Indifferentiability of Iterated Even-Mansour Ciphers with Non-Idealized Key-Schedules: Five Rounds are Necessary and Sufficient

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

June 9, 2017

Abstract. We prove that the 5-round iterated Even-Mansour (IEM) construction 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. Moreover, the security bound we achieve is comparable to (in fact, slightly better than) the previously established 12-round bound.

Keywords: key-alternating cipher · iterated Even-Mansour construction · indifferentiability

* 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, USA. E-mail: `aish@cs.umd.edu`

1 Introduction

Background. A large number of block ciphers are so-called *key-alternating ciphers.* Such block ciphers alternatively apply two types of transformations to the current state: the addition (usually bitwise) of a secret key and the application of a public permutation. In more detail, an r-round key-alternating cipher with message space $\{0,1\}^n$ is a transformation of the form

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

where $(k_0, \ldots, k_r)$ are n-bit round keys (usually derived from a master key k of size close to n), where $P_1, \ldots, P_r$ are fixed, key-independent permutations and where x and y are the plaintext and ciphertext, respectively. In particular, virtually all[1] SPNs (*Substitution-Permutation Networks*) have this form, including, e.g., the AES family.

A recent trend has been to analyze this class of block ciphers in the so-called Random Permutation Model (RPM), which models the 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" permutations endowed with short descriptions. 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, 13, 14, 31, 37, 45], 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 [24, 25]) as the number of rounds increases [14, 31].

The Ideal Cipher Model. Although pseudorandomness has been the primary security requirement for a block cipher, in some cases this property is not enough to establish the security of higher-level cryptosystems using the block cipher. 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* [3, 7]. Problems also arise in the context of block-cipher based hash functions [36, 42] where the adversary can control both the message and the key of the block cipher,

[1] Some SPNs do not adhere to the key-alternating abstraction because they introduce the key at the permutation stage as well—e.g., by using keyed S-boxes.

and hence can exploit "known-key" or "chosen-key" attacks [8, 35] in order to break the collision- or preimage-resistance of the hash function.

Hence, 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 [4, 6, 10, 22, 28, 34, 41, 46], an approach often called the *ideal cipher model (ICM)*. 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, similarly to the random oracle model [5, 9, 27]. On the other hand, a proof in the ideal cipher model is typically considered a good indication of security from the point of view of practice.

Indifferentiability. While an IC remains unachievable in the standard model for reasons stated above (and which boil down to basic considerations on the amount of entropy in the system), it remains an interesting problem to "build" ICs (secure in some provable sense) from *other* ideal primitives. This is precisely the approach taken by the indifferentiability framework, introduced by Maurer et al. [40] and popularized by Coron et al. [17]. Indifferentiability is a simulation-based framework that helps assess whether a construction of a target primitive A (e.g., a block cipher) from a 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 IC). Indifferentiability comes equipped with a composition theorem [40] which implies that a large class of protocols (see [21, 43] for restrictions) are provably secure in the ideal-B model if and only if they are provably secure in the ideal-A model.

We note that indifferentiability does not presuppose the presence of a private key; indeed, a number of indifferentiability proofs concern the construction of a keyless primitive (such as a hash function, compression function or permutation) from a lower-level primitive [1, 17, 32]. In the case of a block cipher, thus, the key is "just another input" to the construction.

Previous Results. Two papers have previously explored the indifferentiability of the IEM construction from an ideal cipher, modeling the underlying permutations as random permutations. Andreeva et al. [1] 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 IC. 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 IC. 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 context. In both settings, the question of the exact number of rounds needed to make the IEM construction indifferentiable from an ideal cipher remained open.

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 IC, in both cases for the trivial key-schedule.[2] 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 [1, 38]. It can be seen as an extension of the attack against the 3-round IEM with the trivial key-schedule [38]. However, unlike this 3-round attack, our 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.

Our 5-round feasibility result can be seen as improving both the 5-round result for the IEM construction with idealized key-schedules [1] (albeit see the fine-grained metrics below) and on the 12-round feasibility result for the IEM construction with the trivial key-schedule [38]. Our simulator runs in time $O(q^5)$, makes $O(q^5)$ IC queries, and achieves security $2^{41} \cdot q^{12}/2^n$, where q is the number of distinguisher queries. By comparison, these metrics are respectively

$$O(q^3),\ O(q^2),\ 2^{34} \cdot q^{10}/2^n$$

for the 5-round simulator of Andreeva et al. [1] with idealized key-schedule, and

$$O(q^4),\ O(q^4),\ 2^{91} \cdot q^{12}/2^n$$

for the 12-round simulator of Lampe and Seurin [38]. Hence, as far as the security bound is concerned at least, we achieve a slight improvement over the previous (most directly comparable) work.

A Glimpse at the Simulator. Our 5-round simulator follows the traditional "chain detection/completion" paradigm, pioneered by Coron et al. [18, 19, 32] for proving indifferentiability of the Feistel construction, which has since 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 for by a more complex "termination" proof). In a nutshell, our

[2] 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.

new 5-round simulator detects and completes *any* path of length 3, 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 [18, 19, 32, 44]) which allowed a much simpler termination argument.

Once a tentative simulator has been determined, the indifferentiability proof usually entails two technical challenges: on the one hand, proving that the simulator works hard enough to ensure that it will never be trapped in an inconsistency, and on the other hand, proving that it does not work in more than polynomial time. 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 3, 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 3 can appear “unexpectedly” and remain unnoticed by the simulator (which will therefore not complete it), except with negligible probability.

The more innovative part of our proof lies in the “termination argument”, i.e., in 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 [18, 19, 23, 32, 44] the proof starts by observing that certain types of paths (namely those that wrap around the IC) are only ever detected and completed if the distinguisher made the corresponding IC query. Hence, assuming the distinguisher makes at most q queries, at most q such paths will be triggered and completed. In virtually all previous indifferentiability proofs, this fact easily allows to upper bound the size of permutation histories 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. (Indeed, all of the indifferentiability results in the afore-mentioned list actually have quite simple termination arguments!) But in the case of our 5-round simulator, this observation only allows us 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. To push the argument further, we make some additional observations—essentially, we note that every triggered path that is not a “wraparound” path associated to some distinguisher query is uniquely (i.e., injectively) associated to one of: (i) a pair of P_3 and P_1 entries, where the P_1 entry was directly queried by the distinguisher, or (ii) symmetrically, a pair of P_3 and P_5 entries, where the P_5 entry was directly queried by the distinguisher, or (iii) a *pair* of P_3 entries. (In some sense, the crucial “trick” that allows to fall back on (iii) in all other cases is the observation that every query that is left over from a previous query cycle and that is not the direct result of a distinguisher query is in a completed path, and this completed path contains a query at P_3.) This suffices, because the distinguisher makes only q queries and because of the afore-mentioned bound on the size of P_3. In order to show that the association described above is truly injective, a structural property of P_2 and P_4 is needed, namely that the table maintaining answers of the simulator for P_2 (resp. P_4) never contains 4 distinct input/output pairs $(x^{(i)}, y^{(i)})$, such that $\bigoplus_{1 \le i \le 4}(x^{(i)} \oplus y^{(i)}) = 0$. Since some queries are "adapted" to fit the IC, proving this part ends up being a source of some tedium as well.

Related Work. 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 [15, 26], 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 IC [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 poor) can be improved.

Organization. Preliminary definitions are given in Section 2. The attack against the 4-round IEM construction is given 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 $\mathcal{D}^{\mathcal{O}_1, \mathcal{O}_2, \ldots} = b$ 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 IEM Construction. Fix integers $n, r \ge 1$. 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.

Note that the first and last key additions do not play any role for indifferentiability where 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. So, we will focus on a slight variant of the trivial key-schedule where $f_0 = f_r = 0$ (see Fig. 1), but our results carry over to the trivial key-schedule (and more generally to any non-idealized key-schedule where the f_i's are permutations on $\{0,1\}^n$).

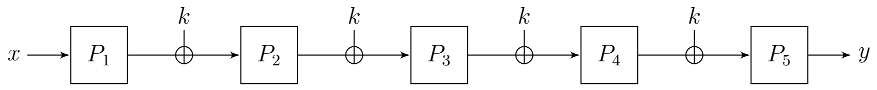

Fig. 1. The 5-round iterated Even-Mansour construction with independent permutations and identical round keys. The first and last round key additions are omitted since they do not play any role for the indifferentiability property.

Indifferentiability. We recall the standard definition of indifferentiability for the IEM construction.

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 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. Definition 1 allows the simulator $\mathcal{S}$ to depend on the number of queries q. In fact, our simulator (cf. Figs. 4 and 5) does not depend on q, but is efficient only with high probability. In Theorem 47 in Section 5.5, we discuss an optimized implementation of our simulator that, among others, uses knowledge of q to abort whenever its runtime exceeds the limit of a "good" execution, thus ensuring that it is efficient with probability 1.

3 Attack Against 4-Round Simulators

We describe an attack against the 4-round IEM construction, improving previous attacks against 3 rounds [1, 38]. Consider the distinguisher $\mathcal{D}$ whose pseudocode is given in Fig. 2 (see also Fig. 3 for an illustration of the attack). This distinguisher can query the permutations/simulator through the interface Query(i, δ, z), and the EM construction/ideal cipher through interfaces Enc(k, x) and Dec(k, y).

```
$y_3 \leftarrow_\$ \{0,1\}^n$
$x_4 \leftarrow_\$ \{0,1\}^n$
$x'_4 \leftarrow_\$ \{0,1\}^n \setminus \{x_4\}$
$k := y_3 \oplus x_4$
$k' := y_3 \oplus x'_4$
$y_4 := \text{Query}(4, +, x_4)$
$y'_4 := \text{Query}(4, +, x'_4)$
$x_1 := \text{Dec}(k, y_4)$
$x'_1 := \text{Dec}(k', y'_4)$
if $x_1 = x'_1$ then
  return $0$
$y_1 := \text{Query}(1, +, x_1)$
$y'_1 := \text{Query}(1, +, x'_1)$
$x_2 := y_1 \oplus k$
$x'_2 := y'_1 \oplus k'$
$k'' := y_1 \oplus x'_2$
$k''' := k'' \oplus k \oplus k'$
$y''_4 := \text{Enc}(k'', x_1)$
$y'''_4 := \text{Enc}(k''', x'_1)$
if $y_4, y'_4, y''_4, y'''_4$ are not distinct then
  return $0$
draw $b \leftarrow_\$ \{0,1\}$
if $b = 1$ then
  $y''_4 \leftarrow_\$ \{0,1\}^n \setminus \{y_4, y'_4\}$
  $y'''_4 \leftarrow_\$ \{0,1\}^n \setminus \{y_4, y'_4, y''_4\}$
$x''_4 := \text{Query}(4, -, y''_4)$
$x'''_4 := \text{Query}(4, -, y'''_4)$
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. 2. Pseudocode of the attack against the 4-round IEM construction.

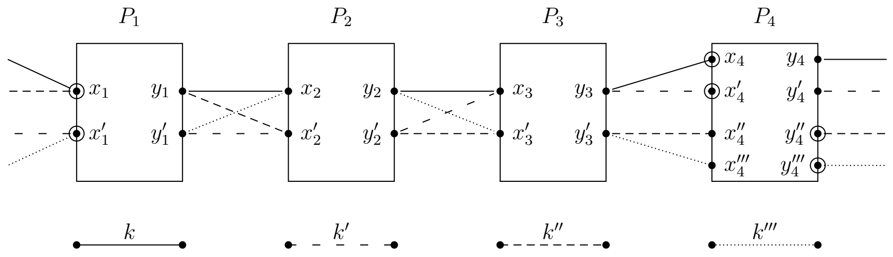

Fig. 3. Illustration of the attack against the 4-round IEM construction. The circled dots correspond to queries made by the distinguisher to the permutations/simulator.

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* $(\text{EM}^{\mathbf{P}}, \mathbf{P})$ *and* $(\text{IC}, \mathcal{S}^{\text{IC}})$

is at least

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

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 $x_3' := y_2' \oplus k'' = y_2 \oplus k'''$ 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 $\{0,1\}^n \backslash \{y_4, y'_4\}$, and y''_4 is sampled uniformly at random from $\{0,1\}^n \backslash \{y_4, y'_4, y''_4\}$. 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

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

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

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

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

We will upper bound $\varDelta_{\mathcal{S}}(\mathcal{D}_0, \mathcal{D}_1)$ 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

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

where

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

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

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

by the Domain Separation Lemma. One also shows that

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

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

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

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* [15, 39] 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

We describe our simulator for proving indifferentiability of the 5-round IEM construction from an ideal cipher.

We start with a high-level overview of how the simulator $\mathcal{S}$ works, deferring the formal description in pseudocode to Section 4.1. 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 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 write $x \in P_i$ (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 $\mathrm{Query}(i, \delta, z)$ allowing the distinguisher to request the value $P_i(z)$ when $\delta = +$ or $P_i^{-1}(z)$ when $\delta = -$ for

$z \in \{0,1\}^n$. 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 the query (i, δ, z) as "pending" and starts a "chain detection/completion" mechanism, called a *permutation 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 query of the distinguisher 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 existing answer), 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 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 *right endpoint* and *left endpoint* of the 2chain, respectively.[4]

The "pending" permutation query (i, δ, z) asked by the distinguisher might create new incomplete paths of length 3 (once answered by the simulator) 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 an 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 these 2chains 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.

To ease the discussion, let us slightly change the notation and assume that the query that initiates the query cycle is either a forward query $(i + 2, +, x_{i+2})$

[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.

[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 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

or an inverse query $(i-1, -, y_{i-1})$. In both cases, adjacent 2chains that might be triggered are of type $(i, i+1)$. For each such 2chain, the simulator computes the endpoint *opposite* the initial query, and marks it "pending" as well. Thus if the initiating query was $(i+2, +, x_{i+2})$, new pending queries of the form $(i-1, -, \cdot)$ are (possibly) created, while if the initiating query was $(i-1, -, y_{i-1})$, new pending queries of the form $(i+2, +, \cdot)$ are (possibly) created. For each of these new pending queries, the simulator recursively detects whether they form a length-3 path with other $(i, i+1)$-2chains, marks these 2chains as "triggered", and so on. Hence, if the initiating query of the distinguisher was of the form $(i+2, +, \cdot)$ or $(i-1, -, \cdot)$, all "pending" queries will be of the form $(i+2, +, \cdot)$ or $(i-1, -, \cdot)$, and all triggered 2chains will be of type $(i, i+1)$. For this reason, we say that such a query cycle is of "type $(i, i+1)$". Note that while this recursive process is taking place, the simulator does *not* assign any new values to the partial permutations $P_1, \ldots, P_5$—indeed, each pending query remains defined only "at one end" during this phase.

Once all 2chains that must eventually be completed have been detected as described above, the simulator starts the completion process. First, it randomly samples the missing endpoints of all "pending" queries. (Thus, a pending query of the form $(i+2, +, x_{i+2})$ will see a value of y_{i+2} sampled; a pending query of the form $(i-1, -, y_{i-1})$ will see a value of x_{i-1} sampled. The fact that each pending query really *does* have a missing endpoint to be sampled is argued in the proof.) Secondly, for each triggered 2chain, the simulator adapts the corresponding path by computing the corresponding input x_{i+3} and output y_{i+3} at position $i+3$ and "forcing" $P_{i+3}(x_{i+3}) = y_{i+3}$. If an overwrite attempt occurs when trying to assign a value for some permutation, the simulator aborts. This completes the high-level description of the simulator's behavior. The important characteristics of an $(i, i+1)$-query cycle are summarized in Table 1.

4.1 Pseudocode of the Simulator and Game Transitions

We now give the full pseudocode for the simulator, and by the same occasion describe the intermediate worlds that will be used in the indifferentiability proof. The distinguisher $\mathcal{D}$ has access to the public interface Query(i, δ, 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 Enc(k, x) and 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 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 $\mathrm{Enc}(k,x)$, resp. $\mathrm{Dec}(k,y)$, is simply answered with $\mathsf{ic}(k,x)$, resp. $\mathsf{ic}^{-1}(k,y)$. The randomness used by the simulator for lazily sampling permutations $P_1,\dots,P_5$ when needed is also made explicit in the pseudocode through uniformly random permutations tapes $\mathbf{p} = (p_1, p_1^{-1}, \dots, p_5, p_5^{-1})$ where $p_i : \{0,1\}^n \to \{0,1\}^n$ 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 Query$(\cdot,\cdot,\cdot)$ 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, following an approach of [32], the Check procedure used by the simulator (see Line 30 of Fig. 4) 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 arithmetic on indices in $\{1,2,3,4,5\}$ is automatically wrapped into that range (e.g., $i+1 = 1$ if $i = 5$). 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 $\perp$ to the distinguisher.

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.

