)$	Adapt at $i+3$
(1,2)	$(5,-)$ and $(3,+)$	4
(2,3)	$(1,-)$ and $(4,+)$	5
(3,4)	$(2,-)$ and $(5,+)$	1
(4,5)	$(3,-)$ and $(1,+)$	2
(5,1)	$(4,-)$ and $(2,+)$	3

4.2 Pseudocode of the Simulator and Game Transitions

We now give the full pseudocode for the simulator, and by the same occasion fully describe the intermediate worlds that will be used in the indifferentiability proof. The distinguisher $\mathcal{D}$ has access to the public interface $\text{Query}(i, \delta, z)$, which in the ideal world is answered by the simulator, and to the ideal cipher/IEM construction interface, that we formally capture with two interfaces $\text{Enc}(k, x)$ and $\text{Dec}(k, y)$ for encryption and decryption respectively. We will refer to queries to any of these two interfaces as *cipher queries*, by opposition to *permutation queries* made to interface $\text{Query}(\cdot, \cdot, \cdot)$. In the ideal world, cipher queries are answered by an ideal cipher IC. We make the randomness of IC explicit through two random tapes ic, $\mathsf{ic}^{-1} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ such that for any $k \in \{0,1\}^n$, $\mathsf{ic}(k, \cdot)$ is a uniformly random permutation and $\mathsf{ic}^{-1}(k, \cdot)$ is its inverse. Hence, in the ideal world, a query $\text{Enc}(k, x)$, resp. $\text{Dec}(k, y)$, is simply answered with $\mathsf{ic}(k, x)$, resp. $\mathsf{ic}^{-1}(k, y)$. The randomness used by the simulator for lazily sampling permutations $P_1, \ldots, P_5$ when needed is also made explicit in the pseudocode through uniformly random permutations tapes $\mathbf{p} = (p_1, p_1^{-1}, \ldots, p_5, p_5^{-1})$. By random permutation tapes, we mean that $p_i : \{0,1\}^n \to \{0,1\}^n$ is a uniformly random permutation and p_i^{-1} is its inverse. Hence, randomness in game G_1 is fully captured by ic and $\mathbf{p}$.

Since we will use two intermediate games, the real world will be denoted G_4. In this world, queries to $\text{Query}(i, \delta, z)$ are simply answered with the corresponding value stored in the random permutation tapes $\mathbf{p}$, while queries to Enc or Dec are answered by the IEM construction based on random permutations $\mathbf{p}$. Randomness in G_4 is fully captured by $\mathbf{p}$.

Intermediate Games. The indifferentiability proof relies on two intermediate games G_2 and G_3. In game G_2, the Check procedure used by the simulator to detect new external chains is modified such that it does not make explicit queries to the ideal cipher; instead it first checks to see if the entry exists in table T recording cipher queries and if not, returns false. In game G_3, the ideal cipher is replaced with the 5-round IEM construction that uses the same random permutation tapes $\mathbf{p}$ as the simulator (and hence both the distinguisher *and* the simulator interact with the 5-round IEM construction instead of the IC).

Summing up, randomness is fully captured by ic and $\mathbf{p}$ in games G_1 and G_2, and by $\mathbf{p}$ in games G_3 and G_4 (since the ideal cipher is replaced by the IEM construction $\mathrm{EM}^{\mathbf{P}}$ when transitioning from G_2 to G_3).

Notes about the Pseudocode. The pseudocode for the public (i.e., accessible by the distinguisher) procedures Query, Enc, and Dec is given in Fig. 4, together with helper procedures that capture the changes from games G_1 to G_4. The pseudocode for procedures that are internal to the simulator is given in Fig. 5. Lines commented with "$\backslash\backslash\mathsf{G}_i$" apply only to game G_i. In the pseudocode and more generally throughout this paper, the result of arithmetics on positions is automatically wrapped into the range $\{1, 2, 3, 4, 5\}$. For any table or tape $\mathcal{T}$ and $\delta \in \{+, -\}$, we let $\mathcal{T}^\delta$ be $\mathcal{T}$ if $\delta = +$ and be $\mathcal{T}^{-1}$ if $\delta = -$. Given a list L, $L \hookleftarrow x$ means that x is appended to L. If the simulator aborts (line 86), we assume it returns a special symbol $\perp$ in response to the distinguisher's query being processed.

Tables T and T^{-1} are used to record the cipher queries that have been issued (by the distinguisher *or* the simulator). Note that tables P_i and P_i^{-1} are modified only by procedure Assign. The table entries are never overwritten, due to the check at line 86.

5 Proof of Indifferentiability

5.1 Main Result and Proof Structure

Our main result is the following theorem.

Theorem 3. *The 5-round iterated Even-Mansour construction* $\mathrm{EM}^{\mathbf{P}}$ *with random permutations* $\mathbf{P} = (P_1, \ldots, P_5)$ *is* (t_S, q_S, ϵ)*-indifferentiable from an ideal cipher with* $t_S = O(q^5)$*,* $q_S = O(q^5)$ *and* $\varepsilon = O(q^{38}/2^n)$*.*

Moreover, the bounds hold even if the distinguisher is allowed to make q *permutation queries in each position (i.e., it can call* $\mathrm{Query}(i, *, *)$ q *times for each* $i \in \{1, 2, 3, 4, 5\}$*) and make* q *cipher queries (i.e.,* Enc *and* Dec *can be called* q *times in total).*

Proof. We use the indifferentiability simulator described in Section 4. The upper bounds on the simulator's running time and query complexity are given in Theorem 42. The upper bound on the distinguisher's distinguishing advantage is given in Theorem 52. □

Proof Structure. Our proof uses a sequence of games G_1, G_2, G_3 and G_4 as described in Section 4.2, with G_1 being the simulated world and G_4 being the real world.

Throughout the proof we will fix an arbitrary information-theoretic distinguisher $\mathcal{D}$ with q queries, meaning that $\mathcal{D}$ has unlimited computation power and can issue a limited number of queries. Using a trick in [20], we allow the distinguisher $\mathcal{D}$ to make q cipher queries and q permutation queries in *each* position,

Game $\mathsf{G}_i(\mathsf{ic}, \mathbf{p})$, $i = 1, 2$ / $\mathsf{G}_i(\mathbf{p})$, $i = 3, 4$

Variables:
  Tables of cipher queries T, T^{-1}
  Tables of defined permutation queries P_i, P_i^{-1}, $i \in \{1, \ldots, 5\}$
  Ordered list of pending queries $\mathsf{Pending}$
  Ordered list of triggered paths $\mathsf{Triggered}$

public procedure Query(i, δ, z):
  return SimQuery(i, δ, z) \\ $\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3$
  return $p_i^{\delta}(z)$ \\ G_4

public procedure Enc(k, x_1):
  if $T(k, x_1) = \perp$ **then**
    $y_5 \leftarrow \mathsf{ic}(k, x_1)$ \\ $\mathsf{G}_1, \mathsf{G}_2$
    $y_5 \leftarrow \mathrm{EM}(k, x_1)$ \\ $\mathsf{G}_3, \mathsf{G}_4$
    $T(k, x_1) \leftarrow y_5$, $T^{-1}(k, y_5) \leftarrow x_1$
  return $T(k, x_1)$

private procedure EM(k, x_1):
  for $i = 1$ **to** 4 **do**
    $x_{i+1} = p_i(x_i) \oplus k$
  return $p_5(x_5)$

public procedure Dec(k, y_5):
  if $T^{-1}(k, y_5) = \perp$ **then**
    $x_1 \leftarrow \mathsf{ic}^{-1}(k, y_5)$ \\ $\mathsf{G}_1, \mathsf{G}_2$
    $x_1 \leftarrow \mathrm{EM}^{-1}(k, y_5)$ \\ $\mathsf{G}_3, \mathsf{G}_4$
    $T(k, x_1) \leftarrow y_5$, $T^{-1}(k, y_5) \leftarrow x_1$
  return $T^{-1}(k, y_5)$

private procedure $\mathrm{EM}^{-1}(k, y_5)$:
  for $i = 5$ **to** 2 **do**
    $y_{i-1} = p_i^{-1}(y_i) \oplus k$
  return $p_1^{-1}(y_1)$

private procedure Check(k, x_1, y_5):
  return $\mathrm{Enc}(k, x_1) = y_5$ \\ G_1
  return $T(k, x_1) = y_5$ \\ $\mathsf{G}_2, \mathsf{G}_3, \mathsf{G}_4$

Fig. 4. Public procedures Query, Enc, and Dec for games G_1 - G_4, and helper procedures EM, EM^{-1}, and Check. This set of procedures captures all changes from game G_1 to G_4, namely: from game G_1 to G_2 only procedure Check is modified; from game G_2 to G_3, the only change is in procedures Enc and Dec where the ideal cipher is replaced by the IEM construction; and from game G_3 to G_4, only procedure Query is modified to return directly the value read in random permutation tables $\mathbf{p}$.

```
33 private procedure SimQuery(i, δ, z):
34   if P_i^δ(z) = ⊥ then
35     Pending ← ((i, δ, z)), Triggered ← ∅
36     forall (i, δ, z) in Pending do FindNewPaths(i, δ, z)
37     forall (i, δ, z) in Pending do ReadTape(i, δ, z)
38     forall (i, i + 1, y_i, x_{i+1}, k) in Triggered do AdaptPath(i, i + 1, y_i, x_{i+1}, k)
39   return P_i^δ(z)

40 private procedure FindNewPaths(i, δ, z):
41 case (δ = +):
42 x_i ← z
43 forall (x_{i−2}, x_{i−1}) in (P_{i−2}, P_{i−1}) do
44   y_{i−2} ← P_{i−2}(x_{i−2}), y_{i−1} ← P_{i−1}(x_{i−1})
45   if i = 2 then k ← y_{i−1} ⊕ x_i
46   else k ← y_{i−2} ⊕ x_{i−1}
47   C ← (i − 2, i − 1, y_{i−2}, x_{i−1}, k)
48   if C ∈ Triggered then continue
49   case i ∈ {1, 2}:
50     if ¬Check(k, x_1, y_5) then
51       continue
52   case i ∈ {3, 4, 5}:
53     if Next(i − 1, y_{i−1}, k) ≠ x_i then
54       continue
55   Triggered ↩ C
56   y_{i−3} ← Prev(i − 2, x_{i−2}, k)
57   if (i − 3, −, y_{i−3}) ∉ Pending then
58     Pending ↩ (i − 3, −, y_{i−3})
59 case (δ = −):
60 y_i ← z
61 forall (x_{i+1}, x_{i+2}) in (P_{i+1}, P_{i+2}) do
62   y_{i+1} ← P_{i+1}(x_{i+1}), y_{i+2} ← P_{i+2}(x_{i+2})
63   if i = 4 then k ← y_i ⊕ x_{i+1}
64   else k ← y_{i+1} ⊕ x_{i+2}
65   C ← (i + 1, i + 2, y_{i+1}, x_{i+2}, k)
66   if C ∈ Triggered then continue
67   case i ∈ {4, 5}:
68     if ¬Check(k, x_1, y_5) then
69       continue
70   case i ∈ {1, 2, 3}:
71     if Prev(i + 1, x_{i+1}, k) ≠ y_i then
72       continue
73   Triggered ↩ C
74   x_{i+3} ← Next(i + 2, y_{i+2}, k)
75   if (i + 3, +, x_{i+3}) ∉ Pending then
76     Pending ↩ (i + 3, +, x_{i+3})

77 private procedure ReadTape(i, δ, z):
78   if δ = + then Assign(i, z, p_i(z)) else Assign(i, p_i^{−1}(z), z)

79 private procedure AdaptPath(i, i + 1, y_i, x_{i+1}, k):
80   y_{i+1} ← P_{i+1}(x_{i+1}), x_{i+2} ← Next(i + 1, y_{i+1}, k), y_{i+2} ← P_{i+2}(x_{i+2})
81   x_{i+3} ← Next(i + 2, y_{i+2}, k)
82   x_i ← P_i^{−1}(y_i), y_{i−1} ← Prev(i, x_i, k), x_{i−1} ← P_{i−1}^{−1}(y_{i−1})
83   y_{i−2} ← Prev(i − 1, x_{i−1}, k)
84   Assign(i + 3, x_{i+3}, y_{i−2}) \\ subscripts are equal because of the wrapping

85 private procedure Assign(i, x_i, y_i):
86   if P_i(x_i) ≠ ⊥ or P_i^{−1}(y_i) ≠ ⊥ then abort
87   P_i(x_i) ← y_i, P_i^{−1}(y_i) ← x_i

88 private procedure Next(i, y_i, k):
89   if i = 5 then return Dec(k, y_i)
90   else return y_i ⊕ k
91 private procedure Prev(i, x_i, k):
92   if i = 1 then return Enc(k, x_i)
93   else return x_i ⊕ k
```

Fig. 5. Private procedures used by the simulator.

as described in Theorem 3. This trick allows us to obtain a better security bound, even though the distinguisher can make more queries than usual.[5] We can assume without loss of generality that $\mathcal{D}$ is *deterministic*, as any distinguisher can be derandomized using the "optimal" random tape and achieve at least the same advantage.

Without loss of generality, we assume that $\mathcal{D}$ outputs 1 with higher probability in the simulated world G_1 than in the real world G_4. We define the *advantage* of $\mathcal{D}$ in distinguishing between G_i and G_j by

$$\Delta_{\mathcal{D}}(\mathsf{G}_i, \mathsf{G}_j) := \Pr_{\mathsf{G}_i}[\mathcal{D}^{\text{Query,Enc,Dec}} = 1] - \Pr_{\mathsf{G}_j}[\mathcal{D}^{\text{Query,Enc,Dec}} = 1].$$

Our goal is to upper bound $\Delta_{\mathcal{D}}(\mathsf{G}_1, \mathsf{G}_4)$.

Our proof starts with discussions about the game G_2, which will be the starting point of the transitions. As usual, there are *bad events* that might cause the simulator to fail. We will prove that bad events are unlikely, and show properties of *good executions* in which bad events don't occur. The proof of efficiency of the simulator (in good executions of G_2) is the most interesting and technical part of this paper, which is given in Section 5.4. During the proof of efficiency we also obtain upper bounds on the sizes of the tables and on the number of calls to each procedure, which will be a crucial component for the transitions.

For the G_1-G_2 transition, note that the only difference between the two games is in Check. If the simulator is efficient, the probability that the two executions diverge in a call to Check is negligible. Therefore, if an execution of G_2 is good, it is identical to the G_1-execution with the same random tapes except with negligible probability (cf. Lemma 40). In particular, this implies that an execution of G_1 is efficient with high probability.

For the G_2-G_3 transition, we use a standard randomness mapping argument. We will map the randomness of good executions of G_2 to the randomness of non-aborting executions of G_3, so that the G_3-executions with the mapped randomness are identical to the G_2-executions with the preimage randomness.

We will show that if a G_3-execution has a preimage, then the answers of the permutation queries output by the simulator must be compatible with the random permutation tapes (cf. Lemma 49). Thus the G_3-execution is identical to the G_4-execution with the same random tapes, where the permutation queries are answered by the corresponding entries of the random tapes. This enables a transition directly from G_2 to G_4 using the randomness mapping, which is a small novelty of our proof.

[5] In the randomness mapping, we will need to convert an arbitrary distinguisher to one that "completes all paths". If the distinguisher is allowed q queries in total, the number of queries will become $6q$; if D is allowed q queries in each position, it only increases from q to $2q$. Moreover, the proof works almost the same for the stronger version of distinguishers.

5.2 Good Executions of G_2: Definitions and Basic Properties

In this section, we will define a set of bad events that may occur in G_2. We will refer to executions of G_2 where these events do not occur as good executions. Later on, we will establish some properties of these good executions. In particular, we will prove that the simulator does not abort and runs in polynomial time in good executions of G_2.

5.2.1 Notation and Definitions

Before we define the bad events, we introduce some notation and definitions.

QUERIES AND 2CHAINS. The central notion for reasoning about the simulator is the notion of 2chain, that we develop below.

Definition 2. A *permutation query* is a triple (i, δ, z) where $1 \leq i \leq 5$, $\delta \in \{+, -\}$ and $z \in \{0,1\}^n$. We call i the *position* of the query, δ the *direction* of the query, and the pair (i, δ) the *type* of the query.

Definition 3. A *cipher query* is a triple (δ, k, z) where $\delta \in \{+, -\}$ and $k, z \in \{0,1\}^n$. We call δ the *direction* and k the *key* of the cipher query.

Definition 4. A permutation query (i, δ, z) is *table-defined* if $P_i^{\delta}(z) \neq \perp$, and *table-undefined* otherwise. Similarly, a cipher query (δ, k, z) is *table-defined* if $T^{\delta}(k, z) \neq \perp$, and *table-undefined* otherwise.

For permutation queries, we sometimes omit i and δ when they are clear from the context and simply say that x_i, resp. y_i, is table-(un)defined to mean that $(i, +, x_i)$, resp. $(i, -, y_i)$, is table-(un)defined.

Note that if $(i, +, x_i)$ is table-defined and $P_i(x_i) = y_i$, then necessarily $(i, -, y_i)$ is also table-defined and $P_i^{-1}(y_i) = x_i$. Indeed, tables P_i and P_i^{-1} are only modified in procedure Assign, where existing entries are never overwritten due to the check at line 86. Thus the two tables always encode a partial permutation and its inverse, i.e., $P_i(x_i) = y_i$ if and only if $P_i^{-1}(y_i) = x_i$. Hence, we often say that a triple (i, x_i, y_i) is table-defined as a shorthand to mean that both $(i, +, x_i)$ and $(i, -, y_i)$ are table-defined with $P_i(x_i) = y_i$ and $P_i^{-1}(y_i) = x_i$.

Similarly, if a cipher query $(+, k, x)$ is table-defined and $T(k, x) = y$, then necessarily $(-, k, y)$ is table-defined and $T^{-1}(k, y) = x$. Indeed, these tables are only modified by calls to Enc/Dec, and always according to the IC tape ic, hence these two tables always encode a partial cipher and its inverse, i.e. $T(k, x) = y$ if and only if $T^{-1}(k, y) = x$. Hence, we often say that a triple (k, x, y) is table-defined as a shorthand to mean that both $(+, k, x)$ and $(-, k, y)$ are table-defined with $T(k, x) = y$ and $T^{-1}(k, y) = x$.

Definition 5 (2chain)**.** An *inner 2chain* is a tuple $(i, i+1, y_i, x_{i+1}, k)$ such that $i \in \{1, 2, 3, 4\}$, $y_i, x_{i+1} \in \{0,1\}^n$, and $k = y_i \oplus x_{i+1}$. A *(5,1)-2chain* is a tuple $(5, 1, y_5, x_1, k)$ such that $y_5, x_1, k \in \{0,1\}^n$. A $(i, i+1)$-2chain refers either to an inner or a $(5,1)$-2chain, and is denoted generically $(i, i+1, y_i, x_{i+1}, k)$, $i \in \{1, \ldots, 5\}$, the second element $i+1$ being taken mod 5. We call $(i, i+1)$ the *type* of the 2chain.

Remark 3. Note that for a 2chain of type $(i, i+1)$ with $i \in \{1,2,3,4\}$, given y_i and x_{i+1}, there is a unique key k such that $(i, i+1, y_i, x_{i+1}, k)$ is a 2chain (hence k is "redundant" in the notation), while for a 2chain of type $(5,1)$, the key might be arbitrary. This convention allows to have a unified notation independently of the type of the 2chain. See also Remark 4 below.

Definition 6. An inner 2chain $(i, i+1, y_i, x_{i+1}, k)$ is said *table-defined* if both $(i, -, y_i)$ and $(i+1, +, x_{i+1})$ are table-defined permutation queries, and *table-undefined* otherwise. A (5,1)-2chain $(5, 1, y_5, x_1, k)$ is said table-defined if both $(5, -, y_5)$ and $(1, +, x_1)$ are table-defined permutation queries *and* $T(k, x_1) = y_5$, and *table-undefined* otherwise.

Remark 4. Our definitions above ensure that whether a tuple $(i, i+1, y_i, x_{i+1}, k)$ is a 2chain or not is independent of the state of tables P_i/P_i^{-1} and T/T^{-1}. Only the fact that a 2chain is table-defined or not depends on these tables.

Definition 7 (endpoints)**.** Let $C = (i, i+1, y_i, x_{i+1}, k)$ be a table-defined 2chain. The *right endpoint* of C, denoted $r(C)$ is defined as

$$\begin{aligned} r(C) &= P_{i+1}(x_{i+1}) \oplus k && \text{if } i \in \{1,2,3,5\} \\ &= T^{-1}(k, P_5(x_5)) && \text{if } i = 4 \text{ and } (-, k, P_5(x_5)) \text{ is table-defined} \\ &= \perp && \text{if } i = 4 \text{ and } (-, k, P_5(x_5)) \text{ is table-undefined.} \end{aligned}$$

The *left endpoint* of C, denoted $\ell(C)$ is defined as

$$\begin{aligned} \ell(C) &= P_i^{-1}(y_i) \oplus k && \text{if } i \in \{2,3,4,5\} \\ &= T(k, P_1^{-1}(y_1)) && \text{if } i = 1 \text{ and } (+, k, P_1^{-1}(y_1)) \text{ is table-defined} \\ &= \perp && \text{if } i = 1 \text{ and } (+, k, P_1^{-1}(y_1)) \text{ is table-undefined.} \end{aligned}$$

We say that an endpoint is *dummy* when it is equal to $\perp$, and *non-dummy* otherwise. Hence, only the right endpoint of a 2chain of type $(4,5)$ or the left endpoint of a 2chain of type $(1,2)$ might be dummy.

When this is clear from the context, we sometimes identify the right and left (non-dummy) endpoints of an $(i, i+1)$-2chain C with the corresponding permutation queries $(i+2, +, r(C))$ and $(i-1, -, \ell(C))$. In particular, when we say that $r(C)$, resp. $\ell(C)$ is table-defined, this implicitly means that it is non-dummy and the corresponding permutation query is table-defined. More importantly, when we say that one of the endpoints of C is table-undefined, we also implicitly mean that it is non-dummy. (Hence, an endpoint must be either dummy, or table-undefined, or table-defined).

Complete Path. Another useful concept is the one of "complete path".

Definition 8. A *complete path* with key k is a 5-tuple of table-defined permutation queries $((1, x_1, y_1), \ldots, (5, x_5, y_5))$ such that

$$y_i \oplus x_{i+1} = k \text{ for } i = 1,2,3,4 \text{ and } T(k, x_1) = y_5. \tag{4}$$

The five table-defined queries (i, x_i, y_i) and the five table-defined 2chains $(i, i+1, y_i, x_{i+1}, k)$, $i \in \{1, \ldots, 5\}$, are said to *belong to the complete path.*

When a 2chain C belongs to a complete path, we sometimes simply say that C is *complete*. Note that if a 2chain C is complete, then $r(C)$ and $\ell(C)$ are non-dummy and table-defined. We also have the following simple but important observation.

Lemma 4. *In any execution of* G_2*, any 2chain belongs to at most one complete path.*

Proof. This follows directly from the fact that tables P_i/P_i^{-1} always encode a partial permutation and that tables T/T^{-1} always encode a partial cipher. □

Query Cycles. When the distinguisher makes a permutations query (i, δ, z) which is already table-defined, the simulator returns the answer immediately. The definition below introduces some vocabulary related to what happens within the simulator when the distinguisher makes a permutation query which is table-undefined.

Definition 9 (query cycle)**.** A *query cycle* is the period of execution between when the distinguisher issues a permutation query (i_0, δ_0, z_0) which is table-undefined and when the answer to this query is returned by the simulator. We call (i_0, δ_0, z_0) the *initiating query* of the query cycle.

A query cycle is called an $(i, i+1)$*-query cycle* if the initiating query is of type $(i-1, -)$ or $(i+2, +)$ (see Lemma 5 (a) and Table 1).

The portion of the query cycle consisting of calls to FindNewPaths at line 36 is called the *detection phase* of the query cycle; the portion of the query cycle consisting of calls to ReadTape at line 37 and to AdaptPath at line 38 is called the *completion phase* of the query cycle.

During a query cycle, we say that a permutation query (i, δ, z) is *pending* (or simply that z is pending when i and δ are clear from the context) if it is appended by the simulator to list Pending at line 35, 58, or 76. We say that a 2chain $C = (i, i+1, y_i, x_{i+1}, k)$ is *triggered* if the simulator appends C to the list Triggered at line 55 or 73.

The lemma below gives some basic properties of query cycles that will be used throughout the indifferentiability proof. Part (a) justifies the name "$(i, i+1)$-query cycle".

Lemma 5. *During an* $(i, i+1)$*-query cycle whose initiating query was* (i_0, δ_0, z_0)*, the following properties always hold:*

(a) Only 2chains of type $(i, i+1)$ *are triggered.*
(b) Only permutations queries of type $(i-1, -)$ *or* $(i+2, +)$ *are pending.*
(c) Any 2chain that is triggered was table-defined at the beginning of the query cycle.
(d) At the end of the detection phase, any pending query is either the initiating query, or the endpoint of a triggered 2chain.
(e) If a 2chain C *is triggered during the query cycle, and the simulator does not abort, then* C *is complete at the end of the query cycle.*

Proof. The proof of (a) and (b) proceeds by inspection of the pseudocode: note that calls to FindNewPaths$(i-1,-,\cdot)$ can only add 2chains of type $(i,i+1)$ to Triggered and permutations queries of type $(i+2,+)$ to Pending, whereas calls to FindNewPaths$(i+2,+,\cdot)$ can only add 2chains of type $(i,i+1)$ to Triggered and permutations queries of type $(i-1,-)$ to Pending. Hence, if the initiating query is of type $(i-1,-)$ or $(i+2,+)$, only 2chains of type $(i,i+1)$ will ever be added to Triggered, and only permutation queries of type $(i-1,-)$ or $(i+2,+)$ will ever be added to Pending. The proof of (c) also follows easily from inspection of the pseudocode. The sole subtlety is to note that for a $(5,1)$-query cycle (where calls to FindNewPaths are of the form $(2,+,\cdot)$ and $(4,-,\cdot)$), for a $(5,1)$-2chain to be triggered one must obviously have $x_1 \in P_1$ and $y_5 \in P_5^{-1}$, but also $T(k,x_1) = y_5$ since otherwise the call to Check(k,x_1,y_5) would return false. The proof of (d) is also immediate, since for a permutation query to be added to Pending, it must be either the initiating query, or computed at line 56 or line 74 as the endpoint of a triggered 2chain. Finally, the proof of (e) follows from the fact that, assuming the simulator does not abort, all values computed during the call to AdaptPath(C) form a complete path to which C obviously belongs. □

The following lemma analyzes how tables T/T^{-1} are modified during a query cycle and will be helpful for the proof.

Lemma 6. *In any execution of* G_2*, the following properties hold:*

(a) During a $(1,2)$*-query cycle, tables* T/T^{-1} *are only modified during the detection phase by calls to* Enc$(\cdot,\cdot)$ *resulting from calls to* Prev$(1,\cdot,\cdot)$ *at line 56.*
(b) During a $(2,3)$*-query cycle, tables* T/T^{-1} *are only modified during the completion phase by calls to* Enc$(\cdot,\cdot)$ *resulting from calls to* Prev$(1,\cdot,\cdot)$ *at line 83.*
(c) During a $(3,4)$*-query cycle, tables* T/T^{-1} *are only modified during the completion phase by calls to* Dec$(\cdot,\cdot)$ *resulting from calls to* Next$(5,\cdot,\cdot)$ *at line 81.*
(d) During a $(4,5)$*-query cycle, tables* T/T^{-1} *are only modified during the detection phase by calls to* Dec$(\cdot,\cdot)$ *resulting from calls to* Next$(5,\cdot,\cdot)$ *at line 74.*
(e) During a $(5,1)$*-query cycle, tables* T/T^{-1} *are not modified.*

Proof. This follows by inspection of the pseudocode. The only non-trivial point concerns $(1,2)$-, resp. $(4,5)$-query cycles, since Prev$(1,\cdot,\cdot)$, resp. Next$(5,\cdot,\cdot)$ are also called during the completion phase, but a moment of thinking should make it clear that they are always called with arguments (x_1,k), resp. (y_5,k) which were previously used during the detection phase, so that this cannot modify tables T/T^{-1} any more. □

We also introduce the following helper lemma.

Lemma 7. *Consider any execution of* G_2*. Assume that two table-defined* $(i,i+1)$*-2chains* $C = (i,i+1,y_i,x_{i+1},k)$ *and* $C' = (i,i+1,y'_i,x'_{i+1},k')$ *have the same key and a common non-dummy endpoint, i.e., are such that* $k = k'$ *and* $r(C) = r(C') \neq \perp$ *or* $\ell(C) = \ell(C') \neq \perp$*. Then* $C = C'$*.*

Proof. We show the result for the case where $k = k'$ and $r(C) = r(C')$, the case where $\ell(C) = \ell(C')$ is similar. Consider first the case where $i \in \{1, 2, 3, 5\}$. By definition of the right endpoint, this implies that $P_{i+1}(x_{i+1}) = P_{i+1}(x'_{i+1})$ and hence $x_{i+1} = x'_{i+1}$ since P_{i+1} always encodes a partial permutation. It follows that $y_i = x_{i+1} \oplus k = x'_{i+1} \oplus k' = y'_i$ if $i \in \{1, 2, 3\}$, and $y_i = T(k, x_{i+1}) = T(k', x'_{i+1}) = y'_i$ if $i = 5$, and hence $C = C'$. Consider now the case $i = 4$, and let $C = (4, 5, y_4, x_5, k)$ and $C' = (4, 5, y'_4, x'_5, k')$. By assumption, $r(C) = r(C') \neq \perp$. Then, by definition of the right endpoint, $P_5(x_5) = T(k, r(C)) = T(k', r(C')) = P_5(x'_5)$, which implies that $x_5 = x'_5$ since P_5 always encodes a partial permutation. It follows that $y_4 = x_5 \oplus k = x'_5 \oplus k' = y'_4$ and hence $C = C'$. □

5.2.2 Bad Events

We now define some bad events that may happen during an execution of G_2.

Definition 10. Consider a permutation query (i_0, δ_0, z_0) or a cipher query (δ_0, k_0, z_0) made by the distinguisher. Let $\mathcal{H}'$ be the multiset of all n-bit strings appearing in the list of table-defined permutation or cipher queries and of all keys and non-dummy endpoints of any table-defined 2chain when the query occurs. That is, write the list of all table-defined permutation queries (i, x_i, y_i), all table-defined cipher queries (δ, k, z), all keys and non-dummy endpoints of all table-defined 2chains, and count each n-bit string as many times as it appears in this list. Then we define the "history" with respect to a permutation query (i_0, δ_0, z_0) made by the distinguisher as the multiset

$$\mathcal{H} \coloneqq \mathcal{H}' \cup \{z_0\},$$

and the "history" with respect to a cipher query (δ_0, k_0, z_0) made by the distinguisher as the multiset

$$\mathcal{H} \coloneqq \mathcal{H}' \cup \{k_0, z_0\}.$$

When we talk of the history with respect to a query cycle, we mean the history with respect to its initiating permutation query.

Remark 5. The sets in the above definition are time-dependent and do not include the values added to the tables *during* the query cycle or due to the distinguisher's cipher query (except z_0 for a permutation query and k_0 and z_0 for a cipher query). Note also that keys and non-dummy endpoints of table-defined 2chains can *always* be expressed as the xor of at most three n-bit values appearing in the list of table-defined permutation of cipher queries, so that strictly speaking the set $\mathcal{H}'$ could consist of these values only. However, the current definition of the history simplifies the discussion along the proof.

Definition 11. Given a query cycle, we denote $\mathcal{P}$ the multiset of random values read by ReadTape on tapes $(p_1, p_1^{-1}, \ldots, p_5, p_5^{-1})$ in the current query cycle. Given a query cycle or a distinguisher's cipher query, we denote $\mathcal{C}$ the multiset of random values read by Enc and Dec on tapes ic or ic^{-1}.[6]

[6] For a query cycle, these Enc/Dec queries are made by the simulator, while for a distinguisher's cipher query, a single call to Enc or Dec is made by the distinguisher.

We note that $\mathcal{P}$ and $\mathcal{C}$ are multisets because two randomly sampled values might turn out to be equal. However, this is unlikely to occur (and it is a "bad event" as defined below).

Definition 12. Let $\mathcal{H}^{\oplus i}$ be the set of values equal to the exclusive-or of exactly i *distinct* elements in $\mathcal{H}$, and let $\mathcal{H}^{\oplus 0} := \{0\}$. The sets $\mathcal{P}^{\oplus i}$ and $\mathcal{C}^{\oplus i}$ are defined similarly.[7]

Definition 13. BadPerm is the event that the exclusive-or of i distinct elements of $\mathcal{P}$ equals the exclusive-or of j distinct elements of $\mathcal{H}$, i.e. $\mathcal{P}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$, with $i \geq 1$, $j \geq 0$, and $i + j \leq 8$.

Definition 14. BadIC is the event that the exclusive-or of i distinct elements of $\mathcal{C}$ equals the exclusive-or of j distinct elements of $\mathcal{H}$, i.e. $\mathcal{C}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$, with $i \geq 1$, $j \geq 0$, and $i + j \leq 2$.

Note that $\mathcal{P}^{\oplus i}$ and $\mathcal{C}^{\oplus i}$ are random sets built from values read from tapes $(p_1, p_1^{-1}, \ldots, p_5, p_5^{-1})$ and $\mathsf{ic}/\mathsf{ic}^{-1}$ respectively, while $\mathcal{H}^{\oplus j}$ is fixed and determined by the history $\mathcal{H}$.

We have used a coarse definition for the bad events, which is easy to remember and convenient to use. It is possible to refine the definition and further reduce the probability of bad events.

Definition 15 (Good Executions)**.** An execution of G_2 is said to be *good* if neither BadPerm nor BadIC occurs in the execution.

5.3 The Simulator Does not Abort in Good Executions

Our goal in this section is to prove that during a good execution of G_2, the simulator never aborts. This is a two-step process: we first show that this holds under a natural assumption on query cycles (namely, that they are *safe*, see definition below); then we show that all query cycles are indeed safe.

Definition 16 (safe query cycle)**.** A query cycle is said to be *safe* if for any 2chain C triggered during the query cycle, both endpoints of C were dummy or table-undefined[8] at the beginning of the query cycle.

Informally, the assumption that a query cycle is safe is more or less equivalent to the assumption that at the beginning of the query cycle, no incomplete path of length 3 exists (but we do not need to formalize this further).

As just explained, our first step will be to prove that the simulator does not abort during a safe query cycle. The simulator can only abort in procedure Assign which is only called during the completion phase. Moreover, this completion phase can be split into two sub-phases: first, the simulator calls ReadTape(i, δ, z) for each pending query (i, δ, z), and then it calls AdaptPath(C) for each triggered

[7] Since $\mathcal{H}$, $\mathcal{P}$, and $\mathcal{C}$ are multisets, two distinct elements may be equal.

[8] Recall that when we say that an endpoint is table-undefined, this implicitly means it is non-dummy.

2chain C. We will consider each sub-phase in turn, showing that for a safe query cycle, the simulator aborts in neither of them.

Consider a query cycle during which a 2chain C is triggered. By Lemma 5 (c), C must be table-defined at the beginning of the query cycle, hence, by definition of the history $\mathcal{H}$, any endpoint of C which was non-dummy at the beginning of the query cycle is in $\mathcal{H}$. The following lemma clarifies the situation in case a triggered 2chain has a dummy endpoint at the beginning of the query cycle.

Lemma 8. *In any execution of* G_2*, if a 2chain triggered during a query cycle had a dummy endpoint at the beginning of the query cycle, then this endpoint is non-dummy when the completion phase starts and moreover it is in* $\mathcal{C}$*, the set of values read on tapes* ic *or* ic^{-1} *during the query cycle.*

Proof. Recall that only the right, resp. left endpoint of a $(4,5)$-, resp. $(1,2)$-2chain can be dummy. We consider the case of the right endpoint of a triggered $(4,5)$-2chain, the other case follows by symmetry. A table-defined $(4,5)$-2chain $C = (4, 5, y_4, x_5, k)$ can be triggered either during a call to FindNewPaths$(3, -, \cdot)$ or to FindNewPaths$(1, +, \cdot)$ in a $(4,5)$-query cycle. We first consider the case where it is triggered during a call to FindNewPaths$(3, -, \cdot)$. Inspection of the pseudocode then shows that right after C has been triggered, a call to $\mathrm{Dec}(k, y_5)$ resulting from a call to $\mathrm{Next}(5, y_5, k)$ at line 74 will make C's right endpoint non-dummy, and moreover $r(C) = \mathsf{ic}^{-1}(k, y_5) \in \mathcal{C}$. Next, we consider the case where a $(4,5)$-2chain C was triggered during a call to FindNewPaths$(1, +, \cdot)$. Note that during a call to FindNewPaths$(1, +, x_1)$, the simulator triggers a table-defined 2chain $C = (4, 5, y_4, x_5, k)$ only if $\mathrm{Check}(k, x_1, y_5)$, where $y_5 = P_5(x_5)$, is true, which can never be if $r(C) = \perp$. This implies that C had a non-dummy right endpoint at the beginning of the query cycle. This is because by Lemma 6, we know that in a $(4,5)$-query cycle the entries in tables T/T^{-1} are modified only during the detection phase by calls to $\mathrm{Dec}(\cdot, \cdot)$ resulting from calls to $\mathrm{Next}(5, \cdot, \cdot)$ at line 74. By inspection of the pseudocode, this call occurs right after a $(4,5)$-2chain C' has been triggered. If this call changed the right endpoint of C from dummy to non-dummy, we have $C = C'$ by Lemma 7 since C and C' share a key and a non-dummy endpoint and C' will not be triggered again in the query cycle by the check at line 48 in the pseudocode. □

We are now ready to prove that for a safe query cycle, the simulator does not abort during the calls to ReadTape.

Lemma 9. *Consider a safe query cycle in a good execution of* G_2*. Then the simulator does not abort during the calls to* ReadTape *occurring during the query cycle.*

Proof. Let τ_0 denote the beginning of the query cycle. Assume towards a contradiction that the simulator aborts in a call to ReadTape during a safe $(i, i+1)$-query cycle. By Lemma 5 (b), ReadTape can only be called for permutation queries of type $(i-1, -)$ or $(i+2, +)$. Assume that the simulator aborts in a call to ReadTape$(i+2, +, x_{i+2})$ (the case of a call to ReadTape$(i-1, -, y_{i-1})$ is similar). This means that we have either $x_{i+2} \in P_{i+2}$ or $p_{i+2}(x_{i+2}) \in P_{i+2}^{-1}$ (where

p_{i+2} is the random permutation tape) when the call occurs. Assume first that $x_{i+2} \in P_{i+2}$ when the call to ReadTape occurs. We first show that x_{i+2} is table-undefined (i.e., $x_{i+2} \notin P_{i+2}$) just before the completion phase starts. Procedure ReadTape is only called on pending queries, hence, by Lemma 5 (d), x_{i+2} is either the initiating query, or the endpoint of some triggered 2chain. If this is the initiating query, then it was table-undefined at τ_0 (otherwise the simulator would have returned immediately), and since permutation tables are not modified by the detection phase, it is still table-undefined when the completion phase starts. If this is the endpoint of a triggered 2chain C, then, by the assumption that the query cycle is safe, this endpoint was either dummy or table-undefined at τ_0. If it was table-undefined at τ_0, then x_{i+2} is still table-undefined when the completion phase starts since permutation tables are not modified by the detection phase. Otherwise, if it was dummy at τ_0, then by Lemma 8, $x_{i+2} = r(C)$ is non-dummy when the completion phase starts and is in $\mathcal{C}$. If x_{i+2} is table-defined when the completion phase starts, then it was already table-defined at τ_0, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens, contradicting the assumption that the execution is good. In all cases, we see that x_{i+2} is table-undefined just before the completion phase starts. Hence, if $x_{i+2} \in P_{i+2}$ when the call to ReadTape occurs, this can only be due to another call to ReadTape$(i+2, +, x_{i+2})$ in the same query cycle. Yet this is impossible since any permutation query is added at most once to Pending in a given query cycle due to the checks at lines 57 and 75. Assume now that $p_{i+2}(x_{i+2}) \in P_{i+2}^{-1}$ when the call to ReadTape occurs. If $p_{i+2}(x_{i+2}) \in P_{i+2}^{-1}$ at the beginning of the query cycle, then $p_{i+2}(x_{i+2}) \in \mathcal{P} \cap \mathcal{H}$ and BadPerm occurs. Otherwise, this can only happen due to another call to ReadTape$(i+2, +, x'_{i+2})$ in the same query cycle, where $x'_{i+2} \neq x_{i+2}$ since any permutation query is added at most once to Pending in a given query cycle. But again this is impossible since p_{i+2} encodes a permutation, so that $x'_{i+2} \neq x_{i+2}$ implies $p_{i+2}(x'_{i+2}) \neq p_{i+2}(x_{i+2})$. Hence, the simulator does not abort in a call to ReadTape. □

Now that we proved that during a safe query cycle, the simulator does not abort during calls to ReadTape, we know that it will try to "adapt" each triggered 2chain C by calling AdaptPath(C). The lemma below shows that the values used in the "adaptation" call to Assign when completing a 2chain are random in some precise sense.

Lemma 10. *Consider a safe $(i, i+1)$-query cycle in a good execution of G_2. Let $C = (i, i+1, y_i, x_{i+1}, k)$ be a triggered 2chain, and assume that* AdaptPath(C) *is called during the completion phase.*[9] *Consider the resulting call to* Assign$(i+3, x_{i+3}, y_{i+3})$[10] *at line 84. Then*

- *if $i \neq 3$, $x_{i+3} = p_{i+2}(r(C)) \oplus k$ where $p_{i+2}(r(C)) \in \mathcal{P}$ and $k \in \mathcal{H}$;*
- *if $i = 3$, $x_{i+3} = x_1 = \mathsf{ic}^{-1}(k, p_5(r(C))) \in \mathcal{C}$;*
- *if $i \neq 2$, $y_{i+3} = p_{i-1}^{-1}(\ell(C)) \oplus k$ where $p_{i-1}^{-1}(\ell(C)) \in \mathcal{P}$ and $k \in \mathcal{H}$;*

[9] The only reason why this call might not occur is because the simulator aborts before the call, which we cannot assume does not happen at this point of the proof.

[10] We denote the third argument y_{i+3} rather than y_{i-2} for clarity.

– *if* $i = 2$, $y_{i+3} = y_5 = \mathsf{ic}(k, p_1^{-1}(\ell(C))) \in \mathcal{C}$.

Proof. We only prove the result for x_{i+3}, the result for y_{i+3} follows by symmetry. For the case $i \neq 3$, the expression of x_{i+3} follows directly from the fact that the simulator does not abort during the calls to ReadTape (Lemma 9) and inspection of the pseudocode. Note that k is the key of C which is table-defined at the beginning of the query cycle (since it is triggered), hence by definition $k \in \mathcal{H}$. Consider now the case $i = 3$, i.e., the completion of a $(3,4)$-2chain $C = (3, 4, y_3, x_4, k)$ during a $(3,4)$-query cycle. By Lemma 9, the simulator does not abort during the call to Assign$(5, x_5, y_5)$ resulting from the call to ReadTape$(5, +, x_5)$, where $x_5 = P_4(x_4) \oplus k$ and $y_5 = p_5(x_5) \in \mathcal{P}$. Hence, when the call to AdaptPath(C) occurs, by inspection of the pseudocode, a call to Next$(5, y_5, k)$ occurs at line 81, resulting in a call to Dec(k, y_5). We argue that the cipher query $(-, k, y_5)$ is table-undefined when this call occurs. If it is table-defined at the beginning of the query cycle, then by definition $y_5 \in \mathcal{H}$, so that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and BadPerm occurs, contradicting the assumption that the execution is good. If y_5 is table-undefined when the query cycle begins, but table-defined when the call to AdaptPath(C) occurs, then, since by Lemma 6, tables T/T^{-1} are not modified during the detection phase in a $(3,4)$-query cycle, this can only be due to a call to AdaptPath(C') for another triggered 2chain C' which caused a call to Dec(k, y_5). But this would mean that C and C' have the same key and $r(C) = r(C') = y_5 \neq \perp$, which by Lemma 7 implies $C = C'$, which is impossible since a 2chain is triggered at most once in a query cycle because of the checks at lines 48 and 66. Hence, $(-, k, y_5)$ is table-undefined when the call to Dec(k, y_5) occurs and hence $x_1 = \mathsf{ic}^{-1}(k, y_5) \in \mathcal{C}$. □

We are now ready to prove that the simulator does not abort during the calls to AdaptPath in a safe query cycle.

Lemma 11. *Consider a safe query cycle in a good execution of* G_2*. Then the simulator does not abort during the calls to* AdaptPath *occurring during the query cycle.*

Proof. Assume towards a contradiction that the simulator aborts in a call to AdaptPath during a safe $(i, i+1)$-query cycle. Let $C = (i, i+1, y_i, x_{i+1}, k)$ be the corresponding 2chain. Consider the resulting call to Assign$(i+3, x_{i+3}, y_{i+3})$. The simulator aborts only if $x_{i+3} \in P_{i+3}$ or if $y_{i+3} \in P_{i+3}^{-1}$. Let us consider the case where $x_{i+3} \in P_{i+3}$ when Assign$(i+3, x_{i+3}, y_{i+3})$ is called (the case where $y_{i+3} \in P_{i+3}^{-1}$ is similar). We distinguish the case $i \neq 3$ and $i = 3$.

Consider first the case $i \neq 3$. Then, by Lemma 10, we have $x_{i+3} = p_{i+2}(x_{i+2}) \oplus k$, where $x_{i+2} = r(C)$, $p_{i+2}(x_{i+2}) \in \mathcal{P}$, and $k \in \mathcal{H}$. If $x_{i+3} \in P_{i+3}$ at the beginning of the query cycle, then by definition $x_{i+3} \in \mathcal{H}$, so that $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$, which means BadPerm occurs. If $x_{i+3} \notin P_{i+3}$ at the beginning of the query cycle, $x_{i+3} \in P_{i+3}$ before the call to Assign$(i+3, x_{i+3}, y_{i+3})$ only due to another call to AdaptPath(C') for a distinct 2chain $C' = (i, i+1, y'_i, x'_{i+1}, k')$ and a resulting call to Assign$(i+3, x'_{i+3}, y'_{i+3})$ with $x'_{i+3} = x_{i+3}$. By Lemma 10, we

have $x'_{i+3} = p_{i+2}(x'_{i+2}) \oplus k'$, where $x'_{i+2} = r(C')$. Hence, $x'_{i+3} = x_{i+3}$ implies

$$p_{i+2}(x_{i+2}) \oplus p_{i+2}(x'_{i+2}) = k \oplus k'. \tag{5}$$

Observe that we cannot have $x_{i+2} = x'_{i+2}$ (i.e., $r(C) = r(C')$), as otherwise

$$k = p_{i+2}(x_{i+2}) \oplus x_3 = p_{i+2}(x'_{i+2}) \oplus x'_3 = k'$$

which by Lemma 7 implies $C = C'$; but this is impossible since a 2chain is triggered at most once in a query cycle because of the checks at lines 48 and 66. Hence, $x_{i+2} \neq x'_{i+2}$ and Eq. (5) implies that $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$, i.e., BadPerm occurs.

Consider now the case $i = 3$, i.e., we are in a (3,4)-query cycle, the 2chain for which AdaptPath is called is $C = (3, 4, y_3, x_4, k)$ and the resulting assignment is Assign$(1, x_1, y_1)$. Then, by Lemma 10, we have $x_1 = \mathsf{ic}^{-1}(k, y_5) \in \mathcal{C}$, where $y_5 = p_5(r(C))$. If $x_1 \in P_1$ at the beginning of the query cycle, then by definition $x_1 \in \mathcal{H}$, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$, which means BadIC occurs. If $x_1 \notin P_1$ at the beginning of the query cycle, $x_1 \in P_1$ before the call to Assign$(1, x_1, y_1)$ only due to another call to AdaptPath(C') for a distinct 2chain $C' = (3, 4, y'_3, x'_4, k')$ and a resulting call to Assign$(1, x'_1, y'_1)$ with $x'_1 = x_1$. By Lemma 10, we have $x'_1 = \mathsf{ic}^{-1}(k', y'_5) \in \mathcal{C}$, where $y'_5 = p_5(r(C'))$. Hence, $x'_1 = x_1$ implies

$$\mathsf{ic}^{-1}(k, y_5) = \mathsf{ic}^{-1}(k', y'_5). \tag{6}$$

Observe that we cannot have $(k, y_5) = (k', y'_5)$ since this would imply $r(C) = p_5^{-1}(y_5) = p_5^{-1}(y'_5) = r(C')$, which by Lemma 7 would imply $C = C'$; but this is impossible since a 2chain is triggered at most once in a query cycle. Hence, $(k, y_5) \neq (k', y'_5)$ and Eq. (6) implies that $\mathcal{C}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$, i.e., BadIC occurs.

This concludes the proof. □

Lemma 12. *In a good execution of* G_2*, the simulator does not abort during a safe query cycle.*

Proof. This follows directly from Lemmas 9 and 11 since the simulator can only abort during calls to ReadTape and AdaptPath. □

It remains now to show that all query cycles in a good execution are safe. The key observation for this is that the simulator ensures that, at the end of the completion phase, any table-defined 2chain with a table-defined endpoint (one can think of it as a "3chain") necessarily belongs to a complete path as per Definition 8. We show that this property is preserved by any distinguisher's cipher query (a direct consequence of Lemma 13) and by any query cycle (Lemmas 14, 15 and 16), and deduce that this holds at the beginning of any query cycle (Lemma 18). We also show that a 2chain which is complete at the beginning of a query cycle cannot be triggered (Lemma 20). From this we are able to deduce that any query cycle is safe.