```
**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** $\mathrm{Query}(i, \delta, z)$:
  **return** $\mathrm{SimQuery}(i, \delta, z)$ \\ $\mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3$
  **return** $p_i^{\delta}(z)$ \\ $\mathsf{G}_4$

**public procedure** $\mathrm{Enc}(k, x_1)$:
  **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** $T(k, x_1)$

**public procedure** $\mathrm{Dec}(k, y_5)$:
  **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** $\mathrm{EM}(k, x_1)$:
  **for** $i = 1$ **to** $4$ **do**
    $x_{i+1} = p_i(x_i) \oplus k$
  **return** $p_5(x_5)$

**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** $\mathrm{Check}(k, x_1, y_5)$:
  **return** $\mathrm{Enc}(k, x_1) = y_5$ \\ $\mathsf{G}_1$
  **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 $\mathbf{p}$.

```
**private procedure** SimQuery($(i, \delta, z)$):
  **if** $P_i^{\delta}(z) = \perp$ **then**
    Pending $\leftarrow ((i, \delta, z))$, Triggered $\leftarrow \emptyset$
    **forall** $(i, \delta, z)$ **in** Pending **do** FindNewPaths$(i, \delta, z)$
    **forall** $(i, \delta, z)$ **in** Pending **do** ReadTape$(i, \delta, z)$
    **forall** $(i, i+1, y_i, x_{i+1}, k)$ **in** Triggered **do** AdaptPath$(i, i+1, y_i, x_{i+1}, k)$
  **return** $P_i^{\delta}(z)$

**private procedure** FindNewPaths$(i, \delta, z)$:
**case** $(\delta = +)$:
$x_i \leftarrow z$
**forall** $(x_{i-2}, x_{i-1})$ **in** $(P_{i-2}, P_{i-1})$ **do**
  $y_{i-2} \leftarrow P_{i-2}(x_{i-2})$, $y_{i-1} \leftarrow P_{i-1}(x_{i-1})$
  **if** $i = 2$ **then** $k \leftarrow y_{i-1} \oplus x_i$
  **else** $k \leftarrow y_{i-2} \oplus x_{i-1}$
  $C \leftarrow (i-2, i-1, y_{i-2}, x_{i-1}, k)$
  **if** $C \in$ Triggered **then continue**
  **case** $i \in \{1, 2\}$:
    **if** $\neg$Check$(k, x_1, y_5)$ **then**
      **continue**
  **case** $i \in \{3, 4, 5\}$:
    **if** Next$(i-1, y_{i-1}, k) \neq x_i$ **then**
      **continue**
  Triggered $\hookleftarrow C$
  $y_{i-3} \leftarrow$ Prev$(i-2, x_{i-2}, k)$
  **if** $(i-3, -, y_{i-3}) \notin$ Pending **then**
    Pending $\hookleftarrow (i-3, -, y_{i-3})$

**case** $(\delta = -)$:
$y_i \leftarrow z$
**forall** $(x_{i+1}, x_{i+2})$ **in** $(P_{i+1}, P_{i+2})$ **do**
  $y_{i+1} \leftarrow P_{i+1}(x_{i+1})$, $y_{i+2} \leftarrow P_{i+2}(x_{i+2})$
  **if** $i = 4$ **then** $k \leftarrow y_i \oplus x_{i+1}$
  **else** $k \leftarrow y_{i+1} \oplus x_{i+2}$
  $C \leftarrow (i+1, i+2, y_{i+1}, x_{i+2}, k)$
  **if** $C \in$ Triggered **then continue**
  **case** $i \in \{4, 5\}$:
    **if** $\neg$Check$(k, x_1, y_5)$ **then**
      **continue**
  **case** $i \in \{1, 2, 3\}$:
    **if** Prev$(i+1, x_{i+1}, k) \neq y_i$ **then**
      **continue**
  Triggered $\hookleftarrow C$
  $x_{i+3} \leftarrow$ Next$(i+2, y_{i+2}, k)$
  **if** $(i+3, +, x_{i+3}) \notin$ Pending **then**
    Pending $\hookleftarrow (i+3, +, x_{i+3})$

**private procedure** ReadTape$(i, \delta, z)$:
  **if** $\delta = +$ **then** Assign$(i, z, p_i(z))$ **else** Assign$(i, p_i^{-1}(z), z)$

**private procedure** AdaptPath$(i, i+1, y_i, x_{i+1}, k)$:
  $y_{i+1} \leftarrow P_{i+1}(x_{i+1})$, $x_{i+2} \leftarrow$ Next$(i+1, y_{i+1}, k)$, $y_{i+2} \leftarrow P_{i+2}(x_{i+2})$
  $x_{i+3} \leftarrow$ Next$(i+2, y_{i+2}, k)$
  $x_i \leftarrow P_i^{-1}(y_i)$, $y_{i-1} \leftarrow$ Prev$(i, x_i, k)$, $x_{i-1} \leftarrow P_{i-1}^{-1}(y_{i-1})$
  $y_{i-2} \leftarrow$ Prev$(i-1, x_{i-1}, k)$
  Assign$(i+3, x_{i+3}, y_{i-2})$ \\ subscripts are equal because of the wrapping

**private procedure** Assign$(i, x_i, y_i)$:
  **if** $P_i(x_i) \neq \perp$ or $P_i^{-1}(y_i) \neq \perp$ **then abort**
  $P_i(x_i) \leftarrow y_i$, $P_i^{-1}(y_i) \leftarrow x_i$

**private procedure** Next$(i, y_i, k)$:
  **if** $i = 5$ **then return** Dec$(k, y_i)$
  **else return** $y_i \oplus k$

**private procedure** Prev$(i, x_i, k)$:
  **if** $i = 1$ **then return** Enc$(k, x_i)$
  **else return** $x_i \oplus k$
```

Fig. 5. Private procedures used by the simulator.

5 Proof of Indifferentiability

5.1 Main Result and Proof Overview

Our main result is the following theorem which uses the simulator described in Section 4.

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* $\varepsilon = 2 \times 10^{12} q^{12}/2^n$*.*

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).*

The upper bounds on the simulator's running time and query complexity are given in Theorem 47. The upper bound on the distinguisher's distinguishing advantage is given in Theorem 58.

Proof Structure. Our proof uses a sequence of games G_1, G_2, G_3 and G_4 as described in Section 4.1, 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 $\mathcal{D}$ that can make a total of $6q$ queries: at most q cipher queries and at most q queries to $\mathrm{Query}(i, \cdot, \cdot)$ for each $i \in \{1, \ldots, 5\}$, as stipulated in Theorem 3. (Giving the distinguisher q queries at *each* position gives it more power while not significantly affecting the proof or the bounds, and the distinguisher's extra power actually leads to *better* bounds at the final stages of the proof [20].[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

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

Our primary goal is to upper bound $\Delta_{\mathcal{D}}(\mathsf{G}_1, \mathsf{G}_4)$ (in Theorem 58), while the secondary goals of upper bounding the simulator's query complexity and running time will be obtained as corollaries along the way.

Our proof starts with discussions about the game G_2, which is in some sense the "anchor point" of the first two game transitions. As usual, there are *bad*

[5] In the randomness mapping, we will need to convert an arbitrary distinguisher to one that "completes all paths". If the distinguisher is only allowed q arbitrary queries in total, the number of queries will balloon up to $6q$; but if D is given extra power as described here, the reduction only increases q to $2q$.

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 do not occur. The proof of efficiency of the simulator (in good executions of G_2) is the most interesting part of this paper; the technical content is in Section 5.4, and a separate high-level overview of the argument is also included immediately below (see "Termination Argument"). 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 transition to G_4 (see below).

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, as argued in Lemma 45. 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. In particular, this implies that an execution of G_1 is efficient with high probability; the details can be found in Theorem 47, where a modified version of the simulator (namely one that knows the value of q, as discussed after Definition 1) is used.

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 the randomness of 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 55). 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.

Termination Argument. Since the termination argument—i.e., the fact that our simulator does not run amok with excessive path completions, except with negligible probability—is one of the more novel aspects of our proof, we provide a separate high-level overview of this argument here.

To start with, observe that at the moment when an $(i, i+1)$-path is triggered, 3 queries on the path are either already in existence or already scheduled for future existence regardless of this event: the queries at position i and $i+1$ are already defined, while the pending query that triggers the path was already scheduled to become defined even before the path was triggered; hence, each triggered path only "accounts" for 2 new queries, positioned either at $i+2$, $i+3$ or at $i-1$, $i-2$ $(= i+3)$, depending on the position of the pending query.

A second observation is that...

- $(1, 2)$-2chains triggered by pending queries of the form $(5, -, \cdot)$, and
- $(4, 5)$-2chains triggered by pending queries of the form $(1, +, \cdot)$, and
- $(5, 1)$-2chains triggered by either pending queries of the form $(2, +, \cdot)$ or $(4, -, \cdot)$

...all involve a cipher query (equivalently, a call to Check, in G_2) to check the trigger condition, and one can argue that this query must have been made by the distinguisher itself. (Because when the simulator makes a query to Enc/Dec that is not for the purpose of detecting paths, it is for the purpose of completing a path.) Hence, because the distinguisher only has q cipher queries, only q such path completions should occur in total. Moreover, these three types of path completions are exactly those that "account" for a new (previously unscheduled) query to be created at P_3. Hence, and because the only source of new queries are path completions and queries coming directly from the distinguisher, the size of P_3 never grows more than $q + q = 2q$, with high probability.

Of the remaining types of 2chain completions (i.e., those that do not involve the presence of a previously made "wraparound" cipher query), those that contribute a new entry to P_2 are the following:

– $(3, 4)$-2chains triggered by pending queries of the form $(5, +, \cdot)$
– $(4, 5)$-2chains triggered by pending queries of the form $(3, -, \cdot)$

We can observe that either type of chain completion involves values y_3, x_4, y_4, x_5 that are well-defined at the time the chain is detected. We will analyze both types of path completion simultaneously, but dividing into two cases according to whether (a) the distinguisher ever made the query $\text{Query}(5, +, x_5)$, or else received the value x_5 as an answer to a query of the form $\text{Query}(5, -, y_5)$, or (b) the query $P_5(x_5)$ is being defined / is already defined as the result of a path completion. (Crucially, (a) and (b) are the *only* two options for x_5.)

For (a), at most q such values of x_5 can ever exist, since the distinguisher makes at most q queries to $\text{Query}(5, \cdot, \cdot)$; moreover, there are at most $2q$ possibilities for y_3, as already noted; and we have the relation

$$y_3 \oplus x_5 = x_4 \oplus y_4 \tag{4}$$

from the fact that y_3, x_4, y_4 and x_5 lie on a common path. One can show that, with high probability,

$$x_4 \oplus y_4 \neq x'_4 \oplus y'_4$$

for all x_4, y_4, x'_4, y'_4 such that $P_4(x_4) = y_4$, $P_4(x'_4) = y'_4$ and such that $x_4 \neq x'_4$.[6] Hence, with high probability (4) has at most a unique solution x_4, y_4 for each y_3, x_5, and scenario (a) accounts for at most $2q^2$ path completions (one for each possible left-hand side of (4)) of either type above.

For (b), there must exist a separate (table-defined) 2chain $(3, x'_3, y'_3)$, $(4, x'_4, y'_4)$ whose right endpoint is x_5. (This is the case if x_5 is part of a previously completed path, and is also the case if $(5, +, x_5)$ became a pending query during the current query cycle without being the initiating query.) The relation

$$y'_3 \oplus x'_4 \oplus y'_4 = y_3 \oplus x_4 \oplus y_4$$

[6] Probabilistically speaking, this trivially holds if P_4 is a random partial permutation defined at only polynomially many points, though our proof is made more complicated by the fact that P_4 also contains "adapted" queries.

(both sides are equal to x_5) implies

$$y_3 \oplus y'_3 = x_4 \oplus y_4 \oplus x'_4 \oplus y'_4 \tag{5}$$

and, similarly to (a), one can show that (with high probability)

$$x_4 \oplus y_4 \oplus x'_4 \oplus y'_4 \neq X_4 \oplus Y_4 \oplus X'_4 \oplus Y'_4$$

for all table-defined queries $(4, x_4, y_4), \ldots, (4, X'_4, Y'_4)$ with $\{(x_4, y_4), (x'_4, y'_4)\} \neq \{(X_4, Y_4), (X'_4, Y'_4)\}$. Thus, we have (modulo the ordering of (x_4, y_4) and (x'_4, y'_4)[7]) at most one solution to the RHS of (5) for each LHS; hence, scenario (b) accounts for at most $4q^2$ path completions[8] of either type above, with high probability.

Combining these bounds, we find that P_2 never grows to size more than $2q + 2q^2 + 4q^2 = 6q^2 + 2q$ with high probability, where the term of $2q$ accounts for (the sum of) direct distinguisher queries to $\mathrm{Query}(2, \cdot, \cdot)$ and "wraparound" path completions involving a distinguisher cipher query. Symmetrically, one can show that P_4 also has size at most $6q^2 + 2q$, with high probability.

One can now easily conclude the termination argument; e.g., the number of $(2, 3)$- or $(3, 4)$-2chains that trigger path completions is each at most $2q \cdot (6q^2 + 2q)$ (the product of the maximum size of P_3 with the maximum size of P_2/P_4); or, e.g., the number of $(1, 2)$-2chains triggered by a pending query $(3, +, \cdot)$ is at most $2q \cdot (6q^2 + 2q)$ (the product of the maximum size of P_3 with the maximum size of P_2), and so forth.

5.2 Executions of G_2: Definitions and Basic Properties

Here, we define a set of bad events that may occur in G_2. An execution of G_2 is *good* if none of these bad events occur. We will prove that in good executions of G_2, the simulator does not abort and runs in polynomial time.

5.2.1 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.

[7] As argued within the proof, this ordering issue does not actually introduce an extra factor of two into the bounds.

[8] Or more exactly, to at most $2q(2q - 1)$ path completions, which leads to slightly better bounds used in the proof.

For permutation queries, we may omit i and δ when 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 $P_i^{-1}(y_i) = x_i$. In fact, we will 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 $P_i(x_i) = y_i$, $P_i^{-1}(y_i) = x_i$.

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 $T^{-1}(k, y) = x$. Similarly, we will 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 $T(k, x) = y$, $T^{-1}(k, y) = x$.

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 $y_5, x_1, k \in \{0,1\}^n$. An $(i, i+1)$-2chain refers either to an inner or a $(5, 1)$-2chain, and is generically denoted $(i, i+1, y_i, x_{i+1}, k)$. 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 *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 *table-defined* if both $(5, -, y_5)$ and $(1, +, x_1)$ are table-defined permutation queries and if $T(k, x_1) = y_5$, 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

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

We say that an endpoint is *dummy* when it is equal to $\bot$, 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 $(1,2)$ can be dummy.

We sometimes identify the right and left (non-dummy) endpoints $r(C)$, $\ell(C)$ of an $(i, i+1)$-2chain C with the corresponding permutation queries $(i+2, +, r(C))$ and $(i-1, -, \ell(C))$. In particular, if we say that $r(C)$ or $\ell(C)$ is "table-defined" this implicitly means that the endpoint in question is non-dummy and that the corresponding permutation query is table-defined. More importantly—and more subtly!—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 is in exactly one of these three possible states: dummy, table-undefined, 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

$$y_i \oplus x_{i+1} = k \text{ for } i = 1,2,3,4 \text{ and } T(k, x_1) = y_5. \tag{6}$$

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.

A 2chain C is also said to be *complete* if it belongs to some complete path. Note that such a 2chain is table-defined; also, its endpoints $r(C)$, $\ell(C)$ are (non-dummy and) table-defined.

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

Proof. This follows from the fact that, by definition, a 2chain stipulates a value of k, and from the fact that the tables P_i/P_i^{-1} as well as $T(k,\cdot)/T^{-1}(k,\cdot)$ encode partial permutations. $\square$

Query Cycles. When the distinguisher makes a permutations query (i, δ, z) that is already table-defined, the simulator returns the answer immediately. The definition below introduces some vocabulary related to the simulator's behavior when the distinguisher makes a permutation query that 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.

Definition 10 (cipher query cycle)**.** A *cipher query cycle* is the period of execution between when the distinguisher issues a table-undefined cipher query (δ, k, z) and when the answer to this query is returned. We call (δ, k, z) the *initiating query* of the cipher query cycle.

Remark 5. Note that a "query cycle" as defined above is a "permutation query cycle" in the informal description in Section 4, and cipher query cycles are not a special case of query cycles. Both query cycles and cipher query cycles require the initiating query to be table-undefined, since otherwise the answer already exists in the tables and is directly returned.

Definition 11 (pending queries, triggered 2chains)**.** During a query cycle, we say that a permutation query (i, δ, z) is *pending* (or that z is pending when i and δ are clear from the context) if it is appended 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.

We present a few lemmas below that give some basic properties of query cycles that will be used throughout the 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 $(i-1, -)$, $(i+2, +)$ become 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 $\mathsf{Pending}$, 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 belongs. $\square$

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 $\mathrm{Enc}(\cdot,\cdot)$ *resulting from calls to* $\mathrm{Prev}(1,\cdot,\cdot)$ *at Line 56.*
(b) During a $(2,3)$-query cycle, tables T/T^{-1} are only modified during the completion phase by calls to $\mathrm{Enc}(\cdot,\cdot)$ *resulting from calls to* $\mathrm{Prev}(1,\cdot,\cdot)$ *at Line 83.*
(c) During a $(3,4)$-query cycle, tables T/T^{-1} are only modified during the completion phase by calls to $\mathrm{Dec}(\cdot,\cdot)$ *resulting from calls to* $\mathrm{Next}(5,\cdot,\cdot)$ *at Line 81.*
(d) During a $(4,5)$-query cycle, tables T/T^{-1} are only modified during the detection phase by calls to $\mathrm{Dec}(\cdot,\cdot)$ *resulting from calls to* $\mathrm{Next}(5,\cdot,\cdot)$ *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 $\mathrm{Prev}(1,\cdot,\cdot)$, resp. $\mathrm{Next}(5,\cdot,\cdot)$ are also called during the completion phase, but they are always called with arguments (x_1,k), resp. (y_5,k) that were previously used during the detection phase, so that this cannot modify the tables T/T^{-1}. $\square$

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 $r(C) = r(C') \neq \bot$ or $\ell(C) = \ell(C') \neq \bot$. 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, $r(C) = r(C') \neq \bot$. 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 $C = C'$. $\square$

5.2.2 Bad Events

In order to define certain bad events that may happen during an execution of G_2, we introduce the following definitions.

Definition 12 ($\mathcal{H}$, $\mathcal{K}$ and $\mathcal{E}$)**.** Consider a permutation query (i_0, δ_0, z_0) or a cipher query (δ_0, k_0, z_0) made by the distinguisher. The following sets are defined with respect to the state of tables when the query occurs. We define the "history" $\mathcal{H}$ as the multiset consisting of the following elements (each n-bit string may appear and be counted multiple times):

– for each table-defined permutation query (i, x_i, y_i), $\mathcal{H}$ contains corresponding elements x_i, y_i and $x_i \oplus y_i$.
– for each table-defined cipher query (k, x_1, y_5), $\mathcal{H}$ contains corresponding elements k, x_1 and y_5.

We define $\mathcal{K}$ as the multiset of all keys of 2chains *triggered in the current query cycle*, and $\mathcal{E}$ as the multiset of non-dummy endpoints of *all* table-defined 2chain plus the value z_0 (the query issued by the distinguisher).

Remark 6. When referring to sets $\mathcal{H}$, $\mathcal{K}$ and $\mathcal{E}$ with respect to a query cycle, we mean with respect to its initiating permutation query (and the state of tables at the beginning of the query cycle). These sets are time-dependent, but they don't change during a query cycle (in particular, the set of triggered 2chains do not depend on the queries that become table-defined during the query cycle). Also note that $\mathcal{K}$ only concerns 2chains triggered in the query cycle, while $\mathcal{E}$ concerns all 2chains that are table-defined at the beginning of the query cycle.

Definition 13 ($\mathcal{P}$, $\mathcal{P}^*$, $\mathcal{A}$ and $\mathcal{C}$)**.** Given a query cycle, let $\mathcal{P}$ be 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, and $\mathcal{P}^*$ be the multiset of $x_i \oplus p_i(x_i)$ and $y_i \oplus p_i^{-1}(y_i)$ for each random value $p_i(x_i)$ or $p_i^{-1}(y_i)$ read from the tapes in the current query cycle.

Let $\mathcal{A}$ be the multiset of the values of $x_i \oplus y_i$ for each adapted query (i, x_i, y_i) with $i \in \{2, 4\}$. Note that $\mathcal{A}$ is non-empty only for $(4,5)$- and $(1,2)$-query cycles.

Given a query cycle or a cipher query cycle, we denote $\mathcal{C}$ the multiset of random values read by Enc and Dec on tapes ic or ic^{-1}.[9]

We define the operations $\cap$, $\cup$ and $\oplus$ of two multisets $\mathcal{S}_1, \mathcal{S}_2$ in the natural way: For each element e that appears s_1 and s_2 times ($s_1, s_2 \geq 0$) in $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively, $\mathcal{S}_1 \cap \mathcal{S}_2$ contains $\min\{s_1, s_2\}$ copies of e and $\mathcal{S}_1 \cup \mathcal{S}_2$ contains $s_1 + s_2$ copies of e. To define $\mathcal{S}_1 \oplus \mathcal{S}_2$, we start from an empty multiset; for each pair of $e_1 \in \mathcal{S}_1$ and $e_2 \in \mathcal{S}_2$ that appear s_1 and s_2 times respectively ($s_1, s_2 \geq 1$), add $s_1 \cdot s_2$ copies of $e_1 \oplus e_2$ to the multiset.

Definition 14. Let $\mathcal{H}^{\oplus i}$ be the multiset of values equal to the exclusive-or of exactly i *distinct* elements in $\mathcal{H}$, and let $\mathcal{H}^{\oplus 0} := \{0\}$. The multisets $\mathcal{K}^{\oplus i}$, $\mathcal{E}^{\oplus i}$, $\mathcal{P}^{\oplus i}$, $\mathcal{P}^{*\oplus i}$, $\mathcal{A}^{\oplus i}$ and $\mathcal{C}^{\oplus i}$ are defined similarly.[10]

We are now ready to define the afore-mentioned "bad events" on executions of G_2.

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

[10] Since $\mathcal{H}$, $\mathcal{K}$, $\mathcal{E}$, $\mathcal{P}$, $\mathcal{P}^*$, $\mathcal{A}$ and $\mathcal{C}$ are multisets, two distinct elements may be equal. Because of the distinctness requirement, we have $\mathcal{H}^{\oplus 2} \neq \mathcal{H} \oplus \mathcal{H}$, etc.

Definition 15. $\mathsf{BadPerm}$ is the event that at least one of the following occurs in a query cycle:

– $\mathcal{P}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$ for $i \geq 1$ and $i + j \leq 4$;
– $\mathcal{P}^{*\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$ for $i \geq 1$ and $i + j \leq 4$;
– $\mathcal{P} \cap \mathcal{E} \neq \emptyset$, $\mathcal{P} \cap (\mathcal{E} \oplus \mathcal{K}) \neq \emptyset$, $\mathcal{P} \cap (\mathcal{K} \oplus \mathcal{H}) \neq \emptyset$, $\mathcal{P} \cap (\mathcal{K} \oplus \mathcal{H}^{\oplus 2}) \neq \emptyset$;
– $\mathcal{P}^{\oplus 2} \cap \mathcal{K}^{\oplus 2} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap (\mathcal{H} \oplus \mathcal{K}) \neq \emptyset$;
– $\mathcal{P}^{*} \cap (\mathcal{H} \oplus \mathcal{E}) \neq \emptyset$.

Definition 16. $\mathsf{BadAdapt}$ is the event that in a $(1, 2)$- or $(4, 5)$-query cycle, $\mathcal{A}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$ for $i \geq 1$ and $i + j \leq 4$.

Definition 17. BadIC is the event that in a query cycle or in a cipher query cycle, either $\mathcal{C} \cap (\mathcal{H} \cup \mathcal{E}) \neq \emptyset$ or $\mathcal{C}$ contains two equal entries.

Note that $\mathcal{P}^{\oplus i}$, $\mathcal{P}^{*\oplus i}$, $\mathcal{A}^{\oplus i}$ 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 $\mathsf{ic}/\mathsf{ic}^{-1}$ during the query cycle, while $\mathcal{H}^{\oplus i}$, $\mathcal{K}^{\oplus i}$ and $\mathcal{E}^{\oplus i}$ are fixed and determined by the states of the tables at the beginning of the query cycle.

Definition 18 (Good Executions)**.** An execution of G_2 is said to be *good* if none of $\mathsf{BadPerm}$, $\mathsf{BadAdapt}$ and 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 19. 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[11] 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 need to formalize this further).