The lemma below says that a cipher query made by the distinguisher cannot switch the state of a 2chain from table-undefined to table-defined, nor switch an endpoint from dummy to table-defined.

Lemma 13. *Consider a cipher query made by the distinguisher in a good execution of* G_2 *that modifies tables* T/T^{-1}*. Let* C *be a 2chain. Then the following two properties hold:*

(a) If C *is table-undefined before the query, then* C *is still table-undefined after the query has been answered.*
(b) If C *is table-defined and one of its endpoints is dummy before the query and non-dummy after the query has been answered, then this endpoint is table-undefined after the query has been answered.*

Proof. We first prove (a). The result is obvious for an $(i, i+1)$-2chain for $i \in \{1, 2, 3, 4\}$ since whether such a 2chain is table-defined or not is independent from tables T/T^{-1}. So we only need to consider $(5, 1)$-2chains. Assume towards a contradiction that this is false, i.e., there exists a $(5, 1)$-2chain $C = (5, 1, y_5, x_1, k)$ which is table-undefined just before the query and table-defined just after the query has been answered. Since tables P_1/P_1^{-1} and P_5/P_5^{-1} are not modified by the query, this necessarily means that $(1, +, x_1)$ and $(5, -, y_5)$ are table-defined before the query, $T(k, x_1) = \perp$ before the query, and $T(k, x_1) = y_5$ after the query. Thus, the distinguisher's cipher query was necessarily an encryption query (k, x_1) or a decryption query (k, y_5), and in both cases we see that the random value read from ic was in $\mathcal{H}$, which means $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens, a contradiction with the assumption that the execution is good.

We then prove (b). We prove it in the case of the right endpoint of a $(4, 5)$-2chain (the case of the left endpoint of a $(1, 2)$-2chain follows by symmetry). Let $C = (4, 5, y_4, x_5, k)$ be a table-defined $(4, 5)$-2chain, let $y_5 = P_5(x_5)$, and assume that C's right endpoint is dummy before the cipher query and table-defined after the query has been answered. Then $T^{-1}(k, y_5) = \perp$ before the query, and after the query $T^{-1}(k, y_5) = x_1$ where $(1, +, x_1)$ is table-defined. Thus, the distinguisher's cipher query was necessarily an encryption query (k, x_1) or a decryption query (k, y_5), and in both cases we see that the random value read from ic was in $\mathcal{H}$, which means $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens, a contradiction with the assumption that the execution is good. □

Lemmas 14, 15, and 16 below say, informally, that if a new "3chain" (i.e., a table-defined 2chain with at least one table-defined endpoint) is created during a query cycle, then it is necessarily complete at the end of the query cycle. Unfortunately, the only way to prove this important result seems to be through a delicate case analysis. Since there are many ways to "create a 3chain", we phrase it in three distinct lemmas depending on the state of the 2chain at the beginning of the query cycle. In the three lemmas, we say that "a complete path has been triggered" if any of the five 2chains belonging to the complete path has been triggered. Note also that the three lemmas assume a safe query cycle, which by Lemma 12 implies that the simulator does not abort, which by Lemma 5 (e) implies that any triggered 2chain is complete at the end of the query cycle.

Lemma 14. *Consider a safe query cycle in a good execution of* G_2*. Let* C *be a 2chain which is table-defined at the beginning of the query cycle and such that*

one of its endpoints is dummy at the beginning of the query cycle and non-dummy at the end of the query cycle. Then, at the end of the query cycle, C belongs to a complete path which was triggered during the query cycle.

Proof. We consider the case of the right endpoint of a $(4,5)$-2chain. The case of the left endpoint of a $(1,2)$-2chain follows by symmetry. Let τ_0 denote the beginning of the query cycle and τ_1 denote its end. Let $C = (4, 5, y_4, x_5, k)$ be a $(4,5)$-2chain such that $r(C) = \perp$ at τ_0 and $r(C) = x_1 \neq \perp$ at τ_1. Let $y_5 = P_5(x_5)$. This means that $T^{-1}(k, y_5) = \perp$ at τ_0 and $T^{-1}(k, y_5) = x_1$ at τ_1. We consider the five possibilities for the type of query cycle.

- *Case of a $(1,2)$- or $(2,3)$-query cycle.* By Lemma 6, the simulator only calls Enc during such query cycles. This means that $y_5 = \mathsf{ic}(k, x_1)$ must have been read during the query cycle, and since $y_5 \in \mathcal{H}$, $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC occurs, contradicting the assumption that the execution is good.
- *Case of a $(3,4)$-query cycle.* By Lemma 6, the simulator only calls Dec during the completion phase in this case. Moreover, it is easy to check from the pseudocode that the call $\mathrm{Dec}(k, y_5)$ can only occur during the call to $\mathrm{AdaptPath}(D)$, where $D = (3, 4, y_3, x_4, k)$ with $x_4 = P_4^{-1}(y_4)$ and $y_3 = x_4 \oplus k$. By Lemma 5 (e), D is necessarily complete at τ_1, and it is easy to check that C belongs to the same complete path as D at τ_1.
- *Case of a $(4,5)$-query cycle.* By Lemma 6, the simulator only calls Dec during the detection phase in this case. Moreover, it is easy to check from the pseudocode that the call to $\mathrm{Dec}(k, y_5)$ can only occur just after C has been triggered. Hence, by Lemma 5 (e), C is necessarily complete at τ_1.
- *Case of a $(5,1)$-query cycle.* By Lemma 6, tables T/T^{-1} are not modified during a $(5,1)$-query cycle. Hence, $r(C)$ cannot become non-dummy during the query cycle. □

Lemma 15. *Consider a safe query cycle in a good execution of G_2. Let C be a 2chain which is table-defined at the beginning of the query cycle and such that one of the endpoints of C is table-undefined at the beginning of the query cycle and table-defined at the end of the query cycle. Then, at the end of the query cycle, C belongs to a complete path which was triggered during the query cycle.*

Proof. Assume that the query cycle we consider is an $(i, i+1)$-query cycle. Let τ_0 denote the beginning of the query cycle and τ_1 denote its end. We consider each possible type for the 2chain C.

Case of an $(i, i+1)$-2chain $C = (i, i+1, y_i, x_{i+1}, k)$. We consider the case where this is the right endpoint of C which goes from non-dummy and table-undefined to table-defined during the query cycle (the case of the left endpoint follows by symmetry). Since $r(C)$ is non-dummy and table-undefined at τ_0, necessarily $(i+2, +, r(C))$ became pending during the query cycle (otherwise it would still be table-undefined at τ_1). Since C is table-defined at τ_0, C was necessarily triggered (either during the call to $\mathrm{FindNewPaths}(i+2, +, r(C))$, or during the call the $\mathrm{FindNewPaths}(i-1, -, \ell(C))$ which then made $r(C)$ pending). Hence, by Lemma 5 (e), C is complete at τ_1.

Case of an $(i+1, i+2)$-2chain $C = (i+1, i+2, y_{i+1}, x_{i+2}, k)$. First, note that since P_i/P_i^{-1} are not modified during the query cycle, the left endpoint of C cannot go from non-dummy and table-undefined to table-defined during the query cycle, hence this is necessarily the right endpoint of C which does. Since $r(C)$ is non-dummy at τ_0, then by definition $x_{i+3} \coloneqq r(C)$ is in $\mathcal{H}$. The only way x_{i+3} can become table-defined during the query cycle is because of a call to $\text{Assign}(i+3, x'_{i+3}, y'_{i+3})$ with $x'_{i+3} = x_{i+3}$ resulting from a call to $\text{AdaptPath}(C')$, where $C' = (i, i+1, y'_i, x'_{i+1}, k')$ has been triggered during the query cycle. By Lemma 10, if $i \neq 3$ we have $x'_{i+3} = p_{i+2}(r(C')) \oplus k'$ where $k' \in \mathcal{H}$ and $p_{i+2}(r(C'))$ is read during the query cycle, so that $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ happens, whereas if $i = 3$ then $x'_{i+3} = x'_1 = \mathsf{ic}^{-1}(k', p_5(r(C')))$ is read during the query cycle, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens. In all cases this contradicts the assumption that the execution is good.

Case of an $(i+2, i+3)$-2chain $C = (i+2, i+3, y_{i+2}, x_{i+3}, k)$. First, note that since P_{i+1}/P_{i+1}^{-1} are not modified during the query cycle, the left endpoint of C cannot go from non-dummy and table-undefined to table-defined during the query cycle, hence this is necessarily the right endpoint of C which does. Note that since $x_{i+4} = x_{i-1} \coloneqq r(C)$ is non-dummy at τ_0, by definition $x_{i-1} \in \mathcal{H}$. The only way x_{i-1} can become table-defined during the $(i, i+1)$-query cycle is because of a call to $\text{Assign}(i-1, p_{i-1}^{-1}(y_{i-1}), y_{i-1})$ with $x_{i-1} = p_{i-1}^{-1}(y_{i-1})$ resulting from a call to $\text{ReadTape}(i-1, -, y_{i-1})$. This implies that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and hence $\mathsf{BadPerm}$ occurs, contradicting the assumption that the execution is good.

Other cases. The case of an $(i-2, i-1)$-, resp. of an $(i-1, i)$-2chain, can be deduced by symmetry from the case of an $(i+2, i+3)$-, resp. $(i+1, i+2)$-2chain. □

Lemma 16. *Consider a safe query cycle in a good execution of G_2. Let C be a 2chain such that*

(i) at the beginning of the query cycle, C is table-undefined;
(ii) at the end of the query cycle, C is table-defined and at least one of its two endpoints is table-defined.

Then, at the end of the query cycle, C belongs to a complete path which was triggered during the query cycle.

Proof. Assume that the query cycle we consider is an $(i, i+1)$-query cycle. Let τ_0 denote the beginning of the query cycle and τ_1 denote its end. We consider each possible type for the 2chain C.

Case of an $(i, i+1)$-2chain $C = (i, i+1, y_i, x_{i+1}, k)$. Since P_i/P_i^{-1} and P_{i+1}/P_{i+1}^{-1} are not modified during an $(i, i+1)$-query cycle, and moreover tables T/T^{-1} are not modified during a $(5, 1)$-query cycle by Lemma 6, C cannot be table-undefined before the query cycle and table-defined after, hence this case is impossible.

Case of an $(i+1,i+2)$-2chain $C = (i+1, i+2, y_{i+1}, x_{i+2}, k)$. First, note that since tables P_{i+1}/P_{i+1}^{-1} are not modified during the query cycle, y_{i+1} must necessarily be table-defined at τ_0 (so that in particular $y_{i+1} \in \mathcal{H}$) for C to be table-defined at τ_1.

We start by showing that x_{i+2} was necessarily table-undefined at τ_0. This is clear for $i \neq 4$ (i.e., when C is an inner 2chain) since otherwise C would be table-defined already at τ_0. If $i = 4$, i.e., we are considering a $(4,5)$-query cycle and a $(5,1)$-2chain $C = (5, 1, y_5, x_1, k)$, and if x_1 is already table-defined at τ_0, then C could become table-defined because of an assignment to T/T^{-1} due to a simulator call to $\mathrm{Next}(k, y_5)$ at line 74. Yet this would mean that $x_1 = \mathsf{ic}^{-1}(k, y_5)$, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC occurs, contradicting the assumption that the execution is good. In all cases, we see that x_{i+2} was necessarily table-undefined at τ_0.

We now distinguish two cases depending on which endpoint of C is table-defined at τ_1. Assume first that this is the left endpoint, and let y_i be the value of $\ell(C)$ at τ_1. Since tables P_i/P_i^{-1} are not modified during the query cycle, y_i was already table-defined at τ_0. Let $x_{i+1} = P_{i+1}^{-1}(y_{i+1})$ and $D = (i, i+1, y_i, x_{i+1}, k)$. Then D was table-defined at τ_0, its right endpoint was either dummy or equal to x_{i+2} and hence table-undefined at τ_0, and at τ_1 its right endpoint is table-defined since C is table-defined. Hence, by Lemmas 14 and 15, D belongs to a complete path which was triggered during the query cycle, and it is easy to check that C belongs to the same complete path as D at τ_1.

Assume now that this is the right endpoint of C which is table-defined at τ_1 and let x_{i+3} denote the value of $r(C)$ at τ_1. Since x_{i+2} was table-undefined at τ_0 and we are considering an $(i, i+1)$-query cycle, x_{i+2} necessarily became pending during the query cycle and a call to $\mathrm{Assign}(i+2, x_{i+2}, p_{i+2}(x_{i+2}))$ occurred. By Lemma 5 (d), x_{i+2} is either the initiating query, in which case it is in $\mathcal{H}$ by definition, or the endpoint of some triggered $(i, i+1)$-2chain D, in which case it is in $\mathcal{H}$ by definition if $r(D)$ is non-dummy at τ_0, or in $\mathcal{C}$ if $r(D)$ is dummy at τ_0 by Lemma 8, which might only happen when $i = 4$. We now distinguish three sub-cases depending on i:

- Case $i \in \{1, 2, 5\}$. Then C is neither a $(4,5)$- nor a $(5,1)$-2chain and its right endpoint is given by $x_{i+3} = p_{i+2}(x_{i+2}) \oplus k = p_{i+2}(x_{i+2}) \oplus y_{i+1} \oplus x_{i+2}$. Assume first that x_{i+3} was table-defined already at τ_0, so that $x_{i+3} \in \mathcal{H}$. Then $p_{i+2}(x_{i+2}) = x_{i+3} \oplus k = x_{i+3} \oplus y_{i+1} \oplus x_{i+2} \in \mathcal{H}^{\oplus 3}$ (recall $x_{i+2} \in \mathcal{H}$ if $i \neq 4$), so $\mathcal{P} \cap \mathcal{H}^{\oplus 3} \neq \emptyset$ and $\mathsf{BadPerm}$ happens, contradicting the assumption that the execution is good. Assume now that x_{i+3} was table-undefined at τ_0. Then it can only become table-defined during the query cycle because of a call to $\mathrm{Assign}(i+3, x'_{i+3}, y'_{i+3})$ with $x'_{i+3} = x_{i+3}$ resulting from a call to $\mathrm{AdaptPath}(C')$, where $C' = (i, i+1, y'_i, x'_{i+1}, k')$ has been triggered during the query cycle. By Lemma 10, we have $x'_{i+3} = p_{i+2}(x'_{i+2}) \oplus k'$ where $x'_{i+2} = r(C')$, $p_{i+2}(x'_{i+2}) \in \mathcal{P}$ and $k' \in \mathcal{H}$. Hence, $x'_{i+3} = x_{i+3}$ implies that

$$p_{i+2}(x_{i+2}) \oplus p_{i+2}(x'_{i+2}) = y_{i+1} \oplus x_{i+2} \oplus k'.$$

Hence, if $x_{i+2} \neq x'_{i+2}$, then $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 3} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs, whereas if $x_{i+2} = x'_{i+2}$ then

$$k = p_{i+2}(x_{i+2}) \oplus x_3 = p_{i+2}(x'_{i+2}) \oplus x'_3 = k',$$

and one can check that C belongs to the same completed path as C' at τ_1.

- Case $i = 3$. Then we are considering a $(3,4)$-query cycle, $C = (4, 5, y_4, x_5, k)$ is a $(4,5)$-2chain, and its right endpoint at τ_1 is given by $x_{i+3} = x_1 = T^{-1}(k, p_5(x_5))$. If $(-, k, p_5(x_5))$ was already table-defined at τ_0, then $p_5(x_5) \in \mathcal{H}$, so that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs. Hence, $(-, k, p_5(x_5))$ became table-defined during the query cycle, which implies that at τ_1, the 2chain $(5, 1, p_5(x_5), x_1, k)$ belongs to a complete path that was triggered during the query cycle. It is easy to see that C belongs to the same complete path at τ_1.
- Case $i = 4$. Then we are considering a $(4,5)$-query cycle, $C = (5, 1, y_5, x_1, k)$ is a $(5,1)$-2chain, and its right endpoint at τ_1 is given by $x_{i+3} = x_2 = p_1(x_1) \oplus k$. If $(-, k, y_5)$ was table-undefined at τ_0, then it became table-defined during the query cycle, so that C belongs to a complete path which was triggered during the query cycle. Assume now that $T^{-1}(k, y_5) = x_1$ already at τ_0, so that $k \in \mathcal{H}$. First, if x_2 was table-defined already at τ_0, then since $p_1(x_1) = x_2 \oplus k$ one has $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ happens. If x_2 was table-undefined at τ_0, then it can only become table-defined during the query cycle because of a call to $\mathrm{Assign}(2, x'_2, y'_2)$ with $x'_2 = x_2$ resulting from a call to $\mathrm{AdaptPath}(C')$, where $C' = (4, 5, y'_4, x'_5, k')$ has been triggered during the query cycle. By Lemma 10, we have $x'_2 = p_1(x'_1) \oplus k'$ where $x'_1 = r(C')$, $p_1(x'_1) \in \mathcal{P}$, and $k' \in \mathcal{H}$. Hence, $x'_2 = x_2$ implies that $p_1(x_1) \oplus p_1(x'_1) = k \oplus k'$. Hence, if $x_1 \neq x'_1$, then $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs, whereas if $x_1 = x'_1$, then

$$k = p_1(x_1) \oplus x_2 = p_1(x'_1) \oplus x'_2 = k',$$

and one can check that C belongs to the same completed path as C' at τ_1.

Case of an $(i+2, i+3)$-2chain $C = (i+2, i+3, y_{i+2}, x_{i+3}, k)$. Assume first that the left endpoint of C is table-defined at τ_1 and denote y_{i+1} the value of $\ell(C)$ at τ_1. Since tables P_{i+1}/P_{i+1}^{-1} are not modified during an $(i, i+1)$-query cycle, y_{i+1} is already table-defined at τ_0. Let $D = (i+1, i+2, y_{i+1}, x_{i+2}, k)$, where x_{i+2} is the value of $P_{i+2}^{-1}(y_{i+2})$ at τ_1. Then either D is table-undefined at τ_0 and table-defined with a table-defined right endpoint at τ_1, in which case we can apply the conclusion of the analysis for the case of a $(i+1, i+2)$-2chain, or D is table-defined at τ_0 and its right endpoint becomes table-defined during the query cycle, in which case we can apply Lemmas 14 and 15. In all cases we can conclude that D belongs to a complete path that was triggered during the query cycle, and C belongs to the same complete path.

Assume now that the right endpoint of C is table-defined at τ_1 and denote $x_{i-1} = x_{i+4}$ the value of $r(C)$ at τ_1. Since C is table-defined at τ_1, let $y_{i+3} = P_{i+3}(x_{i+3})$.

- Case $i \in \{1, 4, 5\}$. Then C is neither a $(4,5)$- nor a $(5,1)$-2chain, hence its right endpoint is given by $x_{i-1} = y_{i+3} \oplus k$ with $k = y_{i+2} \oplus x_{i+3}$, hence

$$y_{i+2} \oplus x_{i+3} \oplus y_{i+3} \oplus x_{i-1} = 0. \tag{7}$$

We will distinguish all possible sub-cases depending on whether y_{i+2}, x_{i+3}, y_{i+3}, and x_{i-1} are table-defined at τ_0 (in which case these values are in $\mathcal{H}$) or not. Note that at least one of the two queries y_{i+2} and x_{i+3} is table-undefined at τ_0 since C is table-undefined at τ_0 (and is not a $(5,1)$-2chain). Moreover, since we are considering an $(i, i+1)$-query cycle, then

- if y_{i+2} was table-undefined at τ_0, then it became table-defined because of a call to ReadTape$(2, +, x_{i+2})$ and hence $y_{i+2} = p_{i+2}(x_{i+2}) \in \mathcal{P}$
- if x_{i+3} (and hence y_{i+3}) was table-undefined at τ_0, then it became table-defined because of a call to Assign$(i+3, x_{i+3}, y_{i+3})$ resulting from a call to AdaptPath(C'), where $C' = (i, i+1, y'_i, x'_{i+1}, k')$ has been triggered during the query cycle. By Lemma 10, we have $x_{i+3} = p_{i+2}(x'_{i+2}) \oplus k'$ where $x'_{i+2} = r(C')$, $p_{i+2}(x'_{i+2}) \in \mathcal{P}$, and $k' \in \mathcal{H}$, and $y_{i+3} = p_{i-1}^{-1}(y'_{i-1}) \oplus k'$ where $y'_{i-1} = \ell(C')$ and $p_{i-1}^{-1}(y'_{i-1}) \in \mathcal{P}$.
- if x_{i-1} was table-undefined at τ_0, then it became table-defined because of a call to ReadTape$(i-1, -, y''_{i-1})$ and $x_{i-1} = p_{i-1}^{-1}(y''_{i-1}) \in \mathcal{P}$.

Finally, note that for the case where x_{i+3} was table-undefined at τ_0, we can assume that $x_{i+2} \neq x'_{i+2}$ since otherwise $k = k'$ and C belongs to the same completed path as C' at τ_1. Hence, we see that when substituting all possibilities in (7) with at least y_{i+2} or x_{i+3} table-undefined at τ_0 (and hence involving an element of $\mathcal{P}$), we always end up with an equation implying that $\mathcal{P}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$ for some $1 \leq i \leq 4$ and some $0 \leq j \leq 4$. (For example, if we assume y_{i+2}, x_{i+3}, y_{i+3}, and x_{i-1} were all table-undefined at τ_0, then Eq. (7) yields

$$p_{i+2}(x_{i+2}) \oplus p_{i+2}(x'_{i+2}) \oplus p_{i-1}^{-1}(y'_{i-1}) \oplus p_{i-1}^{-1}(y''_{i-1}) = 0$$

and hence $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ depending on whether $y'_{i-1} = y''_{i-1}$.)

– Case $i = 2$. Then we are considering a $(2,3)$-query cycle, $C = (4, 5, y_4, x_5, k)$ is a $(4,5)$-2chain, and its right endpoint at τ_1 is given by $x_{i+4} = x_1 = T^{-1}(k, y_5)$ where $y_5 = P_5(x_5)$. If the cipher query $(-, k, y_5)$ was table-undefined at τ_0, then it became table-defined during the query cycle, which implies that at τ_1, the 2chain $(5, 1, y_5, x_1, k)$ belongs to a complete path which was triggered during the query cycle, and C belongs to the same complete path. Assume now that the cipher query (k, x_1, y_5) was table-defined at τ_0, so that $k, x_1, y_5 \in \mathcal{H}$. If x_1 was table-undefined at τ_0, then it became table-defined because of a call ReadTape$(1, -, y_1)$ and hence $x_1 = p_1^{-1}(y_1) \in \mathcal{P}$, so that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and BadPerm happens. Since C is table-undefined at τ_0, either y_4 or x_5 is table-undefined at τ_0. Assume first that x_5 was table-undefined at τ_0. Then it could only become table-defined because of a call to Assign$(5, x_5, y_5)$ resulting from a call to AdaptPath(C') where C' was triggered during the query cycle. By Lemma 10, we have $y_5 \in \mathcal{C}$, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens. Finally, assume that x_5 was table-defined but y_4 was table-undefined at τ_0 (hence $x_5 \in \mathcal{H}$). Then y_4 became table-defined because of a call ReadTape$(4, +, x_4)$ and hence $y_4 = p_4(x_4) \in \mathcal{P}$. Moreover, $y_4 = x_5 \oplus k$ where $k \in \mathcal{H}$, so that $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and BadPerm happens.

– Case $i = 3$. Then we are considering a $(3,4)$-query cycle, $C = (5, 1, y_5, x_1, k)$ is a $(5,1)$-2chain, and its right endpoint at τ_1 is given by $x_{i+4} = x_2 = y_1 \oplus k$ where $y_1 = P_1(x_1)$. If the cipher query (k, x_1, y_5) was table-undefined at τ_0, then it became table-defined during the query cycle, which implies that at τ_1, C belongs to a complete path which was triggered during the query cycle. Assume now that the cipher query (k, x_1, y_5) was table-defined at τ_0, so that $k, x_1, y_5 \in \mathcal{H}$. Assume that y_5 was table-undefined at τ_0. Then y_5 became table-defined because of a call ReadTape$(5, +, x_5)$, so that $y_5 = p_5(x_5) \in \mathcal{P}$. Hence, $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and BadPerm occurs. Assume now that x_1 was table-undefined at τ_0. Then it could only become table-defined because of a call to Assign$(1, x_1, y_1)$ resulting from a call to AdaptPath(C') where C' was triggered during the query cycle. By Lemma 10, we have $x_1 \in \mathcal{C}$, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens. Finally, assume that y_1 was table-defined and x_2 was table-undefined at τ_0 (so that $y_1 \in \mathcal{H}$). Then x_2 became table-defined because of a call ReadTape$(2, -, y_2)$ and hence $x_2 = p_2^{-1}(y_2) \in \mathcal{P}$. Since $x_2 = y_1 \oplus k$ where $k \in \mathcal{H}$, one has $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and hence BadPerm occurs.