Definition 20. A (cipher) query cycle is said to be *good* if no bad event ($\mathsf{BadPerm}$, $\mathsf{BadAdapt}$ or BadIC) occurs in the (cipher) query cycle.

In a good execution of G_2, all query cycles and cipher query cycles are good since no bad event occurs throughout the execution.

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

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

phase can be split into two sub-phases: first, the simulator calls $\text{ReadTape}(i, \delta, z)$ for each pending query (i, δ, z), and then it calls $\text{AdaptPath}(C)$ for each triggered 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 set of endpoints $\mathcal{E}$, any endpoint of C which was non-dummy at the beginning of the query cycle is in $\mathcal{E}$. 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 $\text{FindNewPaths}(3, -, \cdot)$ or to $\text{FindNewPaths}(1, +, \cdot)$ in a $(4,5)$-query cycle. We first consider the case where it is triggered during a call to $\text{FindNewPaths}(3, -, \cdot)$. Inspection of the pseudocode then shows that right after C has been triggered, a call to $\text{Dec}(k, y_5)$ resulting from a call to $\text{Next}(5, y_5, k)$ 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 $\text{FindNewPaths}(1, +, \cdot)$. Note that during a call to $\text{FindNewPaths}(1, +, x_1)$, the simulator triggers a table-defined 2chain $C = (4, 5, y_4, x_5, k)$ only if $\text{Check}(k, x_1, y_5)$, 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 $\text{Dec}(\cdot, \cdot)$ resulting from calls to $\text{Next}(5, \cdot, \cdot)$ 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. $\square$

Lemma 9. *Consider a good and safe query cycle in an execution of G_2. The simulator does not abort during the calls to* ReadTape *in 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 $\text{ReadTape}(i+2, +, x_{i+2})$ (the case of a call to $\text{ReadTape}(i-1, -, y_{i-1})$ 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 occurs, contradicting the assumption that the query cycle 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 $\mathsf{Pending}$ in a given query cycle due to the checks at lines 57 and 75. Assume now that $p_{i+2}(x_{i+2}) \in P^{-1}_{i+2}$ when the call to ReadTape occurs. If $p_{i+2}(x_{i+2}) \in P^{-1}_{i+2}$ at the beginning of the query cycle, then $p_{i+2}(x_{i+2}) \in \mathcal{P} \cap \mathcal{H}$ and $\mathsf{BadPerm}$ 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 $\mathsf{Pending}$ 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. $\square$

Now that we have proved that during a good and safe query cycle, the simulator does not abort during calls to ReadTape, we know 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 good and safe $(i, i+1)$-query cycle in an 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.*[12] *Consider the resulting call to* Assign$(i+3,$ $x_{i+3}, y_{i+3})$[13] *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 $k \in \mathcal{K}$;*
- *if $i = 3$, $x_{i+3} = x_1 = \mathsf{ic}^{-1}(k, p_5(r(C))) \in \mathcal{C}$;*
- *if $i \neq 2$, $y_{i+3} = p^{-1}_{i-1}(\ell(C)) \oplus k$ where $p^{-1}_{i-1}(\ell(C)) \in \mathcal{P}$ and $k \in \mathcal{K}$;*

[12] 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.

[13] 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 triggered, hence by definition $k \in \mathcal{K}$.

Now consider 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 ReadTape$(5, +, x_5)$, where $x_5 = P_4(x_4) \oplus k$. Hence, let $y_5 = p_5(x_5) \in \mathcal{P}$. 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). If the cipher query $(-, k, y_5)$ is table-defined at the beginning of the query cycle, by definition we have $y_5 \in \mathcal{H}$, so that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs, contradicting the assumption that the query cycle is good. Thus $(-, k, y_5)$ is table-undefined at the beginning of the query cycle and is table-defined after the call to Dec(k, y_5). By Lemma 6, only Dec is called in the $(3,4)$-query cycle. Therefore, the cipher query must be defined in a call to Dec and hence we have $x_1 = \mathsf{ic}^{-1}(k, y_5) \in \mathcal{C}$. □

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

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

Proof. Assume towards a contradiction that the simulator aborts in a call to AdaptPath(C) during a good and safe $(i, i+1)$-query cycle where $C = (i, i+1, y_i, x_{i+1}, k)$. 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 $y_{i+3} \in P_{i+3}^{-1}$. By symmetry, we only consider the case where $x_{i+3} \in P_{i+3}$ when Assign$(i+3, x_{i+3}, y_{i+3})$ is called. We distinguish the cases $i \neq 3$ 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 $k \in \mathcal{K}$. 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 $\mathcal{P} \cap (\mathcal{H} \oplus \mathcal{K}) \neq \emptyset$, which means $\mathsf{BadPerm}$ 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

$$p_{i+2}(x_{i+2}) \oplus p_{i+2}(x'_{i+2}) = k \oplus k'. \tag{7}$$

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

$$k = p_{i+2}(x_{i+2}) \oplus x_{i+3} = p_{i+2}(x'_{i+2}) \oplus x'_{i+3} = k'$$

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. (7) implies that $\mathcal{P}^{\oplus 2} \cap \mathcal{K}^{\oplus 2} \neq \emptyset$, i.e., $\mathsf{BadPerm}$ 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

$$\mathsf{ic}^{-1}(k, y_5) = \mathsf{ic}^{-1}(k', y'_5). \tag{8}$$

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. (8) implies that $\mathcal{C}$ contains two equal entries, so BadIC occurs.

This concludes the proof. □

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

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

Since all query cycles in a good execution are good, it remains now to show that they are all 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 cipher query cycle (a direct consequence of Lemma 13) and by any query cycle (Lemmas 15, 16 and 17), and deduce that this holds at the beginning of any query cycle (Lemma 19). We also show that a 2chain which is complete at the beginning of a query cycle cannot be triggered (Lemma 21). 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 good cipher query cycle in an execution of G_2. Let C be a 2chain. Then the following two properties hold:*

(a) If C is table-undefined before the cipher query cycle, then C is still table-undefined at the end of the cipher query cycle.
(b) If C is table-defined and one of its endpoints is dummy before the cipher query cycle, then this endpoint is dummy or table-undefined at the end of the cipher query cycle.

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 there exists a $(5, 1)$-2chain $C = (5, 1, y_5, x_1, k)$ which is table-undefined before the cipher query cycle and is table-defined at the end of the cipher query cycle. Since tables P_1/P_1^{-1} and P_5/P_5^{-1} are not modified in a cipher query cycle, the permutation queries $(1, +, x_1)$ and $(5, -, y_5)$ must be table-defined at the beginning of the cipher query cycle, implying $x_1, y_5 \in \mathcal{H}$. Moreover, since C is table-undefined before the cipher query cycle, the cipher query (k, x_1, y_5) must be defined during the cipher query cycle, so either $x_1 \in \mathcal{C}$ or $y_5 \in \mathcal{C}$; in both cases $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC occurs, contradicting the assumption that the query cycle is good.

We then prove (b). By symmetry, we only give the proof for the case of the right endpoint of a $(4, 5)$-2chain. Assume towards contradiction that there exists a table-defined 2chain $C = (4, 5, y_4, x_5, k)$ such that the right endpoint of C is dummy before the cipher query cycle and is $x_1 \in P_1$ at the end of the query cycle. Since no permutation query is defined during the cipher query cycle, both $(5, x_5, y_5)$ and $(1, x_1, y_1)$ are table-defined at the beginning of the cipher query cycle and hence $y_5, x_1 \in \mathcal{H}$. The cipher query (k, x_1, y_5) must be defined during the cipher query cycle, so we have $x_1 \in \mathcal{C}$ or $y_5 \in \mathcal{C}$; in both cases $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC occurs. $\square$

Lemmas 15, 16, and 17 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. Note also that the three lemmas assume a good and 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.

We say that a complete path has been triggered if any of the five 2chains belonging to the complete path has been triggered.[14] By Lemma 5 (e), each triggered 2chain belongs to a triggered complete path at the end of the query cycle if the simulator doesn't abort. We start with the following useful observation.

Lemma 14. *If a cipher query (k, x_1, y_5) becomes table-defined during a query cycle and the simulator doesn't abort, then the 2chain $(5, 1, y_5, x_1, k)$ belongs to a complete path triggered in the query cycle.*

Proof. This follows by inspection of the pseudocode and by Lemma 5 (e) (we can check both y_5 and x_1 belong to the triggered complete path). $\square$

[14] Note that the complete path does not exist at the point of the execution when the 2chain is triggered.

Lemma 15. *Consider a good and safe query cycle in an 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 query cycle 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 Dec(k,y_5) can only occur during the call to 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 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 16. *Consider a good and safe query cycle in an 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 FindNewPaths$(i+2,+,r(C))$, or during the call the 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 $\mathcal{E}$. The only way x_{i+3} can become table-defined during the query cycle is because of a call to Assign$(i+3,x'_{i+3},y'_{i+3})$ with $x'_{i+3}=x_{i+3}$ 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, if $i\neq 3$ we have $x'_{i+3}=p_{i+2}(r(C'))\oplus k'$ where $k'\in\mathcal{K}$ and $p_{i+2}(r(C'))$ is read during the query cycle, so that $\mathcal{P}\cap(\mathcal{E}\oplus\mathcal{K})\neq\emptyset$ 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 $\mathcal{C}\cap\mathcal{E}\neq\emptyset$ and BadIC happens. In all cases this contradicts the assumption that the query cycle 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 $x_{i-1}\in\mathcal{E}$. The only way x_{i-1} can become table-defined during the $(i,i+1)$-query cycle is because of a call to 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 ReadTape$(i-1,-,y_{i-1})$. This implies that $\mathcal{P}\cap\mathcal{E}\neq\emptyset$ and hence $\mathsf{BadPerm}$ occurs, contradicting the assumption that the query cycle 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 17. *Consider a good and safe query cycle in an 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 $\text{Next}(5, y_5, k)$ 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 query cycle 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 15 and 16, 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 it 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 $\text{Assign}(i+2, x_{i+2}, p_{i+2}(x_{i+2}))$ occurred. By Lemma 5 (d), x_{i+2} is either the initiating query, in which case it is in $\mathcal{E}$ (by definition), or the endpoint of some triggered $(i, i+1)$-2chain D, in which case it is in $\mathcal{E}$ if $r(D)$ is non-dummy at τ_0, or in $\mathcal{C}$ if $r(D)$ is dummy at τ_0 (by Lemma 8). (Note that $r(D)$ can be dummy at τ_0 only 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 $x_{i+3} \in \mathcal{H}$. The key $k = y_{i+1} \oplus x_{i+2} = p_{i+2}(x_{i+2}) \oplus x_{i+3}$, rearranging the terms we have $x_{i+2} \oplus p_{i+2}(x_{i+2}) = y_{i+1} \oplus x_{i+3}$ where the left-hand side is in $\mathcal{P}^*$ and the right-hand side is in $\mathcal{H}^{\oplus 2}$. This implies $\mathcal{P}^* \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs, contradicting the assumption that the query cycle 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 $\text{Assign}(i+3, x'_{i+3}, y'_{i+3})$ with $x'_{i+3} = x_{i+3}$

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. Since $i \in \{1, 2, 5\}$, the adapted query

$$x'_{i+3} = p_{i+2}(x'_{i+2}) \oplus k' = p_{i+2}(x'_{i+2}) \oplus x'_{i+2} \oplus y'_{i+1}$$

where $x'_{i+2} = r(C')$. From $x'_{i+3} = x_{i+3}$, we have

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

rearranging the terms we have

$$x_{i+2} \oplus p_{i+2}(x_{i+2}) \oplus x'_{i+2} \oplus p_{i+2}(x'_{i+2}) = y_{i+1} \oplus y'_{i+1}. \tag{9}$$

Since C' is triggered in the query cycle, we have $x'_{i+2} \oplus p_{i+2}(x'_{i+2}) \in \mathcal{P}^*$ and $y'_{i+1} \in \mathcal{H}$. Hence, if $x_{i+2} \neq x'_{i+2}$, then $\mathcal{P}^{*\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs, whereas if $x_{i+2} = x'_{i+2}$ then

$$k = p_{i+2}(x_{i+2}) \oplus x_{i+3} = p_{i+2}(x'_{i+2}) \oplus x'_{i+3} = k',$$

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 by Lemma 14, 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 Assign($2, x'_2, y'_2$) with $x'_2 = x_2$ resulting from a call to 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 $k' \in \mathcal{K}$. Hence, $x'_2 = x_2$ implies that $p_1(x_1) \oplus p_1(x'_1) = k \oplus k'$. If $x_1 \neq x'_1$, then $\mathcal{P}^{\oplus 2} \cap (\mathcal{H} \oplus \mathcal{K}) \neq \emptyset$ 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^{-1}_{i+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 $P^{-1}_{i+2}(y_{i+2})$ 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 15 and 16. 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

$$y_{i+2} \oplus x_{i+3} \oplus y_{i+3} \oplus x_{i-1} = 0. \tag{10}$$

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 $x_{i+3} = p_{i+2}(x'_{i+2}) \oplus k'$ and $y_{i+3} = p^{-1}_{i-1}(y'_{i-1}) \oplus k'$, so $x_{i+3} \oplus y_{i+3} = p_{i+2}(x'_{i+2}) \oplus p^{-1}_{i-1}(y'_{i-1}) \in \mathcal{P}^{\oplus 2}$.
- 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 $x_{i-1} = p^{-1}_{i-1}(y''_{i-1}) \in \mathcal{P}$.

Finally, note that for the case where x_{i+3} was table-undefined at τ_0, we can assume that $x_{i+2} \neq x'_{i+2}$ and $y''_{i-1} \neq y'_{i-1}$, 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 (10) 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 $\mathcal{P}^{\oplus i} \cap \mathcal{H}^{\oplus 4-i} \neq \emptyset$ for some $1 \leq i \leq 4$ (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. (10) yields

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

and hence $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$.)