Other cases. The case of an $(i-2, i-1)$-, resp. of an $(i-1, i)$-2chain, can be deduced by symmetry from the case of an $(i{+}2, i{+}3)$-, resp. $(i{+}1, i{+}2)$-2chain. □

We collect Lemmas 14, 15, and 16 under a simpler to grasp form in the following lemma. To state it more concisely, we introduce the following definition.

Definition 17. We say a 2chain C *induces a 3chain* if C is table-defined and at least one of its endpoints is table-defined.

Lemma 17. *Consider a safe query cycle in a good execution of* G_2*. Let* C *be a 2chain which does not induce a 3chain at the beginning of the query cycle, but induces a 3chain at the end of the query cycle. Then, at the end of the query cycle,* C *belongs to a complete path which was triggered during the query cycle.*

Proof. Note that the opposite of "C induces a 3chain" is "C is table-defined and its two endpoints are dummy or table-undefined, or C is table-undefined". Hence, Lemmas 14, 15, and 16 cover all possible cases for a 2chain to not induce a 3chain at the beginning of a query cycle and induce a 3chain at the end of the query cycle. □

Lemma 18. *Consider a point* τ_0 *in a good execution of* G_2 *which is either the beginning of a query cycle, or the end of the execution, and assume that all query cycles until* τ_0 *were safe. Let* C *be a 2chain. Assume that at* τ_0*,* C *is table-defined and at least one of the two endpoints of* C *is table-defined. Then* C *is complete at* τ_0*.*

Proof. Let C be a 2chain which induces a 3chain at time τ_0. Recall that the opposite of "C induces a 3chain" is "C is table-defined and its two endpoints are dummy or table-undefined, or C is table-undefined". Consider the last assignment to a table before C induces a 3chain. This cannot have been an assignment to T/T^{-1} due to a cipher query made by the distinguisher; indeed, by Lemma 13,

such an assignment cannot switch the state of a 2chain from table-undefined to table-defined, nor switch the endpoint of a table-defined 2chain from dummy to table-defined (and besides it is clear that it cannot switch an endpoint from non-dummy and table-undefined to table-defined). Hence this can only have happened during a query cycle that occurred before τ_0 (which was necessarily safe by the assumption that all query cycles before τ_0 were safe). But according to Lemma 17, at the end of this previous query cycle, C was complete, and this still holds at τ_0, hence the result. □

Lemma 19. *Consider an $(i, i+1)$-query cycle with initiating query (i_0, δ_0, z_0) in a good execution of G_2. Then, if an $(i, i+1)$-2chain C is triggered during the query cycle, there exists a sequence of $(i, i+1)$-2chains $(C_0, \ldots, C_t)$ triggered in this order such that $C_t = C$, and,* at the beginning of the query cycle*, either $z_0 = r(C_0) \neq \bot$, $\ell(C_1) = \ell(C_0) \neq \bot$, $r(C_2) = r(C_1) \neq \bot$, etc. if $(i_0, \delta_0) = (i+2, +)$, or $\ell(C_0) = z_0$, $r(C_1) = r(C_0) \neq \bot$, $\ell(C_2) = \ell(C_1) \neq \bot$, etc. if $(i_0, \delta_0) = (i-1, -)$.*

Proof. The existence of sequence $(C_0, \ldots, C_t)$ follows easily from inspection of the pseudocode, the only non-trivial point to prove is that all endpoints are already non-dummy at the beginning of the query cycle. But this follows easily from the fact that a $(4, 5)$-2chain with a right dummy endpoint cannot be triggered by a call to FindNewPaths$(1, +, \cdot)$, and that if a $(4, 5)$-2chain C with a right dummy endpoint is triggered by a call to FindNewPaths$(3, -, \cdot)$, then by Lemma 8 its right endpoint $r(C)$ is in $\mathcal{C}$ after getting non-dummy, and hence the call to FindNewPaths$(1, +, r(C))$ cannot trigger any 2chain unless $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens (and from the symmetric observations for a $(1, 2)$-2chain with a dummy left endpoint). □

Lemma 20. *Consider an $(i, i+1)$-query cycle in a good execution of G_2, and assume that all previous query cycles were safe. Then, if an $(i, i+1)$-2chain is complete at the beginning of the query cycle, it cannot be triggered during this query cycle.*

Proof. Let τ_0 denote the beginning of the query cycle. Assume towards a contradiction that there exists an $(i, i+1)$-2chain C which is complete at τ_0 and is triggered during the query cycle. Denote (i_0, δ_0, z_0) the initiating query of the query cycle. By Lemma 19, there exists a sequence $(C_0, \ldots, C_t)$ of triggered $(i, i+1)$-2chains such that $C_t = C$, and, at τ_0, either $r(C_0) = z_0$, $\ell(C_1) = \ell(C_0) \neq \bot$, $r(C_2) = r(C_1) \neq \bot$, etc., or $\ell(C_0) = z_0$, $r(C_1) = r(C_0) \neq \bot$, $\ell(C_2) = \ell(C_1) \neq \bot$, etc.

By definition of a complete 2chain, $r(C)$ and $\ell(C)$ are non-dummy and table-defined at τ_0. Hence, since C and C_{t-1} have a common endpoint, C_{t-1} is table-defined and has one of its endpoints non-dummy and table-defined at τ_0, which by Lemma 18 and the assumption that all previous query cycles were safe implies that C_{t-1} is complete at τ_0. By recursion, it follows that C_0 is complete at τ_0, which implies that the initiating query was table-defined at the beginning of the query cycle, a contradiction. □

Lemma 21. *Any query cycle in a good execution of G_2 is safe.*

Proof. Assume towards a contradiction that this is false, and consider the first query cycle for which this does not hold (hence, all previous query cycles were safe). This means that during this query cycle, some 2chain C was triggered such that C had a table-defined endpoint at the beginning of the query cycle. Since by Lemma 5 (c), any triggered 2chain is table-defined before the query cycle begins, this implies that when the query cycle started, C was table-defined and one of its endpoints was table-defined. But according to Lemma 18, since all previous query cycles were safe, this implies that C was complete at the beginning of the query cycle, which by Lemma 20 (and again the fact that all previous query cycles were safe) implies that C cannot be triggered during the query cycle, a contradiction. □

The main result of this section now follows easily.

Lemma 22. *The simulator does not abort in a good execution of* G_2.

Proof. This is a direct consequence of Lemmas 12 and 21. □

5.4 Efficiency of the Simulator

Now that we have established that the simulator does not abort in good executions of G_2, we prove that it also runs in polynomial time. For this, we first prove some additional properties of good executions of G_2 in Section 5.4.1. The core of the termination argument is in Section 5.4.2.

5.4.1 2chain-Cycles

In this section, we establish some additional properties of good executions in G_2 which would be used to prove the efficiency of the simulator in Section 5.4. From Lemma 21, we know that any query cycle in a good execution is safe i.e. for any 2chain C triggered during the query cycle, the endpoints of the 2chain were either table-undefined or dummy at the beginning of the query cycle. Moreover, by Lemma 22, we know that the simulator does not abort in a good execution. Throughout this section, when we assume a good execution of G_2, we will use these properties (without repeatedly referring to Lemmas 21 or 22).

We start by introducing the notions of 2cycle and 4cycle.

Definition 18 (2cycle)**.** We say that 2 table-defined 2chains

$$C_j = (i, i+1, y_i^{(j)}, x_{i+1}^{(j)}, k^{(j)}), j \in \{1, 2\},$$

form a *2cycle at* $(i, i+1)$ if they satisfy the following conditions:

1. $C_1 \neq C_2$;
2. both endpoints of 2chains C_1 and C_2 are non-dummy and table-undefined;
3. $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_1)$.

Definition 19 (4cycle)**.** We say that 4 table-defined 2chains

$$C_j = (i, i+1, y_i^{(j)}, x_{i+1}^{(j)}, k^{(j)}), j \in \{1,2,3,4\},$$

form a *4cycle at* $(i, i+1)$ if they satisfy the following conditions:

1. $C_1 \neq C_2$, $C_2 \neq C_3$, $C_3 \neq C_4$, $C_4 \neq C_1$;
2. both endpoints of all 2chains are non-dummy and table-undefined;
3. $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_3)$, $\ell(C_3) = \ell(C_4)$, $r(C_4) = r(C_1)$.

In the next few lemmas, we will prove that at the end of a query cycle in a good execution of G_2, there does not exist a 2cycle or a 4cycle at $(4,5)$ or at $(1,2)$. In order to do so, we first prove the following lemma.

Lemma 23. *In a good execution of* G_2*, if* (k, x_1, y_5) *was table-defined through a cipher query made by the simulator in a query cycle, then the 2chain* $(5, 1, y_5, x_1, k)$ *is table-defined at the end of the query cycle.*

Proof. Let us consider the various query cycles in which the cipher query (k, x_1, y_5) could get table-defined.

(1, 2)-query cycle. By Lemma 6, a query (k, x_1, y_5) can get table-defined only through a call $\mathrm{Prev}(1, x_1, k)$ at line 56. Then, by inspection of the pseudocode $(5, -, y_5)$ is a pending query (where $\mathrm{Prev}(1, x_1, k)$ returned y_5 by assumption) and $(1, x_1, y_1)$ is table-defined. Since the simulator does not abort in a good execution of G_2 by Lemma 22, the pending query is table-defined at the end of the query cycle. Hence, the 2chain $(5, 1, y_5, x_1, k)$ is table-defined at the end of the query cycle.

(2, 3)-query cycle. By Lemma 6, a query (k, x_1, y_5) can get table-defined only through a call $\mathrm{Prev}(1, x_1, k)$ at line 83. Since the simulator does not abort in a good execution of G_2 by Lemma 22, the call to $\mathrm{Assign}(5, \cdot, y_5)$ leads to $(5, \cdot, y_5)$ being table-defined. One can also verify from the pseudocode that $\mathrm{Prev}(1, x_1, k)$ is called at line 83 in a (2, 3)-query cycle only if $(1, x_1, y_1)$ is table-defined through $\mathrm{ReadTape}(1, -, y_1)$. Hence, the 2chain $(5, 1, y_5, x_1, k)$ is table-defined at the end of the query cycle.

The cases of a (3, 4) and (4, 5)-query cycles are analogous to those of (2, 3) and (1, 2)-query cycles respectively. In a (5, 1)-query cycle, by Lemma 6, no new cipher queries get table-defined. This completes the proof of the lemma. □

Lemma 24. *In a good execution of* G_2*, a 4cycle never appears at* $(4,5)$ *or at* $(1,2)$ *due to a cipher query by the distinguisher.*

Proof. Assume towards a contradiction that in a good execution of G_2 table-defined 2chains (C_1, C_2, C_3, C_4) form a 4cycle where $C_j = (4, 5, y_4^{(j)}, x_5^{(j)}, k^{(j)})$ due to a cipher query made by the distinguisher i.e. the 4cycle did not appear before the distinguisher's cipher query.

We know that in a cipher query (δ, k, z) by the distinguisher, the only query that can get table-defined is the cipher query itself. If (δ, k, z) was already table-defined, then no new queries are table-defined and the 4cycle cannot be formed. If (δ, k, z) was not table-defined, for the 4cycle to appear, the cipher query should cause $(-, k^{(j)}, y_5^{(j)})$ to be table-defined for some $j \in \{1, \ldots, 4\}$. Without loss of generality assume the new cipher query defined $T^{-1}(k^{(1)}, y_5^{(1)})$ to be $x_1^{(1)}$ which leads to the 4cycle. Then, by definition of a 4cycle, $r(C_1) = r(C_4)$ implying $x_1^{(1)} = x_1^{(4)} = T^{-1}(k^{(4)}, y_5^{(4)})$.

Let $(\delta, k, z) = (-, k^{(1)}, y_5^{(1)})$. Then, $x_1^{(1)} \in \mathcal{C}$. Since the cipher query was the only one to be defined, we have $x_1^{(4)} \in \mathcal{H}$. Hence, we have $x_1^{(1)} \oplus x_1^{(4)} = 0$ where $x_1^{(1)} \in \mathcal{C}$ and $x_1^{(4)} \in \mathcal{H}$. On the other hand if $(\delta, k, z) = (+, k^{(1)}, x_1^{(1)})$. Then, $y_5^{(1)} \in \mathcal{C}$. Since C_1 is table-defined, we also have $y_5^{(1)} \in \mathcal{H}$. Hence, we have $x_1^{(1)} \oplus x_1^{(4)} = 0$ where $y_5^{(1)} \in \mathcal{C}$ and $y_5^{(1)} \in \mathcal{H}$. Thus BadIC occurs regardless of the direction of the cipher query as we have $\mathcal{H}^{\oplus 1} \cap \mathcal{C}^{\oplus 1} \neq \emptyset$.

By a similar case analysis, we can show that in a good execution of G_2 a 4cycle does not appear at $(1, 2)$ due to a cipher query made by the distinguisher. □

Lemma 25. *If a 4cycle does not exist at* $(4, 5)$ *or at* $(1, 2)$ *at the beginning of a query cycle in a good execution of* G_2*, then a 4cycle does not exist at* $(4, 5)$ *or at* $(1, 2)$ *at the end of that query cycle.*

Proof. Assume towards a contradiction that at the end of a query cycle in a good execution of G_2 table-defined 2chains (C_1, C_2, C_3, C_4) form a 4cycle where $C_j = (4, 5, y_4^{(j)}, x_5^{(j)}, k^{(j)})$. Let $r(C_j) = T^{-1}(k^{(j)}, y_5^{(j)}) = x_1^{(j)} \neq \perp$ since the endpoints of the 2chains are not dummy by Definition 19. By the same definition, the endpoint queries of the 2chains are table-undefined.

For the analysis, we consider the last assignment that was made to the tables immediately after which the 4cycle appeared and distinguish between two cases: (i) the last assignment was a cipher query getting table-defined and (ii) the last assignment was due to a permutation query getting table-defined.

Let us consider case (i) first. Here, we assume that the 4cycle formed immediately after a cipher query $(k^{(j)}, x_1^{(j)}, y_5^{(j)})$ that was table-defined in the query cycle for some $j \in \{1, \ldots, 4\}$. (Note that in a query cycle, cipher queries are made only by the simulator.) By Lemma 23, this means that the 2chain $(5, 1, x_1^{(j)}, y_5^{(j)}, k^{(j)})$ is table-defined. In particular, this implies that the right endpoint of C_j $r(C_j) = x_1^{(j)}$ is table-defined which contradicts our assumption.

Next we consider case (ii) i.e. the 4cycle formed immediately after a permutation query $(4, x_4^{(j)}, y_4^{(j)})$ or $(5, x_5^{(j)}, y_5^{(j)})$ was table-defined. Then, for $j = 1, 2, 3, 4$ the cipher queries $T(k^{(j)}, x_1^{(j)}) = y_5^{(j)}$ are table-defined at the beginning of the query cycle; this is because if they are table-defined during the query cycle, by Lemma 23, the right endpoint $r(C_j)$ gets table-defined contradicting our assumption that a 4cycle is formed in the query cycle. Hence, we have $k^{(j)} \in \mathcal{H}$, $x_1^{(j)} \in \mathcal{H}$, $y_5^{(j)} \in \mathcal{H}$ for $j \in \{1, \ldots, 4\}$.

$(1,2)$-*query cycle.* Let the query cycle be a $(1,2)$-query cycle. From Table 1, we can see that in a $(1,2)$-query cycle, calls to ReadTape$(5,-,\cdot)$ occur and adaptation occurs at position 4.

Let us consider the case where all permutation queries $(4, x_4^{(j)}, y_4^{(j)})$ were table-defined at the beginning of the query cycle. Then, for the 4cycle to form at least one permutation query $(5, x_5^{(j)}, y_5^{(j)})$ should get table-defined. In a $(1,2)$-query cycle, this occurs in a call to ReadTape$(5,-,y_5^{(j)})$ which implies that $x_5^{(j)} \in \mathcal{P}$. Now, for the 2chain C_j, we have $x_5^{(j)} = y_4^{(j)} \oplus k^{(j)}$. As observed above, $k^{(j)} \in \mathcal{H}$ and by assumption that all permutation queries at position 4 for the 2chains C_j are table-defined at the beginning of the query cycle, we also have $y_4^{(j)} \in \mathcal{H}$. This implies that BadPerm occurs since $x_5^{(j)} \in \mathcal{P}$ and $x_5^{(j)} = y_4^{(j)} \oplus k^{(j)} \in \mathcal{H}^{\oplus 2}$. So, $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$.

Next, we consider the case that in a $(1,2)$-query cycle at least one permutation query $(4, x_4^{(j)}, y_4^{(j)})$ was table-defined. In a $(1,2)$-query cycle, this occurs in a call to Assign$(4, x_4^{(j)}, y_4^{(j)})$ inside AdaptPath$(1, 2, y_1', x_2', k')$ for some triggered 2chain $C' = (1, 2, y_1', x_2', k')$ in that query cycle. We differentiate two sub-cases here: $\ell(C') = y_5^{(j)}$ or not.

If $\ell(C') = y_5^{(j)}$, then at the end of the query cycle C_j will be such that its right endpoint $r(C_j) = x_1'$ which is table-defined. Hence, this will not form a 4cycle. If $\ell(C') \neq y_5^{(j)}$, let $\ell(C') = y_5'$. Then, Assign$(5, x_5', y_5')$ occurs in a call to ReadTape$(5,-,y_5')$ leading to $x_5' \in \mathcal{P}$. Now, we have

$$x_5' \oplus k' = y_4^{(j)} = x_5^{(j)} \oplus k^{(j)}.$$

In the equation above, $k', k^{(j)} \in \mathcal{H}$, $x_5' \in \mathcal{P}$. Also, $x_5^{(j)} \in \mathcal{P} \cup \mathcal{H}$ since $(5, x_5^{(j)}, y_5^{(j)})$ could either be table-defined as a left endpoint of a 2chain $\tilde{C} \neq C'$ in the current query cycle or could have been table-defined at the beginning of the query cycle itself. Then, either $\mathcal{H}^{\oplus 3} \cap \mathcal{P} \neq \emptyset$ or $\mathcal{H}^{\oplus 2} \cap \mathcal{P}^{\oplus 2} \neq \emptyset$ and BadPerm occurs.

$(2,3)$-*query cycle.* Let the query cycle be a $(2,3)$-query cycle. From Table 1, we can see that in a $(2,3)$-query cycle, calls to ReadTape$(4,+,\cdot)$ occur and adaptation occurs at position 5.

Again, let us consider the case where all permutation queries $(4, x_4^{(j)}, y_4^{(j)})$ were table-defined at the beginning of the query cycle. Then, for the 4cycle to form at least one permutation query $(5, x_5^{(j)}, y_5^{(j)})$ should get table-defined. In a $(2,3)$-query cycle, this occurs in a call to Assign$(5, x_5^{(j)}, y_5^{(j)})$ called during AdaptPath(C') for some triggered chain C' in the query cycle. Since all query cycles in a good execution are safe by Lemma 21, using Lemma 10, we have $y_5^{(j)} \in \mathcal{C}$. However, since all cipher queries $(k^{(j)}, x_1^{(j)}, y_5^{(j)})$ are table-defined at the beginning of the query cycle, we have $y_5^{(j)} \in \mathcal{H}$. Hence, $\mathcal{H} \cap \mathcal{C} \neq \emptyset$ and BadIC occurs. So for a 4cycle to form in a $(2,3)$-query cycle, all permutation queries $(5, x_5^{(j)}, y_5^{(j)})$ need to be table-defined at the beginning of the query cycle.

Now, let us consider the case where at least one permutation query $(4, x_4^{(j)}, y_4^{(j)})$ is table-defined during the query cycle. In a $(2,3)$-query cycle, this occurs in a call to ReadTape$(4, +, x_4^{(j)})$ which implies that $y_4^{(j)} \in \mathcal{P}$. As mentioned above, we have $x_5^{(j)}, y_5^{(j)} \in \mathcal{H}$ since for a 4cycle to form in a $(2,3)$-query cycle, all permutation queries $(5, x_5^{(j)}, y_5^{(j)})$ need to be table-defined at the beginning of the query cycle. We also know that $k^{(j)} \in \mathcal{H}$ since all the cipher queries in the 4cycle are table-defined. Hence, $y_4^{(j)} = x_5^{(j)} \oplus k^{(j)} \in \mathcal{H}^{\oplus 2}$ implying that $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and hence BadPerm occurs.

$(3,4)$-*query cycle.* Let the query cycle be a $(3,4)$-query cycle. From Table 1, we can see that in a $(3,4)$-query cycle, calls to ReadTape$(5, +, \cdot)$ occur and no new permutation query of the form $(4, \cdot, \cdot)$ gets table-defined. So for a 4cycle to form in a $(3,4)$-query cycle, all permutation queries $(4, x_4^{(j)}, y_4^{(j)})$ need to be table-defined at the beginning of the query cycle. Hence, $x_4^{(j)}, y_4^{(j)} \in \mathcal{H}$.

If a permutation query $(5, x_5^{(j)}, y_5^{(j)})$ gets table-defined in a $(3,4)$-query cycle, then $y_5^{(j)} \in \mathcal{P}$. However, since all cipher queries $(k^{(j)}, x_1^{(j)}, y_5^{(j)})$ are table-defined at the beginning of the query cycle, we have $y_5^{(j)} \in \mathcal{H}$. Hence, $\mathcal{H} \cap \mathcal{P} \neq \emptyset$ and BadPerm occurs.

$(4,5)$-*query cycle.* In a $(4,5)$-query cycle, no new permutation queries of the form $(4, \cdot, \cdot)$ and $(5, \cdot, \cdot)$ get table-defined. As shown before, in a good execution of G_2, all cipher queries of the 4cycle need to be table-defined at the beginning of the query cycle. Hence, a 4cycle at $(4,5)$ cannot form in a $(4,5)$-query cycle.

$(5,1)$-*query cycle.* Let the query cycle be a $(5,1)$-query cycle. From Table 1, we can see that in a $(5,1)$-query cycle, calls to ReadTape$(4, -, \cdot)$ occur and no new permutation query of the form $(5, \cdot, \cdot)$ gets table-defined.

So for a 4cycle to form in a $(5,1)$-query cycle, at least one permutation query $(4, x_4^{(j)}, y_4^{(j)})$ needs to be table-defined during the query cycle. Without loss of generality, let $j = 1$. Then, $x_4^{(1)} \in \mathcal{P}$. We differentiate two cases for the analysis: $(4, x_4^{(2)}, y_4^{(2)})$ is table-defined at the beginning of the query cycle or not.

Let us first consider the case where $(4, x_4^{(2)}, y_4^{(2)})$ is table-defined at the beginning of the query cycle. By definition of a 4cycle, $\ell(C_1) = \ell(C_2)$. Hence, $x_4^{(1)} \oplus k^{(1)} = x_4^{(2)} \oplus k^{(2)}$. Here, $k^{(1)}, k^{(2)}, x_4^{(2)} \in \mathcal{H}$ and $x_4^{(1)} \in \mathcal{P}$. Hence, $\mathcal{P} \cap \mathcal{H}^{\oplus 3} \neq \emptyset$ and BadPerm occurs.

Next, we consider the case where $(4, x_4^{(2)}, y_4^{(2)})$ is table-defined during the query cycle. Then, $x_4^{(2)} \in \mathcal{P}$ since we are considering a $(5,1)$-query cycle. By definition of a 4cycle, $\ell(C_1) = \ell(C_2)$. This also means that $x_4^{(1)} \neq x_4^{(2)}$ as otherwise we have $x_5^{(1)} = x_5^{(2)}$ and $C_1 = C_2$ which violates the definition of a 4cycle (Definition 19). Then, $\ell(C_1) = \ell(C_2)$ implies that

$$x_4^{(1)} \oplus k^{(1)} = x_4^{(2)} \oplus k^{(2)}$$

where $x_4^{(1)}, x_4^{(2)} \in \mathcal{P}$ and $k^{(1)}, k^{(2)} \in \mathcal{H}$ such that $x_4^{(1)} \neq x_4^{(2)}$. Hence, $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs.

By a similar case analysis, we can show that in a good execution of G_2 a 4cycle does not appear at $(1,2)$ at the end of a query cycle.

□

Lemma 26. *At the end of a query cycle in a good execution of G_2, a 4cycle does not exist at $(4,5)$ or at $(1,2)$.*

Proof. This follows immediately from Lemmas 24 and 25. □

Corollary 1. *At the end of a query cycle in a good execution of G_2, there does not exist a 2cycle at $(4,5)$ or at $(1,2)$.*

Proof. Assume towards a contradiction that at the end of a query cycle in a good execution of G_2 there exists a 2cycle at (say) $(4,5)$ consisting of table-defined 2chains $C = (4, 5, y_4, x_5, k)$ and $C' = (4, 5, y_4, x_5, k)$. Then, by definition of a 2cycle C and C' are such that $C \neq C'$, both endpoints of C and C' are non-dummy and table-undefined and $\ell(C) = \ell(C')$ and $r(C') = r(C)$.

Then there exists a 4cycle at $(4,5)$ consisting of chains C_j for $j = 1, \ldots, 4$, where $C_j = C$ if $j = 1, 3$ and $C_j = C'$ if $j = 2, 4$. This is because we have $C_j \neq C_{j+1}$ (where $C_{j+1} = C_1$ when $j = 4$) for all j since $C \neq C'$. We know that both endpoints of C and C' are non-dummy and table-undefined. Finally, $\ell(C_j) = \ell(C_{j+1})$ if $j = 1, 3$ since $\ell(C) = \ell(C')$ and $r(C_j) = r(C_{j+1})$ if $j = 2, 4$ since $r(C') = r(C)$. This contradicts Lemma 26 which says that no 4cycle appears at $(4,5)$ in a good execution of G_2. □

Lemma 27. *In a good execution of G_2, for $i \in \{2,4\}$, there do not exist two distinct table-defined queries (i, x_i, y_i) and (i, x_i', y_i') such that $x_i \oplus y_i = x_i' \oplus y_i'$.*

Proof. Assume towards a contradiction that there exist distinct table-defined queries (i, x_i, y_i) and (i, x_i', y_i') where $i \in \{2,4\}$ such that $x_i \oplus y_i = x_i' \oplus y_i'$ in a good execution of G_2. We first analyze the case where $i = 2$. Here we distinguish between two cases: (i) the queries were table-defined in different query cycles and (ii) the queries were table-defined in the same query cycle.

For case (i), let us assume without loss of generality that $(2, x_2', y_2')$ was table-defined in an earlier query cycle. We consider the query cycle in which $(2, x_2, y_2)$ was table-defined. Then, $x_2', y_2' \in \mathcal{H}$. In the query cycle, the permutation query $(2, x_2, y_2)$ can get table-defined only in one of the following ways: (a) in a call to $\mathrm{ReadTape}(2, +, x_2)$ or $\mathrm{ReadTape}(2, -, y_2)$ (b) in a call to $\mathrm{Assign}(2, x_2, y_2)$ that occurs in $\mathrm{AdaptPath}(C)$ for a 2chain C triggered in the query cycle.

If $(2, x_2, y_2)$ was table-defined in a call to $\mathrm{ReadTape}(2, +, x_2)$, then we have $y_2 \in \mathcal{P}$ and $x_2 \in \mathcal{H}$. The former is due to the fact that in a good execution of G_2 all query cycles are safe by Lemma 21. The latter is because by inspection of the pseudocode, we can see that $\mathrm{ReadTape}(2, +, x_2)$ is called only when $(2, +, x_2)$ is a pending query and by Lemma 5(d), $(2, +, x_2)$ is pending only if it were the initiating query or the endpoint of a triggered 2chain. Note also that an endpoint at position 2 cannot be dummy by definition. Hence, by definition,

history $\mathcal{H}$ contains all non-dummy endpoints and the initiating query. Thus, $x_2 \in \mathcal{H}$. By a similar analysis, we can show that when $(2, x_2, y_2)$ is table-defined in a call to ReadTape$(2, -, y_2)$, we have $x_2 \in \mathcal{P}$ and $y_2 \in \mathcal{H}$. Thus, if $(2, x_2, y_2)$ was table-defined in a call to ReadTape$(2, \cdot, \cdot)$ and $x_2 \oplus y_2 = x_2' \oplus y_2'$, we have $\mathcal{H}^{\oplus 3} \cap \mathcal{P} \neq \emptyset$. This implies that $\mathsf{BadPerm}$ occurs which contradicts our assumption that this is a good execution of G_2.

Next, we consider the sub-case where $(2, x_2, y_2)$ is table-defined in a call to Assign$(2, x_2, y_2)$ that occurs in AdaptPath(C) for a 2chain $C = (i, i{+}1, y_i, x_{i+1}, k)$ triggered in the query cycle. From Table 1, we know that 2 is an adapt position only for a $(4, 5)$-query cycle. Hence, from Lemma 10, we can deduce that $x_2 \oplus y_2 \in \mathcal{P}^{\oplus 2}$. This is because $x_2 \oplus y_2 = (p_1(r(C)) \oplus k) \oplus (p_3^{-1}(\ell(C)) \oplus k) \in \mathcal{P}^{\oplus 2}$. Then, if $x_2 \oplus y_2 = x_2' \oplus y_2'$, we have $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$. This implies that $\mathsf{BadPerm}$ occurs which contradicts our assumption that this is a good execution of G_2. This completes the analysis of case (i) where we assumed that the two distinct queries were table-defined in different query cycles.

In case (ii), the queries $(2, x_2, y_2)$ and $(2, x_2', y_2')$ are both table-defined in the same query cycle. As observed before, a permutation query gets table-defined either in a call to ReadPath$(2, \cdot, \cdot)$ or in a call to AdaptPath(C) for a triggered 2chain C. Since both queries are at position 2 and are table-defined in the same query cycle, they are both table-defined through the same procedures i.e. either both through calls to ReadPath or both through calls to AdaptPath. Hence, we consider the following two sub-cases: (a) $(2, x_2, y_2)$ and $(2, x_2', y_2')$ are table-defined through calls to ReadPath$(2, +, x_2)$ and ReadPath$(2, +, x_2')$ respectively or through calls to ReadPath$(2, -, y_2)$ and ReadPath$(2, -, y_2')$ respectively. (b) $(2, x_2, y_2)$ and $(2, x_2', y_2')$ are table-defined through calls to Assign$(2, x_2, y_2)$ and Assign$(2, x_2', y_2')$ that occurred in calls to AdaptPath(C) and AdaptPath(C') respectively.

We analyse sub-case (a) first. As argued earlier, if $(2, x_2, y_2)$ and $(2, x_2', y_2')$ were table-defined through calls to ReadTape$(2, +, x_2)$ and ReadTape$(2, +, x_2')$ respectively, then $x_2, x_2' \in \mathcal{H}$ and $y_2, y_2' \in \mathcal{P}$. Note that $y_2 \neq y_2'$ as the two queries are distinct and the permutation tables always encode a partial permutation. So $x_2 \oplus y_2 = x_2' \oplus y_2'$ implies $\mathcal{H}^{\oplus 2} \cap \mathcal{P}^{\oplus 2} \neq \emptyset$. Hence $\mathsf{BadPerm}$ occurs which contradicts our assumption that this is a good execution of G_2. The analogous analysis works for the case where calls to ReadTape was of the form ReadTape$(2, -, y_2)$ and ReadTape$(2, -, y_2')$.

Next, we analyse the sub-case where $(2, x_2, y_2)$ and $(2, x_2', y_2')$ were adapted in the same query cycle. Say they were adapted in AdaptPath(C) and AdaptPath(C') respectively where $C = (4, 5, y_4, x_5, k)$ and $C' = (4, 5, y_4', x_5', k')$ and $C \neq C'$ since the two queries are distinct. Again, by consulting Table 1, we can see that 2 is an adapt position only in a $(4, 5)$-query cycle. By Lemma 10, $x_2 = p_1(r(C)) \oplus k$ and $y_2 = p_3^{-1}(\ell(C)) \oplus k$ and similarly $x_2' = p_1(r(C')) \oplus k'$ and $y_2' = p_3^{-1}(\ell(C')) \oplus k'$. Since we are in a safe query cycle by Lemma 21, $p_1(r(C))$, $p_1(r(C'))$, $p_3^{-1}(\ell(C))$, $p_3^{-1}(\ell(C')) \in \mathcal{P}$. So, if $x_2 \oplus y_2 = x_2' \oplus y_2'$, we have

$$p_1(r(C)) \oplus p_3^{-1}(\ell(C)) = p_1(r(C')) \oplus p_3^{-1}(\ell(C')). \tag{8}$$

We consider the following sub-cases here: (a) $r(C) \neq r(C')$, (b) $\ell(C) \neq \ell(C')$ and (c) $r(C) = r(C')$ and $\ell(C) = \ell(C')$. Consider case (a) first, if $r(C) \neq r(C')$, then in Equation (8), we have either $0 \in \mathcal{P}^{\oplus 4}$ or $0 \in \mathcal{P}^{\oplus 2}$ (where the latter occurs if $\ell(C) = \ell(C')$). Then, either $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ and in either case, BadPerm occurs. Considering case (b) next, if $\ell(C) \neq \ell(C')$, by a similar argument as just presented in case (a), we have either $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ and in either case, BadPerm occurs.

Finally, considering case (c), if $r(C) = r(C')$ and $\ell(C) = \ell(C')$, we distinguish between two cases here: (1) the cipher queries $T^{-1}(k, x_5)$ and $T^{-1}(k', x'_5)$ are table-defined at the beginning of the query cycle and (2) at least one of the cipher queries is not table-defined at the beginning of the query cycle. Considering case (1), if both cipher queries are table-defined at the beginning of the query cycle, then a 2cycle appears at (4, 5). This is because $C \neq C'$ by assumption that the two adapted queries $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ are distinct, the endpoints of the 2chains are non-dummy by assumption that the cipher queries are table-defined at the beginning of the query cycle, the endpoints are table-undefined since query cycles in a good execution of G_2 are safe by Lemma 21 and finally, $r(C) = r(C')$ and $\ell(C) = \ell(C')$ by assumption. This contradicts Corollary 1 which says that a 2cycle does not appear at (4, 5) in a good execution of G_2.

Next, we consider the case where $r(C) = r(C')$ and $\ell(C) = \ell(C')$ and at least one of the cipher queries is not table-defined at the beginning of the query cycle. If exactly one of them is not table-defined at the beginning of the query cycle, let us assume without loss of generality that $T^{-1}(k, x_5)$ is not table-defined at the beginning of the query cycle and $T^{-1}(k', x'_5)$ is. Also, in a $(4, 5)$-query cycle cipher queries are only of the form $(-, \cdot, \cdot)$ i.e. only inverse cipher queries are made. Then if $r(C) = r(C')$, we have $r(C) \in \mathcal{C}$ and $r(C') \in \mathcal{H}$. Hence BadIC occurs as $\mathcal{C}^{\oplus 1} \cap \mathcal{H}^{\oplus 1} \neq \emptyset$. On the other hand, if both cipher queries are not table-defined at the beginning of the query cycle and if $r(C) = r(C')$, we have $T^{-1}(k, x_5) = T^{-1}(k', x'_5)$ where $r(C) = T^{-1}(k, x_5) \in \mathcal{C}$ and $r(C') = T^{-1}(k', x'_5) \in \mathcal{C}$. These are distinct cipher queries since $C \neq C'$. Hence, BadIC occurs as $0 \in \mathcal{C}^{\oplus 2}$ implying that $\mathcal{C}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$.

Concluding, in a good execution of G_2, there do not exist two distinct queries (i, x_i, y_i) and (i, x'_i, y'_i) such that $x_i \oplus y_i = x'_i \oplus y'_i$ when $i = 2$. A similar case analysis works for the case when $i = 4$.

□

Lemma 28. *In a good execution of G_2, for $i \in \{2, 4\}$ there never exist four distinct table-defined queries $(i, x_i^{(j)}, y_i^{(j)})$ with $j = 1, 2, 3, 4$ such that*

$$\sum_{j=1}^{4} (x_i^{(j)} \oplus y_i^{(j)}) = 0. \tag{9}$$

Proof. Assume towards a contradiction that in a good execution of G_2, there exist four distinct queries $(i, x_i^{(j)}, y_i^{(j)})$ such that Eq. (9) holds. We analyse the case when $i = 2$. The proof for the case where $i = 4$ proceeds similarly. Let us consider the last assignment to the tables before the equation held. The assignment should

have table-defined $(2, x_2^{(j)}, y_2^{(j)})$ for some $j \in \{1, \ldots, 4\}$. Without loss of generality, let us assume that $(2, x_2^{(1)}, y_2^{(1)})$ (i.e. $j = 1$) was the last query to be table-defined. Let us consider the various query cycles in which the query could have been table-defined.

(1, 2)-query cycle and (2, 3)-query cycle. Note that no new queries of the form $(2, \cdot, \cdot)$ get table-defined in such query cycles. So $(2, x_2^{(1)}, y_2^{(1)})$ was not table-defined in a $(1, 2)$ or a $(2, 3)$-query cycle.

(3, 4)-query cycle. By referring to Table 1, we can see that in a $(3, 4)$-query cycle, a query at position 2 gets table-defined only through calls to ReadTape$(2, -, \cdot)$. For queries $(2, x_2^{(j)}, y_2^{(j)})$ where $j = 2, \ldots, 4$, either $(2, x_2^{(j)}, y_2^{(j)})$ was already table-defined at the beginning of the query cycle or it gets table-defined in the same query cycle (but before $(2, x_2^{(1)}, y_2^{(1)})$ by assumption). Hence, we have $y_2^{(j)} \in \mathcal{H}$ for all $j = 1, 2, 3, 4$. This is because either $(2, x_2^{(j)}, y_2^{(j)})$ is table-defined at the beginning of the query cycle or $(2, -, y_2^{(j)})$ is a pending query and hence gets table-defined in the current query cycle. By definition of history $\mathcal{H}$, the initiating query, all (non-dummy) endpoints of table-defined 2chains and table-defined permutation queries exist in $\mathcal{H}$. Combining this fact with Lemma 5 (d), we have $y_2^{(j)} \in \mathcal{H}$ for all $j = 1, 2, 3, 4$.

For the case of $x_2^{(j)}$ for $j = 1, \ldots, 4$, we can immediately deduce that $x_2^{(1)} \in \mathcal{P}$ by assumption that $(2, x_2^{(1)}, y_2^{(1)})$ is the last query to get table-defined (and by assumption that we are in a good execution of G_2 and all query cycles are safe by Lemma 21). For $x_2^{(j)}$ for $j \neq 1$, similar to the reasoning above, $(2, x_2^{(j)}, y_2^{(j)})$ was already table-defined at the beginning of the query cycle or it gets table-defined in the same query cycle (but before $(2, x_2^{(1)}, y_2^{(1)})$ by assumption). Then either $x_2^{(j)} \in \mathcal{H}$ if $(2, x_2^{(j)}, y_2^{(j)})$ was already table-defined at the beginning of the query cycle or $x_2^{(j)} \in \mathcal{P}$ if $(2, x_2^{(j)}, y_2^{(j)})$ gets table-defined in the current query cycle through ReadTape$(2, -, y_2^{(j)})$. (The reasoning for this is similar to prior reasoning for $x_2^{(1)}$.)

Then, if Eq. (9) holds, let t be the number of queries $(j, x_2^{(j)}, y_2^{(j)})$ that get table-defined in the current query cycle (where $1 \leq t \leq 4$). Recall that the queries $(j, x_2^{(j)}, y_2^{(j)})$ are distinct (and that $\mathcal{H}$ is a multiset). Then (a) if $t = 1$, we have $\mathcal{H}^{\oplus 7} \cap \mathcal{P} \neq \emptyset$, (b) if $t = 2$, we have $\mathcal{H}^{\oplus 6} \cap \mathcal{P}^{\oplus 2} \neq \emptyset$, (c) if $t = 3$, we have $\mathcal{H}^{\oplus 5} \cap \mathcal{P}^{\oplus 3} \neq \emptyset$ and (d) if $t = 4$, we have $\mathcal{H}^{\oplus 4} \cap \mathcal{P}^{\oplus 4} \neq \emptyset$. This implies that $\mathsf{BadPerm}$ occurs which contradicts our assumption that we are in a good execution of G_2.

$(4, 5)$-query cycle. Again, by referring to Table 1, we can see that in a $(4, 5)$-query cycle, a query at position 2 gets table-defined only through calls to Assign$(2, \cdot, \cdot)$ that occur in a procedure AdaptPath(C) for some triggered 2chain C. By assumption, $(2, x_2^{(1)}, y_2^{(1)})$ was (the last of the 4 queries to be) table-defined

in this query cycle. Say this was due to call to $\mathrm{Assign}(2, x_2^{(1)}, y_2^{(1)})$ that occurred in $\mathrm{AdaptPath}(C_1)$ for a 2chain $C_1 = (4, 5, y_4^{(1)}, x_5^{(1)}, k^{(1)})$ triggered in the current query cycle. By Lemma 10, we have $x_2^{(1)} = p_1(r(C_1)) \oplus k^{(1)}$ and $y_2^{(1)} = p_3^{-1}(\ell(C_1)) \oplus k^{(1)}$. Here, we have $p_1(r(C_1), p_3^{-1}(\ell(C_1)) \in \mathcal{P}$ since all query cycles are safe in a good execution of G_2.

Then, for all queries $(2, x_2^{(j)}, y_2^{(j)})$ $j = 2, 3, 4$ to be table-defined either the queries were table-defined at the beginning of the query cycle or they were table-defined in the current query cycle (earlier than $(2, x_2^{(1)}, y_2^{(1)})$ by assumption). Correspondingly, for $j = 2, 3, 4$ we either have $x_2^{(j)}, y_2^{(j)} \in \mathcal{H}$ (if $(2, x_2^{(j)}, y_2^{(j)})$ was table-defined at the beginning of the query cycle) or $x_2^{(j)} = p_1(r(C_j)) \oplus k^{(j)}$ and $y_2^{(j)} = p_3^{-1}(\ell(C_j)) \oplus k^{(j)}$ with $p_1(r(C_j), p_3^{-1}(\ell(C_j)) \in \mathcal{P}$ where $C_j = (4, 5, y_4^{(j)}, x_5^{(j)}, k^{(j)})$ is a 2chain triggered in the current query cycle (if $(2, x_2^{(j)}, y_2^{(j)})$ gets table-defined in the current query cycle).

Let t be the number of queries $(j, x_2^{(j)}, y_2^{(j)})$ that get table-defined in the current query cycle (where $1 \le t \le 4$). If $t = 1$, then $(2, x_2^{(1)}, y_2^{(1)})$ is the only query to be table-defined in the query cycle. Hence, we have $x_2^{(1)} \oplus y_2^{(1)} = p_1(r(C_1) \oplus p_3^{-1}(\ell(C_1)) \in \mathcal{P}^{\oplus 2}$ and $\sum_{j=2}^{4} x_2^{(j)} \oplus y_2^{(j)} \in \mathcal{H}^{\oplus 6}$. Thus, if Eq. (9) holds, $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 6} \neq \emptyset$ and hence $\mathsf{BadPerm}$ occurs.

If $t = 2$, assume without loss of generality that $(2, x_2^{(j)}, y_2^{(j)})$ are the queries to get table-defined in the query cycle where $j = 1, 2$. Then, if Eq. (9) holds, we have

$$\sum_{j=1}^{2} x_2^{(j)} \oplus y_2^{(j)} = \sum_{j=3}^{4} x_2^{(j)} \oplus y_2^{(j)} \tag{10}$$

where the LHS is equal to $\sum_{j=1}^{2} p_1(r(C_j) \oplus p_3^{-1}(\ell(C_j))$ and RHS is in $\mathcal{H}^{\oplus 4}$ by assumption. Since $p_1(r(C_j), p_3^{-1}(\ell(C_j)) \in \mathcal{P}$ for $j = 1, 2$, $\mathsf{BadPerm}$ occurs unless $r(C_1) = r(C_2)$ and $\ell(C_1) = \ell(C_2)$. If $r(C_1) = r(C_2)$ and $\ell(C_1) = \ell(C_2)$, then the LHS of Eq. (10) is 0. However, by Lemma 27, in a good execution of G_2, we cannot have two distinct queries $(2, x_2^{(1)}, y_2^{(1)})$ and $(2, x_2^{(2)}, y_2^{(2)})$ such that $\sum_{j=1}^{2} x_2^{(j)} \oplus y_2^{(j)} = 0$. Hence, if Eq. (10) holds, $\mathsf{BadPerm}$ occurs since either $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 4} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 4} \neq \emptyset$.

If $t = 3$, we assume without loss of generality that $(2, x_2^{(j)}, y_2^{(j)})$ are the queries to get table-defined in the query cycle where $j = 1, 2, 3$. Then, if Eq. (9) holds, we have

$$\sum_{j=1}^{3} x_2^{(j)} \oplus y_2^{(j)} = x_2^{(4)} \oplus y_2^{(4)} \tag{11}$$

where by assumption RHS of the equation above is in $\mathcal{H}^{\oplus 2}$ and the LHS is given by the following equation:

$$\sum_{j=1}^{3} x_2^{(j)} \oplus y_2^{(j)} = \sum_{j=1}^{3} p_1(r(C_j) \oplus p_3^{-1}(\ell(C_j)). \tag{12}$$

Since $p_1(r(C_j), p_3^{-1}(\ell(C_j)) \in \mathcal{P}$ for $j = 1, 2, 3$, BadPerm occurs (since we have either $\mathcal{P}^{\oplus 6} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ or $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$). This is because even if some of the right (resp. left) endpoints of the triggered 2chains C_j are equal, the terms $p_1(r(C_j))$ (resp. $p_3^{-1}(\ell(C_j))$) can cancel each other out in the sum on the RHS of Eq. (12) only if there are even number of right (resp. left) endpoints.

If $t = 4$, all queries $(2, x_2^{(j)}, y_2^{(j)})$ for $j = 1, \ldots, 4$ are table-defined in the current query cycle. Then, if Eq. (9) holds, we have

$$\sum_{j=1}^{4} x_2^{(j)} \oplus y_2^{(j)} = \sum_{j=1}^{4} p_1(r(C_j) \oplus p_3^{-1}(\ell(C_j)) = 0 \tag{13}$$

where $p_1(r(C_j), p_3^{-1}(\ell(C_j)) \in \mathcal{P}$ for $j = 1, 2, 3, 4$. Then, BadPerm occurs (since $\mathcal{H}^{\oplus 0} = 0$) unless the $p_1(\cdot)$ and $p_3^{-1}(\cdot)$ terms in Eq. (13) cancel with each other. For the terms in the sum in Eq. (13) to cancel out, we need 4 pairs (r_1, r_2), (r_3, r_4), (ℓ_1, ℓ_2) and (ℓ_3, ℓ_4) such that $r(C_{r_1}) = r(C_{r_2})$, $r(C_{r_3}) = r(C_{r_4})$, $\ell(C_{\ell_1}) = \ell(C_{\ell_2})$ and $\ell(C_{\ell_3}) = \ell(C_{\ell_4})$ where $r_i, \ell_i \in \{1, 2, 3, 4\}$ and $r_i \neq r_{i'}$ for $i \neq i'$, $\ell_i \neq \ell_{i'}$ for $i \neq i'$.

Note that if we have pairs $(r_i, r_{i'})$, $(\ell_j, \ell_{j'})$ such that $(r_i, r_{i'}) = (\ell_j, \ell_{j'})$ this implies that $r(C_{r_i}) = r(C_{r_{i'}})$ and $\ell(C_{r_i}) = \ell(C_{r_{i'}})$ which implies that the equation

$$x_2^{(r_i)} \oplus y_2^{(r_i)} \oplus x_2^{(r_{i'})} \oplus y_2^{(r_{i'})} = p_1(r(C_{r_i})) \oplus p_3^{-1}(\ell(C_{r_i})) \oplus p_1(r(C_{r_{i'}})) \oplus p_3^{-1}(\ell(C_{r_{i'}}))$$

evaluates to 0. However, this implies that in a good execution of G_2 there exist two distinct queries $(2, x_2^{(r_i)}, y_2^{(r_i)})$ and $(2, x_2^{(r_{i'})}, y_2^{(r_{i'})})$ such that $x_2^{(r_i)} \oplus y_2^{(r_i)} \oplus x_2^{(r_{i'})} \oplus y_2^{(r_{i'})} = 0$ which contradicts Lemma 27.

Then, without loss of generality, we can assume that the 4 pairs (r_1, r_2), (r_3, r_4), (ℓ_1, ℓ_2) and (ℓ_3, ℓ_4) are respectively $(2, 3)$, $(4, 1)$, $(1, 2)$ and $(3, 4)$ i.e. $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_3)$, $\ell(C_3) = \ell(C_4)$ and $r(C_4) = r(C_1)$. Recall that C_j are the triggered 2chains such that $(2, x_2^{(j)}, y_2^{(j)})$ was table-defined in Assign$(2, x_2^{(j)}, y_2^{(j)})$ made in the procedure AdaptPath(C_j). To analyse when Eq. (13) can hold, we distinguish between three cases: (i) all cipher queries $T^{-1}(k^{(j)}, y_5^{(j)})$ are table-defined at the beginning of the query cycle, (ii) exactly one cipher query (say) $T^{-1}(k^{(1)}, y_5^{(1)})$ is table-defined during the query cycle and (iii) more than one cipher query $T^{-1}(k^{(j)}, y_5^{(j)})$ gets table-defined in the current query cycle.

In case (i), if all cipher queries $T^{-1}(k^{(j)}, y_5^{(j)})$ are table-defined at the beginning of the query cycle, then we observe that there exists a 4cycle at $(4, 5)$. This is because (a) all four triggered 2chains C_j are distinct by assumption that the queries $(2, x_2^{(j)}, y_2^{(j)})$ are distinct, (b) the endpoints of all triggered 2chains C_j are non-dummy and table-undefined since the cipher queries are table-defined at the beginning of the query cycle and since all the query cycles are safe in a good execution by Lemma 21 and (c) we have $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_3)$, $\ell(C_3) = \ell(C_4)$ and $r(C_4) = r(C_1)$ since (r_1, r_2), (r_3, r_4), (ℓ_1, ℓ_2) and (ℓ_3, ℓ_4) are

respectively $(2,3)$, $(4,1)$, $(1,2)$ and $(3,4)$. However, by Lemma 26, in a good execution of G_2, there does not appear a 4cycle at $(4,5)$. Hence, this contradicts our assumption that we are in a good execution of G_2.