– 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 $k, x_1, y_5 \in \mathcal{H}$. 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 $\mathsf{BadPerm}$ 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 $x_2 = y_1 \oplus k$ 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, then we have $k, x_1, y_5 \in \mathcal{H}$ and either x_1 or y_5 was table-undefined at τ_0. 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 $\mathsf{BadPerm}$ 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.

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 15, 16, and 17 under a simpler to grasp form in the following lemma. To state it more concisely, we introduce the following definition.

Definition 21. We say a 2chain C *induces a 3chain* if C is table-defined and at least one of its endpoints is table-defined.

Lemma 18. *Consider a good and safe query cycle in an 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 15, 16, and 17 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 19. *Consider a point τ_0 in an execution of* G_2 *which is either the beginning of a (cipher) query cycle, or the end of the execution, and assume that* *all query cycles were safe and no bad event has occurred until τ_0. 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. By assumption in the lemma, C 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} in a cipher query cycle (which was good by assumption); 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 good and safe by assumption). But according to Lemma 18, at the end of this previous query cycle, C was complete, and this still holds at τ_0, hence the result. $\square$

Lemma 20. *Consider an $(i, i+1)$-query cycle with initiating query (i_0, δ_0, z_0) in an execution of G_2. As long as BadIC doesn't occur, if an $(i, i+1)$-2chain C is triggered during the query cycle, there exists a sequence of $(i, i+1)$-2chains $(C_0, \dots, C_t)$ triggered in this order such that $C_t = C$, and,* at the beginning of the query cycle, *either $z_0 = r(C_0) \neq \perp$, $\ell(C_1) = \ell(C_0) \neq \perp$, $r(C_2) = r(C_1) \neq \perp$, etc. if $(i_0, \delta_0) = (i+2, +)$, or $\ell(C_0) = z_0$, $r(C_1) = r(C_0) \neq \perp$, $\ell(C_2) = \ell(C_1) \neq \perp$, etc. if $(i_0, \delta_0) = (i-1, -)$.*

Proof. The existence of sequence $(C_0, \dots, C_t)$ 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 $\mathcal{C} \cap \mathcal{E} \neq \emptyset$ and BadIC happens (and from the symmetric observations for a $(1, 2)$-2chain with a dummy left endpoint). $\square$

Corollary 1. *Consider a query cycle in an execution of G_2. Let $\mathcal{R}$ be the set of 2chains satisfying the condition (of being triggered) in Lemma 20. Then we have:*

- *The set $\mathcal{R}$ is determined by the state of tables at the beginning of the query cycle;*
- *As long as BadIC doesn't occur, only 2chains in $\mathcal{R}$ can be triggered;*
- *If BadIC doesn't occur in the query cycle, $\mathcal{R}$ is the set of triggered 2chains in the query cycle.*

Proof. The set $\mathcal{R}$ is determined by the table states at the beginning of the query cycle by definition (cf. Lemma 20). As long as BadIC hasn't occur in the query cycle, by Lemma 20 only 2chains in $\mathcal{R}$ can be triggered. Moreover, by inspection of the pseudocode, we can see that every 2chain satisfying the condition in Lemma 20 will be triggered in the query cycle. Therefore, if BadIC occurs, a 2chain is triggered if and only if it is in $\mathcal{R}$. $\square$

Lemma 21. *Consider an $(i, i+1)$-query cycle in an execution of G_2, before which no bad event has occurred and in which BadIC doesn't occur. 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 20, there exists a sequence $(C_0, \dots, C_t)$ of triggered $(i, i+1)$-2chains such that $C_t = C$, and, at τ_0, either $r(C_0) = z_0$, $\ell(C_1) = \ell(C_0) \neq \perp$, $r(C_2) = r(C_1) \neq \perp$, etc., or $\ell(C_0) = z_0$, $r(C_1) = r(C_0) \neq \perp$, $\ell(C_2) = \ell(C_1) \neq \perp$, etc.

By definition of a complete 2chain, $r(C)$ and $\ell(C)$ are non-dummy and table-defined at τ_0. 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 19 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. $\square$

Lemma 22. *In an execution of G_2, a query cycle is safe if no bad event has occurred before the query cycle and BadIC doesn't occur in the query cycle. In particular, 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 19, since all previous query cycles were safe and no bad event has occurred, this implies that C was complete at the beginning of the query cycle, which by Lemma 21 (and again the fact that all previous query cycles were safe and no bad event has occurred) implies that C cannot be triggered during the query cycle, a contradiction. $\square$

The main result of this section now follows.

Lemma 23. *The simulator does not abort in a good execution of G_2.*

Proof. This is a direct consequence of Lemmas 12 and 22. $\square$

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.

From Lemma 22, 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 23, 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 22 or 23).

First we prove some properties of good executions that will be useful in the termination argument.

Lemma 24. *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. Let $i = 2$ and assume towards a contradiction that there exist distinct table-defined queries $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ such that

$$x_2 \oplus y_2 = x'_2 \oplus y'_2 \tag{11}$$

(the case with $i = 4$ is symmetric).

First we consider the case where $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ get defined in different query cycles. Let $(2, x_2, y_2)$ be defined in an earlier query cycle. Consider the query cycle in which $(2, x'_2, y'_2)$ gets defined, at the beginning of which $(2, x_2, y_2)$ is table-defined and hence $x_2 \oplus y_2 \in \mathcal{H}$ (recall Definition 12). We discuss the possible ways that $(2, x'_2, y'_2)$ gets defined.

If $(2, x'_2, y'_2)$ is defined in a call to ReadTape$(2, +, x'_2)$ or ReadTape$(2, -, y'_2)$, then we have $x'_2 \oplus y'_2 \in \mathcal{P}^*$. Equation (11) implies $\mathcal{P}^* \cap \mathcal{H} \neq \emptyset$ and hence $\mathsf{BadPerm}$ occurs. If $(2, x'_2, y'_2)$ is adapted, we have $x'_2 \oplus y'_2 \in \mathcal{A}$; (11) implies $\mathcal{A} \cap \mathcal{H} \neq \emptyset$ and $\mathsf{BadAdapt}$ occurs.

Then we consider the case where $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ get defined in the same query cycle. Again we discuss how they are defined, which depends on the type of the query cycle. The query cycle cannot be a $(1, 2)$- or $(2, 3)$-query cycle, in which no permutation query at position 2 gets defined.

If the query cycle is a $(3, 4)$-query cycle, then the queries are defined through calls to ReadPath$(2, +, x_2)$ and ReadPath$(2, +, x'_2)$ respectively. Thus, $x_2 \oplus y_2$ and $x'_2 \oplus y'_2$ are different elements of $\mathcal{P}^*$; (11) implies $\mathcal{P}^{*\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs. If the query cycle is a $(5, 1)$-query cycle, the queries are defined through calls to ReadPath$(2, -, y_2)$ and ReadPath$(2, -, y'_2)$, and the rest of the proof is the same as the last case. If the query cycle is a $(4, 5)$-query cycle, then both $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ are adapted and thus $x_2 \oplus y_2, x'_2 \oplus y'_2 \in \mathcal{A}$. Equation (11) implies $\mathcal{A}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ and $\mathsf{BadAdapt}$ occurs. □

The following lemma can be thought of as a "harder version" of Lemma 24, and the proof ideas of the two lemmas are similar.

Lemma 25. *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*

$$\sum_{j=1}^{4} (x_i^{(j)} \oplus y_i^{(j)}) = 0. \tag{12}$$

Proof. By symmetry, we only consider $i = 2$. Assume towards a contradiction that there exist distinct table-defined queries $(2, x_2^{(j)}, y_2^{(j)})$ for $j = 1,2,3,4$ satisfying Eq. (12). We assume without loss of generality that $(2, x_2^{(1)}, y_2^{(1)})$ is the last to be table-defined among the four queries.

Consider the query cycle in which $(2, x_2^{(1)}, y_2^{(1)})$ gets defined. The other queries are either table-defined at the beginning of the query cycle or are defined during the same query cycle. The query cycle cannot be a $(1,2)$- or $(2,3)$-query cycle, in which no permutation query at position 2 gets defined.

First we consider the case where the query cycle is a $(3,4)$-query cycle. If $(2, x_2^{(j)}, y_2^{(j)})$ gets defined in the query cycle, it must be defined through a call to $\mathrm{ReadTape}(2, +, x_2^{(j)})$ and $x_2^{(j)} \oplus y_2^{(j)} \in \mathcal{P}^*$; otherwise the query should be table-defined at the beginning of the query cycle and $x_2^{(j)} \oplus y_2^{(j)} \in \mathcal{H}$. If t of the four queries get defined in the query cycle ($1 \le t \le 4$ since we are considering the query cycle in which $(1, x_2^{(1)}, y_2^{(1)})$ gets defined), then the other $4 - t$ queries are table-defined at the beginning of the query cycle. Equation (12) implies $\mathcal{P}^{*\oplus t} \cap \mathcal{H}^{\oplus(4-t)} \neq \emptyset$ with $1 \le t \le 4$ (note that the entries in $\mathcal{P}^*$ are not canceled because the queries are distinct), so $\mathsf{BadPerm}$ occurs.

The case where the query cycle is a $(5,1)$-query cycle is similar to the last case (the queries are defined through calls to $\mathrm{ReadPath}(2, -, y_2^{(j)})$ in this case).

Then we consider the case where the query cycle is a $(4,5)$-query cycle. If $(2, x_2^{(j)}, y_2^{(j)})$ gets defined in the query cycle, it must be adapted and $x_2^{(j)} \oplus y_2^{(j)} \in \mathcal{A}$. Similarly, assume t of the four queries are adapted in the query cycle ($1 \le t \le 4$), then the other $4 - t$ queries are table-defined at the beginning of the query cycle. Equation (12) implies $\mathcal{A}^{\oplus t} \cap \mathcal{H}^{\oplus(4-t)} \neq \emptyset$ (the entries in $\mathcal{A}$ are not canceled because the queries are distinct) and $\mathsf{BadAdapt}$ occurs. □

We analyze the running time of the simulator in a good execution of G_2. A large part of this analysis consists of upper bounding the size of tables T, T^{-1}, P_i, P_i^{-1}. Since $|T| = |T^{-1}|$ and $|P_i| = |P_i^{-1}|$ we state the results only 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 22. 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 $\mathrm{FindNewPaths}(i, \delta, z)$. 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 4 special types of triggered 2chains under a common name.

Definition 23. 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 24. 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.

Lemma 26. *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 in which C is triggered, C belongs to a complete path which must be Π by Lemma 4. By Lemma 21, this implies that neither C nor any other 2chain belonging to Π will ever be triggered in any subsequent query cycle. □

Lemma 27. *For* $i \in \{1, \dots, 5\}$*, 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* Query$(i, \cdot, \cdot)$*,*
– $(i+1, i+2)$*-2chains that were* $(i+3, +)$*-triggered,*
– $(i-2, i-1)$*-2chains that were* $(i-3, -)$*-triggered,*
– $(i+2, i+3)$*-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).

We first consider entries that were added by a call to ReadTape during an $(i+1, i+2)$- or an $(i-2, i-1)$-query cycle. 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.

$N_{i,+}$ 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 at most the number of distinguisher's calls to 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 (as 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 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 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. 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 (irrespective of whether they are $(i+1,-)$- or $(i+4,+)$-triggered). The result follows. □

The following lemma contains the standard "bootstrapping" argument introduced in [19]:

Lemma 28. *In a good execution of G_2, at most q wrapping 2chains are 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 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 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 21.

Thus, 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 a 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. By Lemma 6, when the simulator makes a cipher query which modifies

T/T^{-1}, the corresponding 2chain should have already been triggered. This 2chain will be complete at the end of the query cycle and cannot be triggered again (Lemma 26). □

Lemma 29. *In a good execution of* G_2*, one always has* $|P_3| \leq 2q$*.*

Proof. By Lemma 27, 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. Further note that any 2chain mentioned for the 3 other types are wrapping 2chains. Hence, by Lemma 28, there are at most q such entries in total, so that $|P_3| \leq 2q$. □

Lemma 30. *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. If $C = (4, 5, y_4, x_5, k)$ was $(3, -)$-triggered, $(5, +, x_5)$ 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 29. 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 24 (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 $(5, +, x_5)$ was a non-initiating pending query in the same query cycle in which 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 24 we have $x_4 = x'_4$ and $C' = (3, 4, y_3, x_4, k) = C$, contradicting our assumption. 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 24 and 25 (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. Combining both cases, the number of $(3, -)$- or $(5, +)$-triggered 2chains is at most $6q^2 - 2q$. □

Lemma 31. *In a good execution of* G_2*,* $|P_2| \leq 6q^2$ *and* $|P_4| \leq 6q^2$*.*

Proof. By Lemma 27, 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 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. Any 2chain mentioned for the other cases are either wrapping, $(3, -)$-triggered, or $(5, +)$-triggered 2chains. By Lemmas 28 and 30, 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 32. *In a good execution of* G_2*, at most* $12q^3$ *2chains are triggered in total.*

Proof. Since the simulator doesn't abort in good executions by Lemma 23, any triggered 2chain belongs to a complete path at the end of the execution. By Lemma 26, 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 29 and 31 is at most $2q \cdot 6q^2 = 12q^3$. □

Lemma 33. *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 34. *In a good execution of* G_2*,* $|P_1| \le 12q^3 + q$ *and* $|P_5| \le 12q^3 + q$*.*

Proof. By Lemma 27, 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 Query$(1, \cdot, \cdot)$, which is at most q, and the total number of triggered 2chains, which is at most $12q^3$ by Lemma 32. Therefore, the size of $|P_1|$ is at most $12q^3 + q$. The same reasoning applies to $|P_5|$. □

Lemma 35. *In good executions of* G_2*, the simulator runs in time* $O(q^8)$ *and uses* $O(q^3)$ *space.*

Proof. By Lemmas 29, 31 and 34, the total number of queries that are defined during an execution is $O(q^3)$. In a non-aborting execution, every call to Assign results in an undefined query becoming defined. Therefore, Assign is called $O(q^3)$ times, which implies ReadTape and AdaptPath are called $O(q^3)$ times. These procedures run in constant time, so their total running time 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 FindNewPaths(i, δ, z), each iteration runs in constant time; the running time of the call is either $|P_{i-2}| \cdot |P_{i-1}|$ or $|P_{i+1}| \cdot |P_{i+2}|$[15]. 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 29, 31 and 34. 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. □

5.4.1 Probability of Good Executions

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

Definition 25 (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. $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_1)$.