In case (ii), exactly one cipher query (say) $T^{-1}(k^{(1)}, y_5^{(1)})$ is table-defined during the query cycle. By assumption, $r(C_1) = r(C_4)$ and $T^{-1}(k^{(4)}, x_5^{(4)})$ is table-defined at the beginning of the query cycle. Also, in a $(4,5)$-query cycle cipher queries are only of the form $(-,\cdot,\cdot)$ i.e. only inverse cipher queries are made. Then, we have $r(C_1) \in \mathcal{C}$ and $r(C_4) \in \mathcal{H}$. Hence BadIC occurs as $\mathcal{C}^{\oplus 1} \cap \mathcal{H}^{\oplus 1} \neq \emptyset$.

In case (iii), more than one cipher query $T^{-1}(k^{(j)}, y_5^{(j)})$ gets table-defined in the current query cycle. Since we have $r(C_2) = r(C_3)$ and $r(C_4) = r(C_1)$, without loss of generality, it is sufficient to analyse the sub-case where both $r(C_4)$ and $r(C_1)$ get table-defined in the current query cycle. The analysis for this sub-case plus case (ii) above sufficiently captures all cases where more than one cipher query $T^{-1}(k^{(j)}, y_5^{(j)})$ gets table-defined in the current query cycle. So, let us consider the case where both $T^{-1}(k^{(1)}, y_5^{(1)})$ and $T^{-1}(k^{(4)}, x_5^{(4)})$ get table-defined in the current query cycle. Since $r(C_1) = r(C_4)$, we have $T^{-1}(k, x_5) = T^{-1}(k', x_5')$ where $r(C_1) = T^{-1}(k^{(1)}, x_5^{(1)}) \in \mathcal{C}$ and $r(C_4) = T^{-1}(k^{(4)}, x_5^{(4)}) \in \mathcal{C}$. These are distinct cipher queries since $C_1 \neq C_4$. Hence, BadIC occurs as $0 \in \mathcal{C}^{\oplus 2}$ implying that $\mathcal{C}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$.

(5, 1)-query cycle. By referring to Table 1, we can see that in a $(5,1)$-query cycle, a query at position 2 gets table-defined only through calls to $\mathrm{ReadTape}(2,+,\cdot)$. For queries $(2, x_2^{(j)}, y_2^{(j)})$ where $j = 2,\ldots,4$, either $(2, x_2^{(j)}, y_2^{(j)})$ was already table-defined at the beginning of the query cycle or it gets table-defined in the same query cycle. Hence, we have $x_2^{(j)} \in \mathcal{H}$ for all $j = 1,2,3,4$. This is because either $(2, x_2^{(j)}, y_2^{(j)})$ is table-defined at the beginning of the query cycle or $(2,+,x_2^{(j)})$ is a pending query and hence gets table-defined in the current query cycle. By definition of history $\mathcal{H}$, the initiating query, all (non-dummy) endpoints of table-defined 2chains and table-defined permutation queries exist in $\mathcal{H}$. Combining this fact with Lemma 5 (d), we have $x_2^{(j)} \in \mathcal{H}$ for all $j = 1,2,3,4$. The rest of the analysis for the case of a $(5,1)$-query cycle proceeds analogous to that of a $(3,4)$-query cycle.

Concluding, in a good execution of G_2, there do not exist four distinct queries $(i, x_i^{(j)}, y_i^{(j)})$ such that $\sum_{j=1}^{4} x_i^{(j)} \oplus y_i^{(j)} = 0$ when $i = 2$. A similar case analysis works for the case when $i = 4$. □

5.4.2 Termination Proof

We now analyze the running time of the simulator in a good execution of G_2. A large part of this analysis consists in upper bounding the size of tables T, T^{-1}, P_i, P_i^{-1}. Since one always has $|T| = |T^{-1}|$ and $|P_i| = |P_i^{-1}|$, we will only state the results for T and P_i.

Note that,during a query cycle, any triggered 2chain C can be associated with the query that became pending just before C was triggered and, reciprocally, any

pending query (i, δ, z), except the initiating query, can be associated with the 2chain C that was triggered just before (i, δ, z) became pending. We make these observations formal through the following definitions.

Definition 20. During a query cycle, we say that a 2chain C is *triggered by query* (i, δ, z) if it is added to $\mathsf{Triggered}$ during a call to FindNewPaths(i, δ, z). We say C is an (i, δ)-triggered 2chain if it is triggered by a query of type (i, δ).

By Lemma 5 (b), a triggered $(i, i+1)$-2chain is either $(i-1, -)$- or $(i+2, +)$-triggered. For brevity, we group four special types of triggered 2chains under a common name.

Definition 21. A (triggered) *wrapping* 2chain is either

- a $(4, 5)$-2chain that was $(1, +)$-triggered,
- a $(1, 2)$-2chain that was $(5, -)$-triggered,
- a $(5, 1)$-2chain that was either $(2, +)$- or $(4, -)$-triggered.

Note that wrapping 2chains are exactly those for which the simulator makes a call to procedure Check to decide whether to trigger the 2chain or not.

Definition 22. Consider a query cycle with initiating query (i_0, δ_0, z_0) and a permutation query $(i, \delta, z) \neq (i_0, \delta_0, z_0)$ which becomes pending. We call the (unique) 2chain that was triggered just before (i, δ, z) became pending the *2chain associated with* (i, δ, z).

Note that uniqueness of the 2chain associated with a non-initiating pending query follows easily from the checks at lines 57 and 75.

The following two lemmas will be useful in the rest of the proof.

Lemma 29. *Consider a good execution of* G_2*, and assume that a complete path exists at the end of the execution. Then at most one of the five 2chains belonging to the complete path has been triggered during the execution.*

Proof. Consider a complete path Π existing at the end of the execution, and consider the first 2chain C belonging to Π which was triggered (if any) at some point in the execution. Since the simulator does not abort in a good execution, by Lemma 5 (e), at the end of the query cycle where C is triggered, C belongs to a complete path which must be Π by Lemma 4. By Lemma 20, this implies that neither C nor any other 2chain belonging to Π will ever be triggered in any subsequent query cycle. □

Lemma 30. *For* $i \in \{1, \ldots, 5\}$*, the number of table-defined permutation queries* (i, x_i, y_i) *during an execution of* G_2 *can never exceed the sum of*

- *the number of distinguisher's calls to* $\mathrm{Query}(i, \cdot, \cdot)$*,*
- *the number of* $(i+1, i+2)$*-2chains that were* $(i+3, +)$*-triggered,*
- *the number of* $(i-2, i-1)$*-2chains that were* $(i-3, -)$*-triggered,*
- *the number of* $(i+2, i+3)$*-2chains that were either* $(i+1, -)$*- or* $(i+4, +)$*-triggered.*

Proof. Entries are added to P_i/P_i^{-1} either by a call to ReadTape during an $(i+1,i+2)$- or an $(i-2,i-1)$-query cycle or by a call to AdaptPath during an $(i+2,i+3)$-query cycle (see Table 1).

Consider first entries that were added by a call to ReadTape during an $(i+1,i+2)$- or an $(i-2,i-1)$-query cycle. Clearly, the number of such table-defined queries cannot exceed the sum of the total number $N_{i,+}$ of queries of type $(i,+)$ that became pending during an $(i-2,i-1)$-query cycle and the total number $N_{i,-}$ of queries of type $(i,-)$ that became pending during an $(i+1,i+2)$-query cycle. Now, $N_{i,+}$ cannot exceed the sum of the total number of initiating and non-initiating pending queries of type $(i,+)$ over all $(i-2,i-1)$-query cycles. The total number of initiating queries of type $(i,+)$ is clearly at most the number of distinguisher's calls to $\mathrm{Query}(i,+,\cdot)$, while the total number of non-initiating pending queries of type $(i,+)$ over all $(i-2,i-1)$-query cycles cannot exceed the total number of $(i-3,-)$-triggered 2chains (since clearly a non-initiating pending query of type $(i,+)$ cannot be associated with an $(i,+)$-triggered $(i-2,i-1)$-2chain). Similarly, $N_{i,-}$ cannot exceed the sum of the total number of distinguisher's call to $\mathrm{Query}(i,-,\cdot)$ and the total number of $(i-3,-)$-triggered $(i+1,i+2)$-2chains. All in all, we see that the total number of triples (i,x_i,y_i) that became table-defined because of a call to ReadTape cannot exceed the sum of

- the number of distinguisher's calls to $\mathrm{Query}(i,\cdot,\cdot)$,
- the number of $(i+1,i+2)$-2chains that were $(i+3,+)$-triggered,
- the number of $(i-2,i-1)$-2chains that were $(i-3,-)$-triggered.

Consider now a triple (i,x_i,y_i) which became table-defined during a call to AdaptPath in an $(i+2,i+3)$-query cycle. Clearly, the total number of such triples cannot exceed the total number of $(i+2,i+3)$-2chains that are triggered over all $(i+2,i+3)$-query cycles (irrespectively of whether they are $(i+1,-)$- or $(i+4,+)$-triggered). The result follows. □

The following lemma contains the standard "bootstrapping" argument introduced in [19] that has been used to prove termination of all simulators for the Feistel and the IEM constructions since then.

Lemma 31. *In a good execution of* G_2*, at most* q *wrapping 2chains get triggered in total.*

Proof. We show that there is a one-to-one mapping from the set of triggered wrapping 2chains to the set of distinguisher's call to Enc/Dec. The lemma will follow from the assumption that the distinguisher makes at most q cipher queries.

By inspection of the pseudocode, we see that for each triggered wrapping 2chain C, there is a triple (k,x_1,y_5) such that $\mathrm{Check}(k,x_1,y_5)$ was true, i.e., (k,x_1,y_5) was table-defined when C was triggered. We show that this mapping is one-to-one, i.e., no two distinct wrapping 2chains can be triggered by the same triple (k,x_1,y_5). For this, note that there is exactly one 2chain of each type in $\{(1,2),(4,5),(5,1)\}$ which can be triggered by a specific call $\mathrm{Check}(k,x_1,y_5)$. Hence, this is clear for two 2chains of the same type, while for two 2chains of

distinct type this follows from the fact that once a chain C of a specific type was triggered by a call to Check(k, x_1, y_5), this 2chain will be complete (unless the simulator aborts) and the two other 2chains of remaining types which *could* be triggered by the same call to Check(k, x_1, y_5) belong to the same complete path as C, hence cannot be triggered by Lemma 20.

Hence, each triggered wrapping 2chain can be mapped to a distinct table-defined cipher query (k, x_1, y_5). This cipher query became table-defined because of call to Enc/Dec made either by the distinguisher or the simulator. It remains to show that the corresponding call cannot have been made by the simulator. But this follows easily from Lemma 6 since when the simulator makes a cipher query which modifies T/T^{-1}, the corresponding 2chain has already been triggered. Hence it will be complete at the end of the query cycle and cannot be triggered again by Lemma 29. □

Lemma 32. *In a good execution of* G_2*, one always has* $|P_3| \leq 2q$*.*

Proof. By Lemma 30, the number of table-defined permutation queries $(3, x_3, y_3)$ (and hence the size of P_3) cannot exceed the sum of

- the number of distinguisher's calls to Query$(3, \cdot, \cdot)$,
- the number of $(4, 5)$-2chains that were $(1, +)$-triggered,
- the number of $(1, 2)$-2chains that were $(5, -)$-triggered,
- the number of $(5, 1)$-2chains that were either $(2, +)$- or $(4, -)$-triggered.

The number of entries of the first type is at most q by the assumption that the distinguisher makes at most q oracle queries to each permutation. On the other hand, note that any 2chain mentioned for the three other types are wrapping 2chains. Hence, by Lemma 31, there are at most q such entries in total, so that $|P_3| \leq 2q$. □

Lemma 33. *In a good execution of* G_2*, the sum of the total numbers of* $(3, -)$*- and* $(5, +)$*-triggered 2chains, resp. of* $(1, -)$*- and* $(3, +)$*-triggered 2chains, is at most* $6q^2 - 2q$*.*

Proof. Let C be a 2chain which is either $(3, -)$- or $(5, +)$-triggered during the execution. (The case of $(1, -)$- or $(3, +)$-triggered 2chains is similar by symmetry.) By Lemma 5 (e), C belongs to a complete path $((1, x_1, y_1), \ldots, (5, x_5, y_5))$ at the end of the execution (since the simulator does not abort), and $C = (3, 4, y_3, x_4, k)$ if it was $(5, +)$-triggered, whereas $C = (4, 5, y_4, x_5, k)$ if it was $(3, -)$-triggered.

Note that when C was triggered, $(5, +, x_5)$ was necessarily table-defined or pending. This is clear if $C = (4, 5, y_4, x_5, k)$ was $(3, -)$-triggered since a triggered 2chain must be table-defined. If $C = (3, 4, y_3, x_4, k)$ was $(5, -)$-triggered, then it was necessarily during the call to FindNewPaths$(5, +, x_5)$ which implies that x_5 was pending.

We now distinguish two cases depending on how $(5, +, x_5)$ became table-defined or pending. Assume first that this was because of a distinguisher's call to Query$(5, \cdot, \cdot)$. There are at most q such calls, hence there are at most q possibilities for x_5. There are at most $2q$ possibilities for y_3 by Lemma 32. Moreover, for each

possible pair (y_3, x_5), there is at most one possibility for the table-defined query $(4, x_4, y_4)$ since otherwise this would contradict Lemma 27 (note that one must have $x_4 \oplus y_4 = y_3 \oplus x_5$). Hence there are at most $2q^2$ possibilities in that case.

Assume now that this $(5, +, x_5)$ was a non-initiating pending query in the same query cycle where C was triggered, or became table-defined during a previous query cycle than the one where C was triggered and for which $(5, +, x_5)$ was neither the initiating query nor became table-defined during the ReadTape call for the initiating query. In all cases there exists a table-defined $(3, 4)$-2chain $C' = (3, 4, y'_3, x'_4, k')$ distinct from $(3, 4, y_3, x_4, k)$ such that $x_5 = r(C') = y'_4 \oplus x'_4 \oplus y'_3$. Since we also have $x_5 = y_4 \oplus x_4 \oplus y_3$, we obtain $x_4 \oplus y_4 \oplus x'_4 \oplus y'_4 = y_3 \oplus y'_3$. If $y_3 = y'_3$, by Lemma 27 we have $x_4 = x'_4$ and $C' = (3, 4, y_3, x_4, k)$, which cannot be. On the other hand, for a fixed (orderless) pair of $y_3 \neq y'_3$, the (orderless) pair of $(4, x_4, y_4)$ and $(4, x'_4, y'_4)$ is unique by Lemmas 27 and 28 (otherwise, one of the lemmas must be violated by the two pairs). There are at most $\binom{2q}{2} = q(2q-1)$ choices of y_3 and y'_3; for each pair there is at most one (orderless) pair of $(4, x_4, y_4)$ and $(4, x'_4, y'_4)$, so there are 2 ways to combine the queries to form two 2chains. Moreover, C' must either have been completed during a previous query cycle than the one where C is triggered, or must have been triggered *before* C in the same query cycle and have made x_5 pending (in which case C was triggered by $(5, +, x_5)$). Thus each way to combine y_3, y'_3, $(4, x_4, y_4)$ and $(4, x'_4, y'_4)$ to form two 2chains corresponds to at most one $(3, +)$- or $(5, -)$-triggered 2chain, so at most $4q^2 - 2q$ such 2chains are triggered this way.

Combining the two cases, the number of $(3, -)$- or $(5, +)$-triggered 2chains is at most $6q^2 - 2q$. □

Lemma 34. *In a good execution of* G_2*, one always has* $|P_2| \leq 6q^2$ *and* $|P_4| \leq 6q^2$.

Proof. By Lemma 30, the number of table-defined queries $(2, x_2, y_2)$ (and hence the size of P_2) cannot exceed the sum of

- the number of distinguisher's calls to $\mathrm{Query}(2, \cdot, \cdot)$,
- the number of $(3, 4)$-2chains that were $(5, +)$-triggered,
- the number of $(5, 1)$-2chains that were $(4, -)$-triggered,
- the number of $(4, 5)$-2chains that were either $(3, -)$- or $(1, +)$-triggered.

There are at most q entries of the first type by the assumption that the distinguisher makes at most q oracle queries. Note that any 2chain mentioned for the other cases are either wrapping, $(3, -)$-triggered, or $(5, +)$-triggered 2chains. Hence, by Lemmas 31 and 33, there are at most $q + 6q^2 - 2q$ entries of the three other types in total. Thus, we have $|P_2| \leq q + q + 6q^2 - 2q = 6q^2$. Symmetrically, $|P_4| \leq 6q^2$. □

Lemma 35. *In a good execution of* G_2*, at most* $12q^3$ *2chains are triggered in total.*

Proof. Since the simulator does not abort in good executions by Lemma 22, any triggered 2chain belongs to a complete path at the end of the execution. Moreover, by Lemma 29, at most one of the five 2chains belonging to a complete

path is triggered in a good execution. Hence, there is a bijective mapping from the set of triggered 2chains to the set of complete paths existing at the end of the execution. Consider all $(3,4)$-2chains which are table-defined at the end of the execution. Each such 2chain belongs to at most one complete path by Lemma 4. Hence, the number of complete paths at the end of the execution cannot exceed the number of table-defined $(3,4)$-2chains, which by Lemmas 32 and 34 is at most $2q \cdot 6q^2 = 12q^3$. □

Lemma 36. *In a good execution of* G_2*, we have* $|T| \leq 12q^3 + q$.

Proof. Recall that the table T is used to maintain the cipher queries that have been issued. In G_2, no new cipher query is issued in Check called in procedure Trigger. So the simulator issues a table-undefined cipher query only if the path containing the cipher query has been triggered. The number of triggered paths is at most $12q^3$, while the distinguisher issues at most q cipher queries. Thus the number of table-defined cipher queries is at most $12q^3 + q$. □

Lemma 37. *In a good execution of* G_2*, one always has* $|P_1| \leq 12q^3 + q$ *and* $|P_5| \leq 12q^3 + q$.

Proof. By Lemma 30, the number of table-defined queries $(1, x_1, y_1)$ (and hence the size of P_1) cannot exceed the sum of the number of distinguisher's call to $\mathrm{Query}(1,\cdot,\cdot)$, which is at most q, and the total number of triggered 2chains, which is at most $12q^3$ by Lemma 35. Therefore, the size of $|P_1|$ is at most $12q^3 + q$. The same reasoning applies to $|P_5|$. □

Lemma 38. *In good executions of* G_2*, the simulator runs in time* $O(q^8)$ *and uses* $O(q^3)$ *space.*

Proof. By Lemmas 32, 34 and 37, the total number of queries that are defined during an execution is $O(q^3)$. In a non-aborting execution, every time Assign is called an undefined query becomes defined. Therefore, Assign is called $O(q^3)$ times, which implies ReadTape and AdaptPath are called $O(q^3)$ times. The aforementioned procedures run in constant time, so the total running time of them is $O(q^3)$.

The procedure FindNewPaths is called once for each pending query. In a non-aborting execution, pending queries become distinct defined queries at the end of the execution, which implies the total number of pending queries in position i is upper bounded by $|P_i|$ at the end of the proof. In a call to $\mathrm{FindNewPaths}(i, \delta, z)$, each iteration runs in constant time; the running time of the call is either $|P_{i-2}| \cdot |P_{i-1}|$ or $|P_{i+1}| \cdot |P_{i+2}|$[11]. Take the sum over the positions, the total running time of FindNewPaths is at most $2\sum_i |P_i| \cdot |P_{i+1}| \cdot |P_{i+2}|$, which is $O(q^8)$ by Lemmas 32, 34 and 37.

[11] Here P_{i-2}, P_{i-1}, P_{i+1} and P_{i+2} refer to the states of the tables when FindNewPaths is called, which is different from the previous P_i referring to the state at the end of the execution. In this proof we will not distinguish between the states at different time points, since they are all subject to Lemmas 32, 34 and 37.

The tables P_i, P_i^{-1}, $\mathsf{Pending}$ and Paths have size at most $O(q^3)$, and the local variables use constant space. Thus the simulator uses $O(q^3)$ space in total. □

Theorem 39. *An execution of* G_2 *is good with probability at least* $1 - 10^{22}q^{38}/2^n$.

Proof. If $13q^3 \geq 2^{n-1}$, the lemma trivially holds. Thus we can assume $13q^3 < 2^{n-1}$ in this proof.

By Lemmas 32, 34, 37 and 36 and by Definition 6, the number of table-defined $(5,1)$-2chains is upper bounded by $12q^2 + q \leq 13q^3$ (since each cipher query in T uniquely determines a $(5,1)$-2chain), and the total number of table-defined $(i, i+1)$-2chains for $i = 1, 2, 3, 4$ is at most $2((12q^3 + q) \cdot 6q^2 + 6q^2 \cdot 2q) \leq 180q^5$ (since each pair of table-defined queries uniquely determine an $(i, i+1)$-2chain). There are at most $40q^3$ table-defined permutation queries and $13q^3$ table-defined cipher queries. By Definition 10, the size of $\mathcal{H}$ can be upper bounded by

$$3 \cdot (13q^3 + 180q^5) + 2 \cdot 40q^3 + 3 \cdot 13q^3 \leq 698q^5.$$

Now we consider the entries of p_i/p_i^{-1} and $\mathsf{ic}/\mathsf{ic}^{-1}$ that have been read, in the same order as they are read in the execution, and compute the probability that $\mathsf{BadPerm}$ or BadIC occurs when each entry is read. More precisely, we want to upper bound the probability that:

- (cf. Definition 13) an entry read from p_i/p_i^{-1} is in $\mathcal{P}^{\oplus j} \oplus \mathcal{H}^{\oplus \ell}$ for some $j, \ell \geq 0$ such that $j + \ell \leq 7$, where $\mathcal{P}$ refers to the random values that *has already been sampled* from permutation tapes in the current query cycle;
- (cf. Definition 14) an entry read from $\mathsf{ic}/\mathsf{ic}^{-1}$ is in $\mathcal{H}$ or in $\mathcal{C}$, where $\mathcal{C}$ refers to the random values that *has already been sampled* from the cipher tape in the current query cycle.

We note that the bounds in Lemmas 32, 34, 37 and 36 hold as long as no bad event has occurred in the execution (indeed, the only property of good executions used in the proof is that "the bad events never occur"). Therefore, unless a bad event occurs, the execution terminates before the numbers of entries read from p_i/p_i^{-1} (resp. $\mathsf{ic}/\mathsf{ic}^{-1}$) exceeds $40q^3$ (resp. $13q^3$), and the bounds on $\mathcal{H}$, $\mathcal{P}$ and $\mathcal{C}$ for good executions also hold.

Assume no bad event has occurred in the execution, and consider a call to ReadTape in which $p_i(x_i)$ is read (the case of p_i^{-1} is symmetric). At this point, less than $13q^3$ entries of p_i/p_i^{-1} has been read, so the value of $p_i(x_i)$ is uniformly distributed among at least $2^n - 13q^3 \geq 2^{n-1}$ values, where the inequality is due to the assumption that $13q^3 \leq 2^{n-1}$. The total size of the sets $\mathcal{P}^{\oplus j} \oplus \mathcal{H}^{\oplus \ell}$ for $j + \ell \leq 7$ is at most $(700q^5)^7$,[12] so the probability that $\mathsf{BadPerm}$ occurs is $(700q^5)^7/2^{n-1}$.

Similarly, we can show the probability that BadIC occurs on a call to Enc/Dec is at most $700q^5/2^{n-1}$.

[12] Note that since the entries of $\mathcal{P}$ will be added to $\mathcal{H}$ after the query cycle, the total size of $\mathcal{P}$ and $\mathcal{H}$ at this point of execution is also upper bounded by the upper bound on $|\mathcal{H}|$, $698q^5$. It is easy to see $\sum_{i=0}^{7}(698q^5)^i \leq (700q^5)^7$.

With a union bound on the at most $40q^3$ calls to ReadTape and $13q^3$ calls to Enc/Dec, the probability that bad events occur in an execution is at most

$$40q^3 \cdot \frac{700^7 q^{35}}{2^{n-1}} + 13q^3 \cdot \frac{700q^5}{2^{n-1}} \leq 10^{22} \cdot \frac{q^{38}}{2^n}. \qquad \square$$

5.5 Transition from G_1 to G_2

Recall that the only difference between G_1 and G_2 is in procedure Check (line 32): in G_2, it does not call Enc and hence does not modify tables T/T^{-1}.

We say two executions of G_1 and G_2 are *identical* if every procedure call returns the same value in the two executions. In particular, the view of the distinguisher $\mathcal{D}$ is the same in the two executions, so $\mathcal{D}$ outputs the same value in identical executions.