[15] 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 29, 31 and 34.

Definition 26 (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 $(1, 2)$. Note that 2cycle are a "special case" of a 4cycle as we can concatenate a 2cycle with itself to obtain a 4cycle (this is formalized in the proof of Corollary 2).

Lemma 36. *In an execution of* G_2*, a 4cycle doesn't appear at* $(4, 5)$ *or at* $(1, 2)$ *for the first time in a good cipher query cycle.*

Proof. Assume towards a contradiction that table-defined 2chains (C_1, C_2, C_3, C_4) form a 4cycle for the first time in a good cipher query cycle where $C_j = (4, 5, y_4^{(j)}, x_5^{(j)}, k^{(j)})$, i.e., the 4cycle did not exist before the cipher query cycle.

If (δ, k, z) is already table-defined when the cipher query is issued, then no new queries are defined and the 4cycle cannot be formed. If (δ, k, z) is table-undefined, then only the cipher query itself gets table-defined during the cipher query cycle of (δ, k, z). For the 4cycle to appear, the cipher query should cause $(-, k^{(j)}, y_5^{(j)})$ to be table-defined for some $j \in \{1, \dots, 4\}$. 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, $r(C_1) = r(C_4)$ by definition of a 4cycle, 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 $x_1^{(1)} = x_1^{(4)}$ where $x_1^{(1)} \in \mathcal{C}$ and $x_1^{(4)} \in \mathcal{H}$. On the other hand, if $(\delta, k, z) = (+, k^{(1)}, x_1^{(1)})$, we have $y_5^{(1)} \in \mathcal{C}$; however, we also have $y_5^{(1)} \in \mathcal{H}$ since C_1 is table-defined. Thus BadIC occurs regardless of the direction of the cipher query as we have $\mathcal{C} \cap \mathcal{H} \neq \emptyset$.

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. $\square$

Lemma 37. *Consider a good query cycle in an execution of* G_2*. If a 4cycle does not exist at* $(4, 5)$ *or at* $(1, 2)$ *at the beginning of the query cycle, then a 4cycle does not exist at* $(4, 5)$ *or at* $(1, 2)$ *at the end of the query cycle.*

Proof. We prove by contradiction. Without loss of generality, assume that a 4cycle (C_1, C_2, C_3, C_4) at $(4, 5)$ exists at the end of the query cycle, where $C_j = (4, 5, y_4^{(j)}, x_5^{(j)}, k^{(j)})$.

Recall that by Definition 26, the endpoints of the 2chains are non-dummy and table-undefined. Let $r(C_j) = T^{-1}(k^{(j)}, y_5^{(j)}) = x_1^{(j)} \neq \perp$.

First we show the cipher queries $T(k^{(j)}, x_1^{(j)}) = y_5^{(j)}$ are table-defined at the beginning of the query cycle. If $T(k^{(j)}, x_1^{(j)}) = y_5^{(j)}$ is table-undefined at the beginning of the query cycle, it must get defined during the query cycle. By Lemma 14, the 2chain $(5, 1, x_1^{(j)}, y_5^{(j)}, k^{(j)})$ belongs to a complete path which was triggered during the query cycle. In particular, $x_1^{(j)}$, an endpoint of C_j, is table-defined at the end of the query cycle, contradicting Definition 26. Hence, we have $k^{(j)} \in \mathcal{H}$, $x_1^{(j)} \in \mathcal{H}$ and $y_5^{(j)} \in \mathcal{H}$ for $j \in \{1, \ldots, 4\}$.

Since the 4cycle first appears after the query cycle, at least one of the permutation queries $(4, x_4^{(j)}, y_4^{(j)})$ and $(5, x_5^{(j)}, y_5^{(j)})$ is table-undefined at the beginning of the query cycle and gets defined during the query cycle. Without loss of generality, we assume $(4, x_4^{(1)}, y_4^{(1)})$ or $(5, x_5^{(1)}, y_5^{(1)})$ gets table-defined during the query cycle. We consider the possible types of the query cycle.

Case of 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 adaptations occur at position 4.

If $(4, x_4^{(1)}, y_4^{(1)})$ is table-defined at the beginning of the query cycle, then according to our assumption, $(5, x_5^{(1)}, y_5^{(1)})$ gets defined in a call to ReadTape$(5, -, y_5^{(1)})$ and hence $x_5^{(1)} \in \mathcal{P}$. We also have $x_5^{(1)} = y_4^{(1)} \oplus k^{(1)} \in \mathcal{H}^{\oplus 2}$, which implies that $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and BadPerm occurs.

If $(4, x_4^{(1)}, y_4^{(1)})$ is adapted during the query cycle, let $C' = (1, 2, y_1', x_2', k')$ denote the triggered 2chain such that Assign$(4, x_4^{(1)}, y_4^{(1)})$ is called in AdaptPath(C'). By Lemma 10, we have $y_4^{(1)} = p_5^{-1}(y_5') \oplus k'$ where $y_5' = \ell(C')$, $p_5^{-1}(y_5') \in \mathcal{P}$ and $k' \in \mathcal{K}$. Note that

$$p_5^{-1}(y_5') \oplus k' = y_4^{(1)} = x_5^{(1)} \oplus k^{(1)} \tag{13}$$

where $k^{(1)} \in \mathcal{H}$; we consider the following two cases:

- If $(5, x_5^{(1)}, y_5^{(1)})$ is table-defined at the beginning of the query cycle, then $x_5^{(1)} \in \mathcal{H}$ and (13) implies $\mathcal{P} \cap (\mathcal{H}^{\oplus 2} \oplus \mathcal{K}) \neq \emptyset$.
- If $(5, x_5^{(1)}, y_5^{(1)})$ gets defined in a call to ReadTape$(5, -, y_5^{(1)})$ during the query cycle, we have $x_5^{(1)} = p_5^{-1}(y_5^{(1)}) \in \mathcal{P}$. If $y_5^{(1)} = y_5'$, then the 2chain C_1 belongs to the complete path containing the triggered 2chain C' and cannot be in the 4cycle (cf. Definition 26). If $y_5^{(1)} \neq y_5'$, (13) implies $\mathcal{P}^{\oplus 2} \cap (\mathcal{H} \oplus \mathcal{K}) \neq \emptyset$.

BadPerm occurs in both cases.

Case of 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 adaptations occur at position 5.

If $(5, x_5^{(1)}, y_5^{(1)})$ gets adapted during the query cycle, then by Lemma 10 we have $y_5^{(1)} \in \mathcal{C}$. Since $y_5^{(1)} \in \mathcal{H}$ (recall that this is because the cipher query is table-defined at the beginning of the query cycle), we have $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC occurs.

If $(5, x_5^{(1)}, y_5^{(1)})$ is table-defined at the beginning of the query cycle, then by our assumption $(4, x_4^{(1)}, y_4^{(1)})$ must be defined in a call to ReadTape$(4, +, x_4^{(1)})$

and $y_4^{(1)} = p_4(x_4^{(1)}) \in \mathcal{P}$. We also have $y_4^{(1)} = x_5^{(1)} \oplus k^{(1)} \in \mathcal{H}^{\oplus 2}$, so $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs.

Case of a $(3,4)$-query cycle. From Table 1, we can see that in a $(3,4)$-query cycle, calls to $\text{ReadTape}(5,+,\cdot)$ occur and no new permutation query of the form $(4,\cdot,\cdot)$ gets table-defined.

By our assumption, $(5, x_5^{(1)}, y_5^{(1)})$ is defined in a call to $\text{ReadTape}(5,+,x_5^{(1)})$ and hence $y_5^{(1)} = p_5(x_5^{(1)}) \in \mathcal{P}$. Since $y_5^{(1)} \in \mathcal{H}$, we have $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs.

Case of a $(4,5)$-query cycle. In a $(4,5)$-query cycle, no new permutation queries of the form $(4,\cdot,\cdot)$ or $(5,\cdot,\cdot)$ get table-defined.

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

By our assumption, $(4, x_4^{(1)}, y_4^{(1)})$ is defined in a call to $\text{ReadTape}(4,-,y_4^{(1)})$ and we have $x_4^{(1)} = p_4^{-1}(y_4^{(1)}) \in \mathcal{P}$. By Definition 26, we have

$$x_4^{(1)} \oplus k^{(1)} = y_3^{(1)} = y_3^{(2)} = x_4^{(2)} \oplus k^{(2)} \tag{14}$$

where $y_3^{(1)} = \ell(C_1)$ and $y_3^{(2)} = \ell(C_2)$, and where $k^{(1)}, k^{(2)} \in \mathcal{H}$. We consider whether $(4, x_4^{(2)}, y_4^{(2)})$ is table-defined at the beginning of the query cycle.

If $(4, x_4^{(2)}, y_4^{(2)})$ is table-defined at the beginning of the query cycle, we have $x_4^{(2)} \in \mathcal{H}$. Then (14) implies $\mathcal{P} \cap \mathcal{H}^{\oplus 3} \neq \emptyset$, so $\mathsf{BadPerm}$ occurs.

If $(4, x_4^{(2)}, y_4^{(2)})$ gets defined in a call to $\text{ReadTape}(4,-,y_4^{(2)})$ during the query cycle, we have $x_4^{(2)} = p_4^{-1}(y_4^{(2)}) \in \mathcal{P}$. If $y_4^{(1)} = y_4^{(2)}$, we have $x_4^{(1)} = x_4^{(2)}$ and, by (14), $k^{(1)} = k^{(2)}$. Then $x_5^{(1)} = y_4^{(1)} \oplus k^{(1)} = y_4^{(2)} \oplus k^{(2)} = x_5^{(2)}$, which implies $C_1 = C_2$, violating Definition 26. If $y_4^{(1)} \neq y_4^{(2)}$, (14) implies $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs. □

Lemma 38. *At the beginning of a query cycle in an execution of* G_2*, if no bad event has occurred, there does not exist a 4cycle at* $(4,5)$ *or at* $(1,2)$*.*

Proof. This follows immediately from Lemmas 36 and 37. □

Corollary 2. *At the beginning of a query cycle in an execution of* G_2*, if no bad event has occurred, there does not exist a 2cycle at* $(4,5)$ *or at* $(1,2)$*.*

Proof. We only need to prove that a 2cycle exists at $(4,5)$ or $(1,2)$ only if a 4cycle exists at the same position, and this is a direct corollary of Lemma 38.

Assume there exists a 2cycle at (say) $(4,5)$ consisting of table-defined 2chains $C = (4,5,y_4,x_5,k)$ and $C' = (4,5,y_4,x_5,k)$. By Definition 25, we have $C \neq C'$, 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 $r(C') = r(C)$. □

Probability of Good Executions. In this section, we upper bound the probability of the bad events. Since we don't assume the execution is good, the lemmas that assume the execution to be good cannot be used. However, most lemmas 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". The following lemma gives a stronger observation.

Lemma 39. *Consider a query cycle or a cipher query cycle in an execution of G_2, before which no bad event has occurred. Then as long as* BadIC *hasn't occurred in the current (cipher) query cycle, we have*

(a) (Triggered 2chains, only applies to a query cycle) At most $12q^3$ 2chains are triggered, including those triggered in the current query cycle;
(b) (Table sizes) We have $|P_1| \leq 13q^3$, $|P_2| \leq 6q^2$, $|P_3| \leq 2q$, $|P_4| \leq 6q^2$, $|P_5| \leq 13q^3$ and $|T| \leq 13q^3$; the bounds hold even if we include the queries that are about to be defined when completing the triggered 2chains.

Proof. First we prove the lemma assuming the current (cipher) query cycle is good and is completed. We can treat the partial execution until the end of the current (cipher) query cycle as a shorter good execution (the execution is good since no bad event occurs before or during the (cipher) query cycle). Then property (a) follows by Lemma 32 and (b) follows by Lemmas 29, 31, 33 and 34. (We note that the distinguisher issues less queries in the shorter execution, so the lemmas actually give a stronger bound.)

For the case of a cipher query cycle, only one cipher query is made and BadIC is the only possible bad event, so the cipher query cycle is good if BadIC doesn't occur for the distinguisher's cipher query, and the above result suffices.

For the case of a query cycle, by Corollary 1, let $\mathcal{R}$ define the set of triggered 2chains when the query cycle is good. Then $\mathcal{R}$ is determined by the state at the beginning of the query cycle. As long as BadIC hasn't occurred, by Corollary 1, the 2chains triggered in the query cycle must belong to $\mathcal{R}$, i.e., only a subset of 2chains triggered in good query cycle are triggered. Therefore, the number of triggered paths and the number of queries about to be defined is no more than those when the query cycle is good and is complete. This generalizes the result (in the first paragraph of the proof) to the more general case where we only require BadIC hasn't occurred (instead of requiring the query cycle is good). □

Lemma 40. *Consider a query cycle or a cipher query cycle in an execution of G_2, before which no bad event has occurred. The sizes of $\mathcal{H}$ and $\mathcal{E}$ are upper bounded by $159q^3$ and $386q^5$ respectively. Moreover, as long as* BadIC *hasn't occurred, the size of $\mathcal{K}$ is upper bounded by $12q^3$.*

Proof. Adding up the bounds in Lemma 39 (b), there are at most $40q^3$ table-defined permutation queries and $13q^3$ table-defined cipher queries at the beginning of the query cycle. By Definition 12, the size of $\mathcal{H}$ is at most

$$3 \cdot 40q^3 + 3 \cdot 13q^3 \le 159q^3.$$

By Definition 6, the number of table-defined $(5,1)$-2chains is upper bounded by $|T| \le 13q^3$ (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 $2(13q^3 \cdot 6q^2 + 6q^2 \cdot 2q) \le 180q^5$ (since each pair of table-defined queries uniquely determine an $(i,i+1)$-2chain). Thus, the size of $\mathcal{E}$ can be upper bounded by

$$2 \cdot (13q^3 + 180q^5) \le 386q^5.$$

By Lemma 39 (a), as long as BadIC doesn't occur, at most $12q^3$ 2chains are triggered in the current query cycle, so the size of $\mathcal{K}$ is at most $12q^3$. □

Lemma 41. *Consider a cipher query in an execution of* G_2*, before which no bad event has occurred. If the cipher query is already table-defined, then* BadIC *doesn't occur; otherwise,* BadIC *occurs with probability at most* $1116q^5/2^n$*.*

Proof. The lemma trivially holds if $13q^3 > 2^{n-1}$, so we can assume $13q^3 < 2^{n-1}$ in this proof. Without loss of generality, we consider a cipher query to $\text{Enc}(k, x_1)$.

If $x_1 \in T$ when the query occurs, BadIC doesn't occur because the tape $\mathsf{ic}/\mathsf{ic}^{-1}$ is not read. Now we consider the case $(k, x_1) \notin T$. Assume the tape entry being read is $\mathsf{ic}(k, x_1) = y_5$, which implies $y_5 \in \mathcal{C}$.

In the following proof, we let $\bar{\mathcal{C}}$ denote the subset of $\mathcal{C}$ that contains only the entries of $\mathsf{ic}/\mathsf{ic}^{-1}$ read before $\mathsf{ic}(k, x_1)$. I.e., $\bar{\mathcal{C}}$ contains the elements of $\mathcal{C}$ that are fixed before $\mathsf{ic}(k, x_1)$ is sampled.

By assumption, BadIC hasn't occurred in the current query cycle or cipher query cycle, so by Lemma 39 (b) we have $|T| \le 13q^3$. Hence, $\bar{\mathcal{C}}$ contains at most $13q^3$ elements (since each of them corresponds to a distinct defined cipher query). Moreover, the size of $\mathcal{H}$ and $\mathcal{E}$ are upper bounded by $159q^3$ and $386q^5$ respectively (cf. Lemma 40). BadIC occurs if the value of $\mathsf{ic}(k, x_1)$ is in $\bar{\mathcal{C}}$, $\mathcal{H}$ or $\mathcal{E}$, and the three sets contain at most $13q^3 + 159q^3 + 386q^5 \le 558q^5$ values in total.

Since $\mathsf{ic}(k, \cdot)/\mathsf{ic}^{-1}(k, \cdot)$ encodes a permutation and at most $13q^3$ entries of $\mathsf{ic}/\mathsf{ic}^{-1}$ have been read (cf. Lemma 39), the value of $\mathsf{ic}(k, x_1)$ is uniformly distributed among at least $2^n - 13q^3 \ge 2^{n-1}$ values. Therefore, each value is chosen with probability at most $1/2^{n-1}$, so BadIC occurs with probability at most $558q^5/2^{n-1} = 1116q^5/2^n$. □

When upper bounding the probabilities of $\mathsf{BadPerm}$ and $\mathsf{BadAdapt}$, we assume BadIC does not occur during the query cycle, which allows us to use Lemmas 22 and 39 and Corollary 1. In particular, the number of triggered 2chains (and thus the number of pending and adapted queries) in the query cycle are determined at the beginning of the query cycle and can be upper bounded.

Lemma 42. *Consider a query cycle in an execution of G_2, before which no bad event has occurred. Let $\mathcal{R}$ be defined as in Corollary 1 and let s be the size of $\mathcal{R}$. Assuming* BadIC *doesn't occur, then* BadPerm *occurs in the query cycle with probability at most $1.84 \times 10^7 \cdot (s+1)q^9/2^n$.*

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 Corollary 1, the 2chains in $\mathcal{R}$ are triggered. Thus, s 2chains are triggered and there are at most $s+1$ pending queries in the query cycle (since each triggered 2chain adds at most one pending query). Thus, at most $s+1$ entries of p_i/p_i^{-1} are read in the query cycle. We consider these entries in the same order as they are read, and compute the probability that BadPerm occurs for the first time when each entry is read.

Without loss of generality, we consider an entry $p_i(x_i)$ (the case of $p_i^{-1}(y_i)$ is symmetric). In the following proof, we let $\bar{\mathcal{P}}$ and $\bar{\mathcal{P}}^*$ denote the subsets of $\mathcal{P}$ and $\mathcal{P}^*$ that contain only the elements corresponding to the tape entries which are read before $p_i(x_i)$.

We first consider the sub-events of BadPerm involving the set $\mathcal{P}$. As we are computing the probability that BadPerm occurs for the first time after $p_i(x_i)$ is read, the event must involve the entry $p_i(x_i)$. Then the probability of a sub-event of the form $\mathcal{P}^{\oplus i} \cap \mathcal{S}$ is the probability that $p_i(x_i)$ hits a value in $\bar{\mathcal{P}}^{\oplus i-1} \oplus \mathcal{S}$. Therefore, we need to compute the probability that the value of $p_i(x_i)$ is in $\mathcal{H}^{\oplus 3}$, $\bar{\mathcal{P}} \oplus \mathcal{H}^{\oplus 2}$, $\bar{\mathcal{P}}^{\oplus 2} \oplus \mathcal{H}$, $\bar{\mathcal{P}}^{\oplus 3}$, $\mathcal{E}$, $\mathcal{E} \oplus \mathcal{K}$, $\mathcal{K} \oplus \mathcal{H}$, $\mathcal{K} \oplus \mathcal{H}^{\oplus 2}$, $\bar{\mathcal{P}} \oplus \mathcal{K}^{\oplus 2}$ or $\bar{\mathcal{P}} \oplus \mathcal{H} \oplus \mathcal{K}$.

The sub-events involving $\mathcal{P}^*$ are similar: the new entry of $\mathcal{P}^*$ is $p_i(x_i) \oplus x_i$ where x_i is determined at the beginning of the query cycle and $p_i(x_i)$ is an independent random value. BadPerm occurs for the first time if the value of $x_i \oplus p_i(x_i)$ is in $\mathcal{H}^{\oplus 3}$, $\bar{\mathcal{P}}^* \oplus \mathcal{H}^{\oplus 2}$, $\bar{\mathcal{P}}^{*\oplus 2} \oplus \mathcal{H}$, $\bar{\mathcal{P}}^{*\oplus 3}$ or $\mathcal{H} \oplus \mathcal{E}$.

Since there are $s+1$ pending queries in the query cycle, the sizes of $\bar{\mathcal{P}}$ and $\bar{\mathcal{P}}^*$ cannot exceed $s \leq 12q^3$ where the inequality is due to Lemma 39 (a). By Lemma 40, we have $|\mathcal{H}| \leq 159q^3$, $|\mathcal{E}| \leq 386q^5$ and $|\mathcal{K}| \leq 12q^3$. Since $|\mathcal{S}^{\oplus i}| \leq |\mathcal{S}|^i$ and $|\mathcal{S}_1 \oplus \mathcal{S}_2| = |\mathcal{S}_1| \cdot |\mathcal{S}_2|$, the sum of sizes of the aforementioned multisets (in both sub-cases) is no more than $9.2 \times 10^6 q^9$.

Since p_i/p_i^{-1} encodes a permutation and at most $13q^3$ entries of p_i/p_i^{-1} have been read (by Lemma 39 and the observation that entries that have been read correspond to distinct table-defined queries), the value of $p_i(x_i)$ is uniformly distributed among at least $2^n - 13q^3 \geq 2^{n-1}$ values, with each value being chosen with probability at most $1/2^{n-1}$. Hence, the probability that $p_i(x_i)$ or $x_i \oplus p_i(x_i)$ hits a value in the corresponding multisets is at most $1.84 \times 10^7 q^9/2^n$.

By a union bound over the $s+1$ entries read in the query cycle, we obtain the bound stated in the lemma. □

Lemma 43. *Consider a $(1,2)$- or $(4,5)$-query cycle of a G_2 execution, before which no bad event has occurred. Let $\mathcal{R}$ be defined as in Corollary 1 and let s be the size of $\mathcal{R}$. Assuming* BadIC *doesn't occur, then* BadAdapt *occurs in the query cycle with probability at most $8.9 \times 10^6 \cdot sq^9/2^n$.*

Proof. If $13q^3 \geq 2^{n-1}$, the lemma trivially holds. Thus we can assume $13q^3 \leq 2^{n-1}$ in this proof. We only give the proof for a $(4,5)$-query cycle; the proof for a $(1,2)$-query cycle is symmetric.

By Corollary 1, the 2chains in $\mathcal{R}$ are triggered in the query cycle. Thus s queries are adapted in the query cycle.

Recall $\mathsf{BadAdapt}$ is the event that $\mathcal{A}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$ for $i \geq 1$ and $i + j \leq 4$, i.e., there exist adapted queries $(2, x_2^{(t)}, y_2^{(t)})$ for $t \in [1,i]$ such that

$$\sum_{t=1}^{i} x_2^{(t)} \oplus y_2^{(t)} \in \mathcal{H}^{\oplus j}. \tag{15}$$

For $i = 1, 2, 3, 4$, we consider each fixed i-tuple of distinct adapted queries and upper bound the probability of (15), assuming BadIC doesn't occur.

Let $C_t = (4, 5, y_4^{(t)}, x_5^{(t)}, k^{(t)})$ be the triggered 2chain in which $(2, x_2^{(t)}, y_2^{(t)})$ is adapted. Since the adapted queries are distinct, C_t are distinct 2chains.

Let $y_3^{(t)} = \ell(C_t)$ and $x_1^{(t)} = r(C_t)$, where the evaluation takes place after the cipher query $T^{-1}(k^{(t)}, y_5^{(t)})$ has been defined (the cipher query must be defined before or during the query cycle since C_t is triggered) and hence $x_1^{(t)} \neq \perp$. Then $x_2^{(t)} \oplus y_2^{(t)} = p_1(x_1^{(t)}) \oplus p_3^{-1}(y_3^{(t)})$ where $p_1(x_1^{(t)}), p_3^{-1}(y_3^{(t)}) \in \mathcal{P}$ are sampled in the current query cycle. The left-hand side of (15) is

$$\sum_{t=1}^{i} x_2^{(t)} \oplus y_2^{(t)} = \sum_{t=1}^{i} p_1(x_1^{(t)}) \oplus p_3^{-1}(y_3^{(t)}) \tag{16}$$

and we show that the random values of $\mathcal{P}$ at the right-hand side of (16) are not all canceled out. This is obvious if $i = 1, 3$ since the number of $p_1(x_1^{(t)})$ is odd and the random values can only be canceled out in pairs.

If $i = 2$ and the random values in the RHS of (16) all cancel out, we have $x_1^{(1)} = x_1^{(2)}$ and $y_3^{(1)} = y_3^{(2)}$. We show that both cipher queries $T^{-1}(k^{(1)}, y_5^{(1)})$ and $T^{-1}(k^{(2)}, y_5^{(2)})$ are table-defined at the beginning of the query cycle, otherwise BadIC occurs: note that if $T^{-1}(k^{(t)}, y_5^{(t)})$ is defined during the query cycle, it is defined in a call to Dec by Lemma 6 and hence $x_1^{(t)} \in \mathcal{C}$. If both $T^{-1}(k^{(1)}, y_5^{(1)})$ and $T^{-1}(k^{(2)}, y_5^{(2)})$ are defined during the query cycle, then $\mathcal{C}$ contains two equal entries and BadIC occurs. If one of the cipher queries is table-defined at the beginning of the query cycle and the other is defined during the query cycle, we have $x_1^{(1)} = x_1^{(2)} \in \mathcal{C} \cap \mathcal{H}$ and BadIC also occurs.

Since BadIC doesn't occur by assumption, both cipher queries are table-defined at the beginning of the query cycle and so the endpoints of C_1 and C_2 are non-dummy. By Lemma 22, the query cycle is safe, so the endpoints of the triggered 2chains C_1 and C_2 are table-undefined at the beginning of the query cycle. By Definition 25, C_1 and C_2 form a 2cycle at the beginning of the query cycle, contradicting Corollary 2.

If $i = 4$ and the random values in the RHS of (16) all cancel out, assume without loss of generality that $x_1^{(1)} = x_1^{(2)}$ and $x_1^{(3)} = x_1^{(4)}$. Similarly to the case of $i = 2$, we can prove that the cipher queries $T^{-1}(k^{(t)}, y_5^{(t)})$ are table-defined at the beginning of the query cycle, otherwise BadIC occurs. We discuss the following possibilities, omitting the detailed arguments that the 2chains form a 2cycle/4cycle (the first two conditions of Definition 25/26 are proved similarly to the case of $i = 2$, and the last condition is given explicitly in the discussion):

- $y_3^{(1)} = y_3^{(2)}$ and $y_3^{(3)} = y_3^{(4)}$: The 2chains C_1 and C_2 form a 2cycle at $(4, 5)$ at the beginning of the query cycle, contradicting Corollary 2.
- $y_3^{(2)} = y_3^{(3)}$ and $y_3^{(4)} = y_3^{(1)}$: The 2chains C_1, C_2, C_3 and C_4 form a 4cycle at $(4, 5)$ at the beginning of the query cycle, contradicting Lemma 38.
- $y_3^{(1)} = y_3^{(3)}$ and $y_3^{(2)} = y_3^{(4)}$: the same as the last item except with C_1 and C_2 swapped, so it also contradicts Lemma 38.

Hence, the random values in the RHS of (16) cannot all cancel out.

As discussed in the proof of Lemma 42, the values of $p_1(x_1^{(t)})$ and $p_3^{-1}(y_3^{(t)})$ are uniformly and independently distributed among at least $2^n - 13q^3 \geq 2^{n-1}$ values. The right-hand side of (16) contains at least one of these random values, so it equals any specific value with probability at most $1/2^{n-1}$ (indeed, with all but one random value fixed, it is uniformly distributed over at least 2^{n-1} values).