Lemma 40. *For randomly chosen tapes $\mathcal{T} = (p_1, \ldots, p_5, \mathsf{ic})$, the probability that the execution of G_2 with $\mathcal{T}$ is good and that the executions of G_1 and G_2 with the same tapes $\mathcal{T}$ are identical is at least $1 - (10^{22}q^{38}/2^n + 2704q^8/2^n)$.*

Proof. The bound trivially holds if $q^3 > 2^{n-2}$, so we assume $q^3 \leq 2^{n-2}$.

Recall that the only difference between G_1 and G_2 is in Check. The only side effect of Check is that the call to Enc may add a new entry to the tables T/T^{-1}. The tables T/T^{-1} are only used in Enc, Dec and the G_2-version of Check; moreover, the answers of Enc and Dec are always consistent with the cipher encoded by ic regardless of the state of T/T^{-1}. Therefore, if every call to Check returns the same value in the two executions, the two executions can never "diverge" and are identical.

Observe that a call to $\text{Check}(k, x_1, y_5)$ returns different value in the two executions only if $\mathsf{ic}(k, x_1) = y_5$ and $(k, x_1) \notin T$ in the execution of G_2. Since the event can be characterized in G_2 only, we will compute its probability by considering a G_2-execution with random tapes. We will assume the execution of G_2 is good when computing the probability of divergence.

As discussed above, divergence would not occur if $(k, x_1) \in T$ at the moment $\text{Check}(k, x_1, y_5)$ is called. This implies that the cipher tape entry $\mathsf{ic}(k, x_1)$ hasn't been read in the execution, since the tape ic is only read in Enc and Dec, where a corresponding entry is immediately added to T. Therefore, the value of $\mathsf{ic}(k, x_1)$ is uniformly distributed over $\{0,1\}^n \setminus \{y \mid y = T(k, x) \text{ for some } x\}$. By Lemma 36, the size of T is at most $12q^3 + q \leq 13q^3$, so the probability that $\mathsf{ic}(k, x_1) = y_5$ is at most $1/(2^n - 13q^3)$. Moreover, if $\text{Check}(k, x_1, y_5)$ is called multiple times and divergence doesn't occur in the first call, then divergence wouldn't occur in the subsequent calls to $\text{Check}(k, x_1, y_5)$. To upper bound the probability of divergence, we only need to consider the first call to Check with each argument.

The procedure Check is only called in FindNewPaths. It is easy to see from Fig. 5 that if $\text{Check}(k, x_1, y_5)$ is called, we either have $k = y_4 \oplus x_5$ and the three queries $(4, -, y_4)$, $(5, x_5, y_5)$ and $(1, +, x_1)$ are pending or defined, or have $k = y_1 \oplus x_2$ and the three queries $(5, -, y_5)$, $(1, x_1, y_1)$ and $(2, +, x_2)$ are pending

or defined. Since good executions don't abort, the pending queries are defined at the end of the execution. By Lemmas 34 and 37, there are

$$(12q^3+q)^2 \cdot 6q^2 + (12q^3+q)^2 \cdot 6q^2 \leq 2028q^8$$

ways to choose three defined queries in positions $(4,5,1)$ or $(5,1,2)$.

With a union bound over all distinct arguments, the probability that divergence occurs in the first call to Check with some argument is at most $2028q^8/(2^n - 13q^3) \leq 2704q^8/2^n$, where the inequality is due to the assumption that $q^3 \leq 2^{n-2}$.

With a union bound, the probability that either the execution of G_2 is bad or the executions of G_1 and G_2 diverge is at most

$$10^{22}q^{38}/2^n + 2704q^8/2^n. \qquad \square$$

Lemma 41. *We have*

$$\Delta_{\mathcal{D}}(\mathsf{G}_1, \mathsf{G}_2) \leq 10^{22}q^{38}/2^n + 2704q^8/2^n.$$

Proof. Since $\mathcal{D}$ outputs the same value in identical executions of G_1 and G_2, this is a direct corollary of Lemma 40. $\square$

Theorem 42. *With an optimized implementation, the simulator runs in time $O(q^5)$ and makes at most $O(q^5)$ queries to the cipher oracle in the simulated world G_1.*

Proof. By Lemma 38, the simulator runs in time $O(q^8)$ and uses $O(q^3)$ space in good executions of G_2. The simulator has the same running time in identical executions of G_1 and G_2, so by Lemma 40, the simulator runs in time $O(q^8)$ with high probability in G_1.

Using a similar trick as [20], we can trade off more space for a better running time. In particular, the simulator can be implemented to run in $O(q^5)$ time and $O(q^5)$ space. In the following proof, we consider the running time of a G_1-execution that is identical to a good execution of G_2, so we can use the bounds proved in good executions of G_2.[13] Note that the optimized simulator always has the same behavior as the one described in the pseudocode, even in "bad" executions.

In the proof of Lemma 38, the running time is dominated by FindNewPaths. We observe that it is unnecessary to check every pair of table-defined queries in FindNewPaths. Indeed, the simulator only needs to go through defined queries in the adjacent position, and the query in the next position is fixed and can be computed. E.g., in a call to FindNewPaths$(2,+,x_2)$, instead of iterating through every pair in $P_1 \times P_5^{-1}$, the simulator can iterate through $x_1 \in P_1$, compute $y_5 := \mathrm{Enc}(y_1 \oplus x_2, x_1)$, and check if $y_5 \in P_5^{-1}$.

However, the trick doesn't apply to calls of the form $(1,+,*)$ or $(5,-,*)$, since the simulator doesn't know if $\mathrm{Enc}(k,x_1) = y_5$ for some key k. To handle

[13] Most of the bounds hold in G_1, except for the bound on the size of T since the simulator issues more queries to the cipher oracle in G_1.

these calls, the simulator maintains a hash table that maps each (table-undefined) query $(1, +, x_1)$ or $(5, -, y_5)$ to the 2chains it should trigger. Specifically, a query $(1, +, x_1)$ is mapped to a set of pairs $(y_4, x_5) \in P_4^{-1} \times P_5$ such that $\mathrm{Dec}(y_4 \oplus x_5, y_5) = x_1$, and $(5, -, y_5)$ is mapped to a set of $(x_1, x_2) \in P_1 \times P_2$ such that $\mathrm{Enc}(y_1 \oplus x_2, x_1) = y_5$. Then in a call to FindNewPaths$(1, +, *)$ or FindNewPaths$(5, -, *)$, it only takes a table lookup to find all triggered 2chains.

The table should be updated every time a query at position 1, 2, 4 or 5 becomes table-defined. For example, when a query $(1, x_1, y_1)$ is defined, for each table-defined query $(2, x_2, y_2)$ the set mapped from $(5, -, y_5)$ should be updated, where $y_5 \coloneqq \mathrm{Enc}(y_1 \oplus x_2, x_1)$. By Lemmas 34 and 37, there are $O(q^5)$ pairs of defined queries in positions $(1, 2)$ or $(4, 5)$, thus the size of the table is $O(q^5)$ and it takes $O(q^5)$ time to update.

The new implementation of FindNewPaths now only takes $O(q^5)$ time: There are $O(q^3)$ calls of form $(1, -, *)$ or $(5, +, *)$, in which P_2 or P_4 is traversed and each takes $O(q^2)$ running time. The $O(q^3)$ calls of form $(1, +, *)$ or $(5, -, *)$ take $O(1)$ time to find triggered 2chains; there are at most $O(q^3)$ triggered 2chains throughout the execution (cf. Lemma 35), so handling the triggered 2chains uses $O(q^3)$ running time. There are $O(q^2)$ calls at positions 2, 3 or 4, and each call uses $O(q^3)$ time to traverse a table.

Recall in the proof of Lemma 38 that the other procedures runs in $O(q^3)$ time. Therefore, the total running time of the optimized implementation is still dominated by FindNewPaths but has been improved to $O(q^5)$. Since each cipher query takes constant time, the upper bound on the running time implies that at most $O(q^5)$ cipher queries are issued by the simulator.

Finally, note that the above bound is only proved for the case when the G_1-execution is identical to a good G_2-execution. However, since this holds except with negligible probability (cf. Lemma 40) and since the simulator knows the value of q (cf. Definition 1), we can let the simulator abort when the running time or the number of queries exceeds the corresponding bound. Then the simulator is efficient with probability 1 and the change affects an execution only with negligible probability. □

5.6 Transition from G_2 to G_4

In this section, we will use a randomness mapping argument to prove the indistinguishability of G_2 and G_4.

We start with a standard randomness mapping from G_2 to G_3, which has a similar structure to the randomness mapping in [20].

Additional Assumption on $\mathcal{D}$. Some of the lemmas in this section only hold when the distinguisher *completes all paths*, as defined below:

Definition 23. A distinguisher $\mathcal{D}$ *completes all paths* if at the end of every non-aborting execution, $\mathcal{D}$ has made queries $\mathrm{Query}(i, +, x_i) = y_i$ or $\mathrm{Query}(i, -, y_i) =$

x_i for $i = 1, 2, 3, 4, 5$ where $x_i = y_{i-1} \oplus k$ for $i = 2, 3, 4, 5$, for every pair of k and x_1 such that $\mathcal{D}$ has queried $\mathrm{Enc}(k, x_1) = y'_5$ or $\mathrm{Dec}(k, y'_5) = x_1$.[14]

Note that for an arbitrary distinguisher $\mathcal{D}$, it is easy to construct an equivalent distinguisher $\mathcal{D}'$ that completes all paths and that always outputs the same value as $\mathcal{D}$: Let $\mathcal{D}'$ run $\mathcal{D}$ until $\mathcal{D}$ outputs a value b. For each cipher query that has been made by $\mathcal{D}$, $\mathcal{D}'$ issues the permutation queries $\mathrm{Query}(i, +, x_i)$ to "complete the path" as in Definition 23. Finally $\mathcal{D}'$ outputs b, ignoring the answers of the extra queries. The distinguisher $\mathcal{D}'$ issues at most q extra queries in each position.

In the rest of this section (except for Lemma 51), we will assume without loss of generality that the distinguisher $\mathcal{D}$ completes all paths (the assumption is without loss of generality with the cost of a multiplicative factor in the final bound). The definitions and lemmas that only hold under this assumption will be marked with (*).

FOOTPRINTS. The random permutation tapes are used in both G_2 and G_3, while the IC tapes ic is only used in G_2. In this section, we will rename the random permutation tapes in G_3 as $\mathbf{t} = (t_1, t_1^{-1}, \ldots, t_5, t_5^{-1})$, in order to distinguish them from $\mathbf{p} = (p_1, p_1^{-1}, \ldots, p_5, p_5^{-1})$ in G_2.

Similar to previous works, we will characterize an execution with its *footprint*, which is basically the subsets of random tapes that have been read.

Definition 24. A *partial random permutation tape* is a pair of tables $\tilde{p}, \tilde{p}^{-1} : \{0,1\}^n \to \{0,1\}^n \cup \{\perp\}$, such that $\tilde{p}^{-1}(\tilde{p}(x)) = x$ for all x satisfying $\tilde{p}(x) \neq \perp$, and such that $\tilde{p}(\tilde{p}^{-1}(y)) = y$ for all y satisfying $\tilde{p}^{-1}(y) \neq \perp$.

A *partial ideal cipher tape* is a pair of tables $\mathsf{pic}, \mathsf{pic}^{-1} : \{0,1\}^{2n} \to \{0,1\}^n \cup \{\perp\}$, such that $\mathsf{pic}^{-1}(k, \mathsf{pic}(k, u)) = u$ for all k, u such that $\mathsf{pic}(k, u) \neq \perp$, and such that $\mathsf{pic}(k, \mathsf{pic}^{-1}(k, v)) = v$ for all k, v such that $\mathsf{pic}^{-1}(k, v) \neq \perp$.

We will use *partial tape* to refer to either a partial random permutation tape or a partial ideal cipher tape. We note that $\tilde{p}$ determines $\tilde{p}^{-1}$ and vice-versa, therefore we may use either $\tilde{p}$ or $\tilde{p}^{-1}$ to designate the pair $\tilde{p}, \tilde{p}^{-1}$. Similarly for the partial ideal cipher tape $\mathsf{ic}, \mathsf{ic}^{-1}$.

Definition 25. We say a random permutation tape p is *compatible* with a partial random permutation tape $\tilde{p}$ if $p(x) = \tilde{p}(x)$ for all x such that $\tilde{p}(x) \neq \perp$. Similarly, an ideal cipher tape ic is *compatible* with a partial ideal cipher tape pic if $\mathsf{ic}(k, u) = \mathsf{pic}(k, u)$ for all k, u such that $\mathsf{pic}(k, u) \neq \perp$.

Definition 26. Consider an execution of G_2 with random tapes $p_1, p_2, \ldots, p_5, \mathsf{ic}$. The *footprint* of the execution is the set of partial random tapes $\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \mathsf{pic}$ consisting of entries of the corresponding tapes that are accessed[15] at some point during the execution.

[14] Note that y'_5 isn't necessarily equal to y_5; but in a good execution we always have $y_5 = y'_5$.

[15] Recall that $\tilde{p}_i$ is actually a pair $\tilde{p}_i, \tilde{p}_i^{-1}$; an entry $\tilde{p}_i(x_i) = y_i$ can be accessed by reading either $\tilde{p}_i(x_i)$ or $\tilde{p}_i^{-1}(y_i)$. Similarly for pic.

Similarly, the *footprint* of an execution of G_3 with random tapes $t_1, t_2, \ldots, t_5$ is the set of partial random tapes $\tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_5$ consisting of entries of the corresponding tapes that are accessed.

We note that with the fixed deterministic distinguisher $\mathcal{D}$, the footprint of an execution is determined by its random tapes. Among all possible footprints, only a small portion of them are actually obtainable in some execution of $\mathcal{D}$. We let FP_2 and FP_3 denote the set of obtainable footprints in G_2 and G_3 respectively.

We say the random tapes of an execution are *compatible* with a footprint ω if each tape is compatible with the corresponding partial random tape in ω.

Lemma 43. *For $i = 2, 3$ and for an obtainable footprint $\omega \in \mathsf{FP}_i$, an execution of G_i has footprint ω if and only if the random tapes are compatible with ω.*

Proof. The "only if" direction is trivial by the definition of footprints, so we only need to prove the "if" direction.

Let $\mathcal{T}$ denote the set of random tapes of the execution. Since ω is obtainable, there exists a set of random tapes $\mathcal{T}'$ such that the execution of G_i with tapes $\mathcal{T}'$ has footprint ω. By the definition of footprints, only entries in ω are read during the execution with $\mathcal{T}'$. If $\mathcal{T}$ is compatible with ω, the executions with $\mathcal{T}$ and with $\mathcal{T}'$ can never diverge (recall that the distinguisher $\mathcal{D}$ is deterministic by assumption) and thus are identical. In particular, they should have the same footprint ω. □

The above proof shows that an execution can be recovered given its footprint.

We let $\Pr_{\mathsf{G}_i}[\omega]$ denote the probability that the footprint of an execution of G_i is ω. For a set of footprints $\mathcal{S}$, let $\Pr_{\mathsf{G}_i}[\mathcal{S}]$ denote the probability that the footprint of an execution of G_i is in $\mathcal{S}$. Since each execution has exactly one footprint, the events of obtaining different footprints are mutually exclusive and we have

$$\Pr_{\mathsf{G}_i}[\mathcal{S}] = \sum_{\omega \in \mathcal{S}} \Pr_{\mathsf{G}_i}[\omega].$$

For an obtainable footprint $\omega \in \mathsf{FP}_i$, $\Pr_{\mathsf{G}_i}[\omega]$ equals the probability that the random tapes are compatible with ω by Lemma 43. Let $|\tilde{p}_i|$ (resp. $|\mathsf{pic}|$) denote the number of non-$\perp$ entries in $\tilde{p}_i$ (resp. pic), and let $|\mathsf{pic}(k)|$ denote the number of non-$\perp$ entries of the form $(k, *)$ in pic. For $\omega = (\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \mathsf{pic}) \in \mathsf{FP}_2$, we have

$$\Pr_{\mathsf{G}_2}[\omega] = \Big(\prod_{i=1}^{5} \prod_{\ell=0}^{|\tilde{p}_i|-1} \frac{1}{2^n - \ell}\Big)\Big(\prod_{k \in \{0,1\}^n} \prod_{\ell=0}^{|\mathsf{pic}(k)|-1} \frac{1}{2^n - \ell}\Big), \tag{14}$$

and for $\omega = (\tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_5) \in \mathsf{FP}_3$, we have

$$\Pr_{\mathsf{G}_3}[\omega] = \prod_{i=1}^{5} \prod_{\ell=0}^{|\tilde{t}_i|-1} \frac{1}{2^n - \ell}. \tag{15}$$

RANDOMNESS MAPPING. Let $\mathsf{FP}_2^* \subseteq \mathsf{FP}_2$ denote the set of footprints that can be obtained by *good* executions of G_2, and let $\mathsf{FP}_3^* \subseteq \mathsf{FP}_3$ denote the set of footprints that can be obtained by *non-aborting* executions of G_3. Note that an execution can be recovered from the footprint, so an execution of G_2 (resp. G_3) with footprint in FP_2^* (resp. FP_3^*) must be good (resp. non-aborting).

We will define an injective mapping ζ that maps each footprint $\omega \in \mathsf{FP}_2^*$ to $\zeta(\omega) \in \mathsf{FP}_3^*$, such that the executions with footprints ω and $\zeta(\omega)$ are "identical" and such that ω and $\zeta(\omega)$ are obtained with similar probability (in G_2 and G_3 respectively).

Definition 27. (*) We define the injection $\zeta : \mathsf{FP}_2^* \to \mathsf{FP}_3^*$ as follows: for $\omega = (\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \mathsf{pic}) \in \mathsf{FP}_2^*$, let $\zeta(\omega) = (\tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_5)$ where

$$\tilde{t}_i = \{(x, y) \in \{0,1\}^n \times \{0,1\}^n : P_i(x) = y\}$$

and where P_i refers to the state of the table at the end of the execution of G_2 with footprint ω.

The mapping ζ is well-defined because the execution of G_2 can be recovered from its footprint ω and, in particular, the states of the tables at the end of the execution can be recovered from ω. We still need to prove that $\zeta(\omega)$ is in FP_3^* and that ζ is injective.

Lemma 44. (*) *At the end of a good execution of G_2, for each table entry $T(k, x_1) = y_5$ the 2chain $(5, 1, y_5, x_1, k)$ belongs to a complete path which has been triggered during the execution.*

Proof. By Lemma 5 (e), since good executions don't abort, a triggered 2chain must belong to a complete path. Therefore, we only need to prove that for each $T(k, x_1) = y_5$, $(5, 1, y_5, x_1, k)$ is equivalent to a triggered 2chain.

We assume without loss of generality that $T(k, x_1) = y_5$ is defined in a call to $\mathrm{Enc}(k, x_1)$; the proof is symmetric if it is defined in a call to $\mathrm{Dec}(k, y_5)$.

If $\mathrm{Enc}(k, x_1)$ is called by the distinguisher: since the distinguisher completes all paths (cf. Definition 23), the simulator has issued permutation queries (i, x_i, y_i) for $i = 1, 2, 3$ with $x_2 = y_1 \oplus k$ and $x_3 = y_2 \oplus k$. We consider the first query cycle at the end of which the three queries are all table-defined, i.e., the 2chain $C = (1, 2, y_1, x_2, k)$ is table-defined and its right endpoint is table-defined. By Lemma 17 and since query cycles in good executions must be safe (cf. Lemma 21), C belongs to a complete path which was triggered during the query cycle. The lemma follows by the fact that C is equivalent to $(5, 1, y_5, x_1, k)$.

If $\mathrm{Enc}(k, x_1)$ is called by the simulator, note that the simulator only makes cipher queries in FindNewPaths and AdaptPath; we can see from the pseudocode that $(5, 1, y_5, x_1, k)$ is equivalent to a 2chain being triggered. □

The only difference between G_2 and G_3 is in the procedures Enc and Dec: in G_3, the cipher queries are answered by the 5-round IEM construction of the permutations encoded by tapes $\mathbf{t}$, while in G_2 they are answered by the ideal cipher encoded by ic.

The distinguisher and the simulator are identical in the two worlds, so we can say an execution of G_2 is *identical* to an execution of G_3 if the views of the distinguisher and the simulator are identical in the two executions. In particular, we have the following useful observations:

- the distinguisher outputs the same value in identical executions;
- an execution of G_3 is non-aborting if it is identical to a non-aborting execution of G_2.

Lemma 45. (*) *For $\omega \in \mathsf{FP}_2^*$, the footprint $\zeta(\omega)$ is obtainable in G_3. Moreover, the execution of G_2 with footprint $\omega \in \mathsf{FP}_2^*$ is identical to the execution of G_3 with footprint $\zeta(\omega)$.*

Proof. Let $\omega = (\tilde{p}_1, \ldots, \tilde{p}_5, \mathsf{pic})$ and $\zeta(\omega) = (\tilde{t}_1, \ldots, \tilde{t}_5)$. Let $\mathcal{T} = (t_1, \ldots, t_5)$ be an arbitrary set of random permutation tapes compatible with $\zeta(\omega)$ (which can be obtained by arbitrarily expanding the partial tapes in $\zeta(\omega)$). We will prove that the execution of G_3 with $\mathcal{T}$ is identical to the execution of G_2 with footprint ω, and that the execution has footprint $\zeta(\omega)$.

We prove the two executions are identical by running them in parallel and prove by induction that they never diverge. The executions can diverge only when the distinguisher or the simulator accesses the tapes or the cipher oracle. By symmetry, we only consider the forward queries (i.e., when tape entry $p_i(x_i)$ is read or when Enc is called).

A tape entry $p_i(x_i) = y_i$ is read only in the procedure ReadTape. In G_2, the simulator adds a corresponding table entry $P_i(x_i) = y_i$ immediately after reading the tape, which is never overwritten and exists at the end of the execution. By the definition of the mapping, $\zeta(\omega)$ contains the same entry $\tilde{t}(x_i) = y_i$ and, as $\mathcal{T}$ is compatible with $\zeta(\omega)$, $\mathcal{T}$ also contains $t(x_i) = y_i$. Thus the tape entry being read in G_3 is the same as the one in G_2.

If the distinguisher or the simulator calls Enc, the value of $T(k, x_1) = y_5$ is returned. In G_2, the table T is a subset of the cipher encoded by ic and its entries are never overwritten, so we have $T(k, x_1) = y_5$ at the end of the execution. The execution of G_2 with footprint ω is good; by Lemma 44, the 2chain $(5, 1, y_5, x_1, k)$ is complete, i.e., P contains queries (i, x_i, y_i) for $i = 1, 2, 3, 4, 5$ satisfying (4). Similarly to the discussion in the last case, we have $t_i(x_i) = y_i$ for $i = 1, 2, 3, 4, 5$. It is easy to check that these entries are used in a call to $\mathrm{EM}(k, x_1)$ or $\mathrm{EM}^{-1}(k, y_5)$, so in G_3 we also have $T(k, x_1) = y_5$ and the two executions don't diverge on cipher queries. Hence, the two executions never diverge and are identical.

Now we prove the G_3-execution with $\mathcal{T}$ has footprint $\zeta(\omega)$. As discussed before, all tape entries read by the simulator or by the cipher oracle (in EM and EM^{-1}) are in $\zeta(\omega)$. On the other hand, each entry $\tilde{t}(x_i) = y_i$ in $\zeta(\omega)$ corresponds to an entry $P_i(x_i) = y_i$ at the end of the execution (this is true for both executions since they are identical), which is assigned in a call to ReadTape or AdaptPath.

If $P_i(x_i) = y_i$ is assigned in ReadTape, then the simulator should have read $t_i(x_i)$ or $t_i^{-1}(y_i)$ in the same procedure call. If $P_i(x_i) = y_i$ is assigned in AdaptPath, then there exist defined queries (j, x_j, y_j) for $j \in \{1, 2, 3, 4, 5\} \setminus \{i\}$ and a cipher query $T(k, x_1) = y_5$ such that $x_{j+1} = y_j \oplus k$ for $j = 1, 2, 3, 4$. When

$T(k, x_1) = y_5$ is assigned, the entries $t_j(x_j) = y_j$ for $j = 1, 2, 3, 4, 5$ are read in the call to $\mathrm{EM}(k, x_1)$ or $\mathrm{EM}^{-1}(k, y_5)$, which includes the entry $t_i(x_i) = y_i$. In both cases, $t(x_i) = y_i$ is read and the footprint should contain $\tilde{t}(x_i) = y_i$. □

Lemma 46. (*) *The mapping ζ is an injection from FP_2^* to FP_3^*.*

Proof. Since the simulator doesn't abort in good executions of G_2 (cf. Lemma 22), Lemma 45 implies that the execution of G_3 with footprint $\zeta(\omega)$ is non-aborting and hence $\zeta(\omega) \in \mathsf{FP}_3^*$.

Lemma 45 also implies that the ζ is injective: For a fixed $\zeta(\omega)$, the G_2-execution with footprint ω is identical to the G_3-execution with footprint $\zeta(\omega)$, which can be recovered from $\zeta(\omega)$. Let $\omega = (\tilde{p}_1, \ldots, \tilde{p}_5, \mathsf{pic})$, and it can be uniquely reconstructed as follows: the partial permutation tapes $\tilde{p}_i$ contain entries that are read by the simulator during the execution; the partial tape pic contains cipher queries that are issued by the distinguisher or the simulator. □

Lemma 47. (*) *For $\omega = (\tilde{p}_1, \ldots, \tilde{p}_5, \mathsf{pic}) \in \mathsf{FP}_2^*$ and $\zeta(\omega) = (\tilde{t}_1, \ldots, \tilde{t}_5)$, we have*

$$\sum_{i=1}^{5} |\tilde{t}_i| = \sum_{i=1}^{5} |\tilde{p}_i| + |\mathsf{pic}|. \tag{16}$$

Proof. Consider the good G_2-execution with footprint ω: The state of P_i at the end of the execution is the same as the partial tape $\tilde{t}_i$ (cf. Definition 27). On the other hand, $\tilde{p}_i$ contains an entry $\tilde{p}_i(x_i) = y_i$ if and only if $P_i(x_i) = y_i$ is assigned in a call to ReadTape. Thus the difference between the sizes of $\tilde{t}_i$ and $\tilde{p}_i$ equals the number of adapted queries assigned in AdaptPath. From the pseudocode, we observe that AdaptPath is called for each triggered 2chain, in which exactly one query is assigned.

We only need to prove the number of triggered 2chains equals the size of pic. Note that pic contains the same queries as the table T at the end of the execution. By Lemma 44, for each $T(k, x_1) = y_5$, a 2chain equivalent to $(5, 1, y_5, x_1, k)$ has been triggered. On the other hand, each triggered 2chain is completed at the end of the good execution, and by Lemma 29 the triggered 2chains are not equivalent. Thus the triggered 2chains belong to distinct complete paths, each containing a distinct table-defined $(5, 1)$-2chain that corresponds to a distinct query in T. Thus there is a one-one correspondence between triggered 2chains and entries in T, implying $|T| = |\mathsf{pic}|$ equals the number of triggered 2chains. □

Lemma 48. (*) *For $\omega \in \mathsf{FP}_2^*$, we have*

$$\Pr_{\mathsf{G}_3}[\zeta(\omega)] \geq \Pr_{\mathsf{G}_2}[\omega] \cdot (1 - 169q^6/2^n)$$

Proof. The lemma trivially holds if $169q^6 \geq 2^n$. In the following proof we will assume $169q^6 < 2^n$, which implies $2^n - 13q^3 > 0$.

Let $\omega = (\tilde{p}_1, \ldots, \tilde{p}_5, \mathsf{pic})$ and $\zeta(\omega) = (\tilde{t}_1, \ldots, \tilde{t}_5)$. By Lemma 36, we have $|T| \leq 12q^3 + q \leq 13q^3$. Since pic contains the same entries as T at the end of the execution and $\mathsf{pic}(k)$ is a subset of pic, for any key $k \in \{0, 1\}^n$ we have

$$|\mathsf{pic}(k)| \leq |\mathsf{pic}| \leq 13q^3. \tag{17}$$

P_i contains every entry of $\tilde{p}_i$ because they are read in ReadTape, so at the end of the execution we have $|\tilde{p}_i| \le |P_i| = |\tilde{t}_i|$. By (14) and (15), we have

$$\begin{aligned}
\frac{\Pr_{\mathsf{G}_3}[\zeta(\omega)]}{\Pr_{\mathsf{G}_2}[\omega]} &= \left(\prod_{i=1}^{5}\prod_{\ell=|\tilde{p}_i|}^{|\tilde{t}_i|-1}\frac{1}{2^n-\ell}\right)\left(\prod_{k}\prod_{\ell=0}^{|\mathsf{pic}(k)|-1}(2^n-\ell)\right) \\
&\ge \left(\frac{1}{2^n}\right)^{\sum_{i=1}^{5}(|\tilde{t}_i|-|\tilde{p}_i|)}(2^n-13q^3)^{\sum_k|\mathsf{pic}(k)|} \\
&= \left(\frac{2^n-13q^3}{2^n}\right)^{|\mathsf{pic}|} \\
&\ge \left(\frac{2^n-13q^3}{2^n}\right)^{13q^3} \ge 1-\frac{169q^6}{2^n}.
\end{aligned}$$

where the first and the second-to-last inequalities use (17), and where the second equality uses (16) and the fact that $\sum_k |\mathsf{pic}(k)| = |\mathsf{pic}|$ with the sum taken over all possible keys. □

Lemma 49. (*) *If the footprint of a G_3-execution is $\zeta(\omega)$ for some $\omega \in \mathsf{FP}_2^*$, then the view of the distinguisher in this execution is identical to its view in the G_4-execution with the same random tapes. In particular, $\mathcal{D}$ outputs the same value in the two executions.*

Proof. Let $\zeta(\omega) = (\tilde{t}_1, \ldots, \tilde{t}_5)$, and consider a G_3-execution with tapes $\mathcal{T} = (t_1, \ldots, t_5)$ that has footprint $\zeta(\omega)$. By Lemma 45, the G_3-execution with footprint $\zeta(\omega)$ is identical to the G_2-execution with footprint ω; in particular, the tables P_i/P_i^{-1} in the two executions are identical. Thus at the end of the G_3-execution, the table P_i is the same as $\tilde{t}_i$ by Definition 27. The tape t_i is compatible with $\tilde{t}_i$, so if $P_i(x_i) = y_i$ we have $t_i(x_i) = y_i$.

Now we prove that executions of G_3 and G_4 with the same tapes $\mathcal{T}$ are identical to the distinguisher. The cipher oracle Enc and Dec are the same in the two games. Consider a distinguisher call to $\mathrm{Query}(i, +, x_i)$ ($\mathrm{Query}(i, -, y_i)$ is symmetric): In G_3, SimQuery is called which returns the value of $P_i(x_i) = y_i$. The table P_i is never overwritten, so we have $t_i(x_i) = y_i$ as discussed before. In G_4, Query returns $t_i(x_i) = y_i$, which is the same as in G_3. Therefore, the two executions behave identically in the view of the distinguisher. □

Lemma 50. (*) *If the distinguisher $\mathcal{D}$ completes all paths, we have*

$$\Delta_{\mathcal{D}}(\mathsf{G}_2, \mathsf{G}_4) \le 10^{22}q^{38}/2^n + 169q^6/2^n$$

Proof. Let $\mathsf{FP}_2^*(1) \subseteq \mathsf{FP}_2^*$ be the set of footprints of good G_2-executions in which $\mathcal{D}$ outputs 1, and let $\mathsf{FP}_3^*(1) \subseteq \mathsf{FP}_3^*$ be the set of footprints of non-aborting G_3-executions in which $\mathcal{D}$ outputs 1.

By Lemma 45, for $\omega \in \mathsf{FP}_2^*(1)$ we have $\zeta(\omega) \in \mathsf{FP}_3^*(1)$. Moreover, by Lemma 49, if a G_3-execution has footprint $\zeta(\omega) \in \mathsf{FP}_3^*(1)$, $\mathcal{D}$ outputs 1 in the G_4-execution

with the same random tapes. Since ζ is injective, the probability that $\mathcal{D}$ outputs 1 in G_4 is at least

$$\begin{aligned}\sum_{\omega\in\mathsf{FP}_2^*(1)} \Pr_{\mathsf{G}_3}[\zeta(\omega)] &\geq \sum_{\omega\in\mathsf{FP}_2^*(1)} \Pr_{\mathsf{G}_2}[\omega]\cdot\Big(1-\frac{169q^6}{2^n}\Big)\\ &\geq \Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)]-\frac{169q^6}{2^n} \qquad (18)\end{aligned}$$

where the first inequality uses Lemma 48 and where the second inequality is due to $\Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)]\leq 1$.

The probability that $\mathcal{D}$ outputs 1 in G_2 is the sum of two parts: the probability that the execution is good and $\mathcal{D}$ outputs 1, which equals $\Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)]$, and the probability that the execution is not good and $\mathcal{D}$ outputs 1, which is no larger than the probability that the execution is not good. Combined with (18)

$$\begin{aligned}\Delta_{\mathcal{D}}(\mathsf{G}_2,\mathsf{G}_4) &\leq \Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)]+1-\Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*]-\big(\Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)]-\frac{169q^6}{2^n}\big)\\ &= 1-\Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*]+169q^6/2^n\\ &\leq 10^{22}q^{38}/2^n+169q^6/2^n\end{aligned}$$

where the last inequality uses Theorem 39. □

Lemma 51. *For an arbitrary distinguisher $\mathcal{D}$, we have*

$$\Delta_{\mathcal{D}}(\mathsf{G}_2,\mathsf{G}_4)\leq 3\times 10^{33}q^{38}/2^n.$$

Proof. As discussed after Definition 23, an arbitrary distinguisher $\mathcal{D}$ with q queries can be converted to an equivalent distinguisher $\mathcal{D}'$ that completes all paths and that makes at most q extra queries in each position. $\mathcal{D}'$ makes at most $2q$ queries in each position and is subject to Lemma 50, so we have

$$\begin{aligned}\Delta_{\mathcal{D}}(\mathsf{G}_2,\mathsf{G}_4)=\Delta_{\mathcal{D}'}(\mathsf{G}_2,\mathsf{G}_4) &\leq 10^{22}(2q)^{38}/2^n+169(2q)^6/2^n\\ &\leq 3\times 10^{33}q^{38}/2^n.\end{aligned}$$

□

5.7 Indistinguishability of G_1 and G_4

Theorem 52. *Any distinguisher with q queries cannot distinguish G_1 from G_4 with advantage more than $10^{34}q^{38}/2^n$.*

Proof. For any distinguisher $\mathcal{D}$ we have

$$\begin{aligned}\Delta_{\mathcal{D}}(\mathsf{G}_1,\mathsf{G}_4) &= \Delta_{\mathcal{D}}(\mathsf{G}_1,\mathsf{G}_2)+\Delta_{\mathcal{D}}(\mathsf{G}_2,\mathsf{G}_4)\\ &\leq 10^{22}q^{38}/2^n+2704q^8/2^n+3\times 10^{33}q^{38}/2^n\\ &\leq 10^{34}q^{38}/2^n\end{aligned}$$

where the first inequality uses Lemmas 41 and 51. □

Acknowledgments

We thank Dana Dachman-Soled and Jonathan Katz for discussions that led to the termination argument used in this work.

Work of the fourth author was performed under financial assistance award 70NANB15H328 from the U.S. Department of Commerce, National Institute of Standards and Technology.

References

[1] E. Andreeva, A. Bogdanov, Y. Dodis, B. Mennink, and J. P. Steinberger. On the Indifferentiability of Key-Alternating Ciphers. In *Advances in Cryptology - CRYPTO 2013 (Proceedings, Part I)*, volume 8042 of *LNCS*, pages 531–550. Springer, 2013. Full version available at http://eprint.iacr.org/2013/061.

[2] E. Andreeva, A. Bogdanov, and B. Mennink. Towards Understanding the Known-Key Security of Block Ciphers. In *Fast Software Encryption - FSE 2013*, volume 8424 of *LNCS*, pages 348–366. Springer, 2013.

[3] M. Bellare and T. Kohno. A Theoretical Treatment of Related-Key Attacks: RKA-PRPs, RKA-PRFs, and Applications. In *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 491–506. Springer, 2003.

[4] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.

[5] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[6] M. Bellare and P. Rogaway. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, 2006. Full version available at http://eprint.iacr.org/2004/331.

[7] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. *J. Cryptology*, 7(4):229–246, 1994.

[8] A. Biryukov, D. Khovratovich, and I. Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. In *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009.

[9] J. Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In *Fast Software Encryption - FSE '06*, volume 4047 of *LNCS*, pages 328–340. Springer, 2006.

[10] J. Black, P. Rogaway, and T. Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, 2002.

[11] A. Bogdanov, L. R. Knudsen, G. Leander, F.-X. Standaert, J. P. Steinberger, and E. Tischhauser. Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations - (Extended Abstract). In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 45–62. Springer, 2012.

[12] D. Boneh and V. Shoup. A Graduate Course in Applied Cryptography. Available at toc.cryptobook.us.

[13] S. Chen, R. Lampe, J. Lee, Y. Seurin, and J. P. Steinberger. Minimizing the Two-Round Even-Mansour Cipher. In *Advances in Cryptology - CRYPTO 2014 (Proceedings, Part I)*, volume 8616 of *LNCS*, pages 39–56. Springer, 2014. Full version available at http://eprint.iacr.org/2014/443.

[14] S. Chen and J. Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In *Advances in Cryptology - EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 327–350. Springer, 2014. Full version available at http://eprint.iacr.org/2013/222.

[15] B. Cogliati and Y. Seurin. On the Provable Security of the Iterated Even-Mansour Cipher against Related-Key and Chosen-Key Attacks. In *Advances in Cryptology - EUROCRYPT 2015 (Proceedings, Part I)*, volume 9056 of *LNCS*, pages 584–613. Springer, 2015. Full version available at http://eprint.iacr.org/2015/069.

[16] B. Cogliati and Y. Seurin. Strengthening the Known-Key Security Notion for Block Ciphers. In *Fast Software Encryption - FSE 2016*, volume 9783 of *LNCS*, pages 494–513. Springer, 2016.

[17] J. Coron, T. Holenstein, R. Künzler, J. Patarin, Y. Seurin, and S. Tessaro. How to Build an Ideal Cipher: The Indifferentiability of the Feistel Construction. *J. Cryptology*, 29(1):61–114, 2016.

[18] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005.

[19] J.-S. Coron, J. Patarin, and Y. Seurin. The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *LNCS*, pages 1–20. Springer, 2008.

[20] Y. Dai and J. Steinberger. Indifferentiability of 8-Round Feistel Networks. CRYPTO 2016, to appear, 2016. Available at http://eprint.iacr.org/2015/1069.

[21] G. Demay, P. Gazi, M. Hirt, and U. Maurer. Resource-Restricted Indifferentiability. In *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 664–683. Springer, 2013. Full version available at http://eprint.iacr.org/2012/613.

[22] A. Desai. The Security of All-or-Nothing Encryption: Protecting against Exhaustive Key Search. In *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *LNCS*, pages 359–375. Springer, 2000.

[23] Y. Dodis, M. Stam, J. P. Steinberger, and T. Liu. Indifferentiability of Confusion-Diffusion Networks. In *Advances in Cryptology - EUROCRYPT 2016 (Proceedings, Part II)*, volume 9666 of *LNCS*, pages 679–704. Springer, 2016.

[24] O. Dunkelman, N. Keller, and A. Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 336–354. Springer, 2012.

[25] S. Even and Y. Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. *J. Cryptology*, 10(3):151–162, 1997.

[26] P. Farshim and G. Procter. The Related-Key Security of Iterated Even-Mansour Ciphers. In *Fast Software Encryption - FSE 2015*, volume 9054 of *LNCS*, pages 342–363. Springer, 2015. Full version available at http://eprint.iacr.org/2014/953.

[27] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986.

[28] L. Granboulan. Short Signatures in the Random Oracle Model. In *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 364–378. Springer, 2002.

[29] C. Guo and D. Lin. Separating invertible key derivations from non-invertible ones: sequential indifferentiability of 3-round Even-Mansour. *Designs, Codes and Cryptography*, pages 1–21, 2015. Available at http://dx.doi.org/10.1007/s10623-015-0132-0.

[30] C. Guo and D. Lin. Indifferentiability of 3-Round Even-Mansour with Random Oracle Key Derivation. IACR Cryptology ePrint Archive, Report 2016/894, 2016. Available at http://eprint.iacr.org/2016/894.

[31] V. T. Hoang and S. Tessaro. Key-Alternating Ciphers and Key-Length Extension: Exact Bounds and Multi-user Security. In *Advances in Cryptology - CRYPTO 2016 (Proceedings, Part I)*, volume 9814 of *LNCS*, pages 3–32. Springer, 2016.

[32] T. Holenstein, R. Künzler, and S. Tessaro. The Equivalence of the Random Oracle Model and the Ideal Cipher Model, Revisited. In *Symposium on Theory of Computing - STOC 2011*, pages 89–98. ACM, 2011. Full version available at http://arxiv.org/abs/1011.1264.

[33] T. Iwata and T. Kohno. New Security Proofs for the 3GPP Confidentiality and Integrity Algorithms. In *Fast Software Encryption - FSE 2004*, volume 3017 of *LNCS*, pages 427–445. Springer, 2004.

[34] J. Kilian and P. Rogaway. How to Protect DES Against Exhaustive Key Search. In *Advances in Cryptology - CRYPTO '96*, volume 1109 of *LNCS*, pages 252–267. Springer, 1996.

[35] L. R. Knudsen and V. Rijmen. Known-Key Distinguishers for Some Block Ciphers. In *Advances in Cryptology - ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 315–324. Springer, 2007.

[36] X. Lai and J. L. Massey. Hash Function Based on Block Ciphers. In *Advances in Cryptology - EUROCRYPT '92*, volume 658 of *LNCS*, pages 55–70. Springer, 1992.

[37] R. Lampe, J. Patarin, and Y. Seurin. An Asymptotically Tight Security Analysis of the Iterated Even-Mansour Cipher. In *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 278–295. Springer, 2012.

[38] R. Lampe and Y. Seurin. How to Construct an Ideal Cipher from a Small Set of Public Permutations. In *Advances in Cryptology - ASIACRYPT 2013 (Proceedings, Part I)*, volume 8269 of *LNCS*, pages 444–463. Springer, 2013. Full version available at http://eprint.iacr.org/2013/255.

[39] A. Mandal, J. Patarin, and Y. Seurin. On the Public Indifferentiability and Correlation Intractability of the 6-Round Feistel Construction. In *Theory of Cryptography Conference - TCC 2012*, volume 7194 of *LNCS*, pages 285–302. Springer, 2012. Full version available at http://eprint.iacr.org/2011/496.

[40] U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *Theory of Cryptography Conference- TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, 2004.

[41] R. C. Merkle. One Way Hash Functions and DES. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *LNCS*, pages 428–446. Springer, 1989.

[42] B. Preneel, R. Govaerts, and J. Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *Advances in Cryptology - CRYPTO '93*, volume 773 of *LNCS*, pages 368–378. Springer, 1993.

[43] T. Ristenpart, H. Shacham, and T. Shrimpton. Careful with Composition: Limitations of the Indifferentiability Framework. In *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, 2011.

[44] Y. Seurin. *Primitives et protocoles cryptographiques à sécurité prouvée*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, France, 2009.

[45] J. Steinberger. Improved Security Bounds for Key-Alternating Ciphers via Hellinger Distance. IACR Cryptology ePrint Archive, Report 2012/481, 2012. Available at http://eprint.iacr.org/2012/481.

[46] R. S. Winternitz. A Secure One-Way Hash Function Built from DES. In *IEEE Symposium on Security and Privacy*, pages 88–90, 1984.

A The Domain Separation Lemma

In this section we give a simple proof of the "Domain Separation Lemma" by Boneh and Shoup (Lemma 2) using the H-coefficient technique. For the sake of brevity we will include only a very brief sketch of the H-coefficient technique itself; see [14] for further details.

We recall that in order to upper bound the distinguishability of "real" and "ideal" worlds by a (fixed, deterministic) distinguisher D, Patarin's H-coefficient technique consists of dividing the set $\mathcal{T}$ of possible transcripts (sequences of queries and responses) obtainable by D into "good" and "bad" varieties, that we shall denote $\mathcal{T}_1$ and $\mathcal{T}_2$. If X and Y denote the transcript distributions in the real and ideal worlds respectively (these are random variables over $\mathcal{T}$), then one can show that D's distinguishing advantage is upper bounded by

$$\epsilon_{\mathsf{ratio}} + \epsilon_{\mathsf{bad}}$$

where

$$\epsilon_{\mathsf{ratio}} := \max_{\tau \in \mathcal{T}_1} \left(1 - \frac{\Pr[X = \tau]}{\Pr[Y = \tau]}\right)$$

(note this is a maximum taken over all "good" transcripts) and where

$$\epsilon_{\mathsf{bad}} := \Pr[Y \in \mathcal{T}_2]$$

is the probability of obtaining a "bad" transcript in the ideal world. Another key ingredient of the technique—necessary for lower bounding the ratio $\frac{\Pr[X=\tau]}{\Pr[Y=\tau]}$, $\tau \in \mathcal{T}_1$—is the fact that

$$\Pr[X = \tau] = \frac{|\mathsf{RealRT} \downarrow \tau|}{|\mathsf{RealRT}|}$$
$$\Pr[Y = \tau] = \frac{|\mathsf{IdealRT} \downarrow \tau|}{|\mathsf{IdealRT}|}$$

where RealRT, $\mathsf{IdealRT}$ are the sets of all random tapes (or "oracles") in the real and ideal worlds, presuming these random tapes have a uniform distribution in each case, and where "$\mathsf{RealRT} \downarrow \tau$", "$\mathsf{IdealRT} \downarrow \tau$" denote the set of real and ideal random tapes that are compatible with τ in the straightforward, naïve sense. (Not to worry, we shall review these concepts below in the context of our application.) In other words, and as can be conceptually useful,

$$\Pr[X = \tau] = \Pr_{\omega \in \mathsf{RealRT}}[\omega \downarrow \tau]$$
$$\Pr[Y = \tau] = \Pr_{\omega \in \mathsf{IdealRT}}[\omega \downarrow \tau]$$

where the probabilities are taken over a uniform random choice of a random tape $\omega \in \mathsf{RealRT}$, $\mathsf{IdealRT}$ respectively, and where "$\omega \downarrow \tau$" is a shorthand for "$\omega \in (\mathsf{RealRT} \downarrow \tau)$" in the first line and for "$\omega \in (\mathsf{IdealRT} \downarrow \tau)$" in the second line.

In our case we recall that D's oracle consists of a family of permutations $\{\pi_u : u \in U\}$ on $\{0,1\}^n$. In the "ideal world" this family is instantiated by $|U|$ independent random permutations, hence an element

$$\omega = \mathsf{IdealRT}$$

is a $|U|$-tuple of permutations of $\{0,1\}^n$ (and $\mathsf{IdealRT}$ consists exactly of all $(2^n!)^{|U|}$ distinct such $|U|$-tuples); in the "real world" this family is instantiated by a family of $|V|$ independent random permutations $\{\nu_v : v \in V\}$ via

$$\pi_u := \nu_{f(u)}$$

where $f : U \to V$ is some arbitrary function, hence an element

$$\omega \in \mathsf{RealRT}$$

is a $|V|$-tuple of permutations of $\{0,1\}^n$ (and, moreover, RealRT consists of all $(2^n!)^{|V|}$ distinct such $|V|$-tuples). Refining our comments above, we say that a transcript τ is *compatible* with a random tape $\omega \in \mathsf{RealRT} \cup \mathsf{IdealRT}$ (i.e., that $\omega \downarrow \tau$) if and only if every query-response pair present in τ matches the relevant entry in the relevant permutation specified by ω.

We also recall that in the terminology of Lemma 2 the distinguisher D *causes a collision* if and only if its transcript contains two distinct queries (u, x, y), (u', x', y') such that $f(u) = f(u')$ and such that $(x = x' \vee y = y')$. We will define the set of bad transcripts $\mathcal{T}_2$ to be those that contain a collision. Hence

$$\epsilon_{\mathsf{bad}} := \Pr[Y \in \mathcal{T}_2]$$

is precisely the probability that the distinguisher causes a collision in the ideal world.

On the other hand, if a transcript τ contains no collision, then we claim that

$$\Pr[X = \tau] \geq \Pr[Y = \tau]$$

(i.e., the transcript is at least as likely in the real as in the ideal world). To see this it suffices to argue that

$$\Pr_{\omega \in \mathsf{RealRT}}[\omega \downarrow \tau] \geq \Pr_{\omega \in \mathsf{IdealRT}}[\omega \downarrow \tau].$$

Indeed, if τ contains m_u queries under "key" $u \in U$, and if $\ell_v = \sum_{u:f(u)=v} m_u$, then

$$\Pr_{\omega \in \mathsf{RealRT}}[\omega \downarrow \tau] = \prod_{v \in V} \frac{(2^n - \ell_v)!}{2^n!}$$

whereas

$$\Pr_{\omega \in \mathsf{IdealRT}}[\omega \downarrow \tau] = \prod_{u \in U} \frac{(2^n - m_u)!}{2^n!}$$

and it is easy to check that the latter product is less than or equal to the former. Hence

$$\frac{\Pr[X = \tau]}{\Pr[Y = \tau]} \geq 1$$

for all $\tau \in \mathcal{T}_1$, so

$$\epsilon_{\mathsf{ratio}} = 0$$

and D's distinguishing advantage is upper bounded by

$$\epsilon_{\mathsf{ratio}} + \epsilon_{\mathsf{bad}} = \epsilon_{\mathsf{bad}}$$

which is precisely D's probability of causing a collision in the ideal world, as stated by the Domain Separation Lemma.