By Lemma 40, the multiset $\mathcal{H}$ contains at most $159q^3$ elements, so the sum of sizes of the multisets $\mathcal{H}^{\oplus j}$, $j = 0, \ldots, 4-i$, is at most $(159q^3+1)^{4-i} \leq (160q^3)^{4-i}$. The probability that (16) hits a value in $\mathcal{H}^{\oplus j}$ for some $0 \leq j \leq 4-i$ is at most $(160q^3)^{4-i}/2^{n-1}$.

By Lemma 39 (a), at most $12q^3$ paths are triggered in the query cycle, so $s \leq 12q^3$. There are at most s^i ways to choose i distinct adapted queries. With a union bound over the possible values of i and the choices of the i adapted queries, the probability that $\mathsf{BadAdapt}$ occurs in the query cycle is at most

$$\sum_{i=1}^{4} s^i \cdot \frac{(160q^3)^{4-i}}{2^{n-1}} \leq \sum_{i=1}^{4} s \cdot \frac{(12q^3)^{i-1}(160q^3)^{4-i}}{2^{n-1}} \leq 8.9 \times 10^6 \cdot \frac{sq^9}{2^n} \qquad \square$$

Theorem 44. *An execution of* G_2 *is good with probability at least*

$$1 - 4.2 \times 10^8 q^{12}/2^n.$$

Proof. Consider an execution of G_2. We compute the probability that a bad event occurs *for the first time* in each query cycle or cipher query cycle. In this proof, we let the execution terminate right after any bad event occurs.

For a cipher query cycle, only BadIC can occur, whose probability is at most $1116q^5/2^n$ by Lemma 41.

Now we consider the ith query cycle. Let $\mathcal{R}_i$ be the set of 2chains satisfying the condition of being triggered as defined in Corollary 1, and let s_i be the size of $\mathcal{R}_i$. Recall that $\mathcal{R}_i$ and s_i are determined at the beginning of the query cycle. Since the execution terminates whenever a bad event occurs, only 2chains in $\mathcal{R}_i$ can be triggered. This implies that at most s_i 2chains are triggered and hence at most s_i cipher queries are made during the query cycle. With a union bound over the cipher queries and by Lemma 41, BadIC occurs with probability at most $1116 s_i q^5/2^n$.

Since for any events A and B we have $\Pr[A|B] \geq \Pr[A \wedge B]$, by Lemmas 42 and 43 and using a union bound, the probability that BadIC doesn't occur but $\mathsf{BadPerm}$ or $\mathsf{BadAdapt}$ occurs in the query cycle is at most

$$1.84 \times 10^7 \cdot (s_i + 1) q^9/2^n + 8.9 \times 10^6 \cdot s_i q^9/2^n.$$

Now we consider a G_2-execution. The distinguisher makes q permutation queries to each position and q cipher queries, so there are $5q$ query cycles and q cipher queries cycles in total. The probability that no bad event occurs during the execution is

$$1 - \Big(\frac{1116q^5}{2^n} q + \frac{1116q^5}{2^n} \sum_{i=1}^{5q} s_i + \frac{1.84 \times 10^7 q^9}{2^n} \sum_{i=1}^{5q} (s_i + 1) + \frac{8.9 \times 10^6 q^9}{2^n} \sum_{i=1}^{5q} s_i\Big) \tag{17}$$

where the four terms in the parentheses correspond to the probability of BadIC in cipher query cycles, the probabilities of BadIC, $\mathsf{BadPerm}$ and $\mathsf{BadAdapt}$ in query cycles respectively.

By Lemma 39, as long as no bad event occurs before the current query cycle and BadIC doesn't occur in the current query cycle, at most $12q^3$ 2chains are triggered; this holds even if any bad event occurs since by assumption the execution terminates right after a bad event occurs.[16] Thus we have

$$\sum_{i=1}^{5q} s_i \leq 12q^3 \tag{18}$$

and the lemma follows by combining (17) and (18). □

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 45. *The probability that a G_2-execution is good and is identical to the G_1-execution with the same random tapes is at least $1 - 4.3 \times 10^8 q^{12}/2^n$, where the probability is taken over the random choice of the tapes.*

[16] If the execution terminates early, we let $s_i := 0$ for the subsequent query cycles. The upper bound (17) still holds.

Proof. The bound trivially holds if $13q^3 > 2^{n-2}$, so we assume $13q^3 \leq 2^{n-2}$.

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 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 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 $\{0,1\}^n \setminus \{y \,|\, y = T(k, x) \text{ for some } x\}$. By Lemma 33, 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 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 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 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 31 and 34, 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

$$4.2 \times 10^8 q^{12}/2^n + 2704q^8/2^n \leq 4.3 \times 10^8 q^{12}/2^n$$

where the probability of bad execution is upper bounded by Theorem 44. $\square$

Lemma 46. *We have*

$$\Delta_{\mathcal{D}}(\mathsf{G}_1, \mathsf{G}_2) \leq 4.3 \times 10^8 q^{12}/2^n.$$

Proof. Since $\mathcal{D}$ outputs the same value in identical executions of G_1 and G_2, this is a direct corollary of Lemma 45. □

Theorem 47. *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 35, 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 45, 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.[17] 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 35, 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 $y_5 := \text{Enc}(y_1 \oplus x_2, x_1)$, 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 $\text{Enc}(k, x_1) = y_5$ for some key k. To handle 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 $\text{Dec}(y_4 \oplus x_5, y_5) = x_1$, and $(5, -, y_5)$ is mapped to a set of $(x_1, x_2) \in P_1 \times P_2$ such that $\text{Enc}(y_1 \oplus x_2, x_1) = y_5$. 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 $y_5 := \text{Enc}(y_1 \oplus x_2, x_1)$. By Lemmas 31 and 34, 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 32), so handling the triggered 2chains uses

[17] 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.

$O(q^3)$ 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 35 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 45) 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 cipher 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 27. A distinguisher $\mathcal{D}$ *completes all paths* if at the end of every non-aborting execution, $\mathcal{D}$ has made queries $\text{Query}(i, +, x_i) = y_i$ or $\text{Query}(i, -, y_i) = x_i$ 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 $\text{Enc}(k, x_1) = y'_5$ or $\text{Dec}(k, y'_5) = x_1$.[18]

In the rest of this section, we consider a fixed deterministic distinguisher $\mathcal{D}$ that completes all paths. The definitions and lemmas that only apply to distinguishers that complete all paths will be marked with (*). In Lemma 57, we will get rid of this extra assumption and generalize the result to arbitrary distinguishers using the following observation:

Lemma 48. *Given an arbitrary distinguisher $\mathcal{D}$ with q queries in each position (as described in Theorem 3), there exists an equivalent distinguisher $\mathcal{D}'$ with $2q$ queries in each position that completes all paths. $\mathcal{D}'$ is equivalent with $\mathcal{D}$ in the sense that in the executions (of G_2 or G_4) with the same random tapes, $\mathcal{D}'$ always outputs the same value as $\mathcal{D}$.*

Proof. We define $\mathcal{D}'$ as follows: 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 $\text{Query}(i, +, x_i)$ to "complete the path" as in Definition 27. 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. □

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

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 28. 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 $\tilde{\mathsf{ic}}, \tilde{\mathsf{ic}}^{-1} : \{0,1\}^{2n} \to \{0,1\}^n \cup \{\perp\}$, such that $\tilde{\mathsf{ic}}^{-1}(k, \tilde{\mathsf{ic}}(k,u)) = u$ for all k, u such that $\tilde{\mathsf{ic}}(k,u) \neq \perp$, and such that $\tilde{\mathsf{ic}}(k, \tilde{\mathsf{ic}}^{-1}(k,v)) = v$ for all k, v such that $\tilde{\mathsf{ic}}^{-1}(k,v) \neq \perp$.

We will use *partial random 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 can 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 29. 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 $\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \tilde{\mathsf{ic}}$ consisting of entries of the corresponding tapes that are accessed[19] at some point during the execution.

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 obtainable in executions with $\mathcal{D}$. Let FP_2 and FP_3 denote the set of obtainable footprints in G_2 and G_3 respectively.

Also note that an execution can be recovered given its footprint, since the tape entries not in the footprint are never used during the execution. This implies that executions with the same footprint are identical; in particular, if an execution is good (resp. non-aborting), then executions with the same footprint must also be good (resp. non-aborting).

Definition 30. 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 $\tilde{\mathsf{ic}}$ if $\mathsf{ic}(k,u) = \tilde{\mathsf{ic}}(k,u)$ for all k, u such that $\tilde{\mathsf{ic}}(k,u) \neq \perp$.

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 49. *For $i = 2, 3$ and 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.*

[19] 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 $\tilde{\mathsf{ic}}$.

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. □

For $i = 2, 3$, 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 hence

$$\Pr_{\mathsf{G}_i}[\mathcal{S}] = \sum_{\omega \in \mathcal{S}} \Pr_{\mathsf{G}_i}[\omega]. \tag{19}$$

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 49. Let $|\tilde{p}_i|$ (resp. $|\tilde{\mathsf{ic}}|$) denote the number of non-$\perp$ entries in $\tilde{p}_i$ (resp. $\tilde{\mathsf{ic}}$), and let $|\tilde{\mathsf{ic}}(k)|$ denote the number of non-$\perp$ entries of the form $(k, *)$ in $\tilde{\mathsf{ic}}$. For $\omega = (\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \tilde{\mathsf{ic}}) \in \mathsf{FP}_2$, we have

$$\Pr_{\mathsf{G}_2}[\omega] = \Big(\prod_{i=1}^{5} \prod_{\ell=0}^{|\tilde{p}_i|-1} \frac{1}{2^n - \ell}\Big)\Big(\prod_{k \in \{0,1\}^n} \prod_{\ell=0}^{|\tilde{\mathsf{ic}}(k)|-1} \frac{1}{2^n - \ell}\Big), \tag{20}$$

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

$$\Pr_{\mathsf{G}_3}[\omega] = \prod_{i=1}^{5} \prod_{\ell=0}^{|\tilde{t}_i|-1} \frac{1}{2^n - \ell}. \tag{21}$$

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. Since an execution can be recovered from the footprint and executions with the same footprint are identical, any 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 31. (*) We define the injection $\zeta : \mathsf{FP}_2^* \to \mathsf{FP}_3^*$ as follows: for $\omega = (\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \tilde{\mathsf{ic}}) \in \mathsf{FP}_2^*$, let $\zeta(\omega) = (\tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_5)$ where

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

in which P_i refers to the state of the table P_i 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 50. (*) *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) and since good executions don't abort, a 2chain that has been triggered must belong to a complete path at the end of the execution. We only need to prove that for each $T(k, x_1) = y_5$, $(5, 1, y_5, x_1, k)$ belongs to the same complete path as a triggered 2chain (we say 2chains that belong to the same complete path are *equivalent* for short).

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 27), 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 18 and since query cycles in good executions must be safe (cf. Lemma 22), C belongs to a complete path which was triggered during the query cycle. The lemma then follows from 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 observe from the pseudocode that $(5, 1, y_5, x_1, k)$ is equivalent to the triggered 2chain that is being handled when the call occurs. □

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 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 51. (*) *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 $\omega = (\tilde{p}_1, \ldots, \tilde{p}_5, \tilde{\mathsf{ic}})$ 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 50, the 2chain $(5, 1, y_5, x_1, k)$ is complete, i.e., there exists a complete path consisting of queries (i, x_i, y_i), $i = 1, 2, 3, 4, 5$, satisfying (6). 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 $\zeta(\omega)$. The above proof already shows that all tape entries read by the simulator or by the cipher oracle (in EM and EM^{-1}) are in $\zeta(\omega)$. We only need to prove every entry $\tilde{t}(x_i) = y_i$ in $\zeta(\omega)$ has been read during the G_3-execution.

By Definition 31, $\tilde{t}(x_i) = y_i$ corresponds to an entry $P_i(x_i) = y_i$ at the end of the execution (this is true for the G_3-execution as well since it is identical to the G_2-execution), 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 consider the complete path in which the query is adapted: 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 $j = 1, 2, 3, 4$. In the call to $\mathrm{EM}(k, x_1)$ or $\mathrm{EM}^{-1}(k, y_5)$ where $T(k, x_1) = y_5$ was assigned, the entries $t_j(x_j) = y_j$ for $j = 1, 2, 3, 4, 5$ are read (note that $t_i(x_i) = y_i$ is among these entries). $\square$

Lemma 52. (*) *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 23), Lemma 51 implies that the execution of G_3 with footprint $\zeta(\omega)$ is non-aborting and hence $\zeta(\omega) \in \mathsf{FP}_3^*$.

Lemma 51 also implies that the ζ is injective: given 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 $\zeta(\omega)$. Then $\omega = (\tilde{p}_1, \dots, \tilde{p}_5, \tilde{\mathsf{ic}})$ can be uniquely reconstructed as follows: $\tilde{p}_i$ contains entries that are read by the simulator during the execution; $\tilde{\mathsf{ic}}$ contains cipher queries that are issued by the distinguisher or the simulator. □

Lemma 53. (*) *For $\omega = (\tilde{p}_1, \dots, \tilde{p}_5, \tilde{\mathsf{ic}}) \in \mathsf{FP}_2^*$ and $\zeta(\omega) = (\tilde{t}_1, \dots, \tilde{t}_5)$, we have*

$$\sum_{i=1}^{5} |\tilde{t}_i| = \sum_{i=1}^{5} |\tilde{p}_i| + |\tilde{\mathsf{ic}}|. \tag{22}$$

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 31). 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 $\tilde{\mathsf{ic}}$. Note that $\tilde{\mathsf{ic}}$ contains the same queries as the table T at the end of the execution. By Lemma 50, 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 26 the triggered 2chains are not equivalent. Thus the triggered 2chains belong to distinct complete paths, each containing a distinct query in T. Thus there is a one-one correspondence between triggered 2chains and entries in T, implying $|T| = |\tilde{\mathsf{ic}}|$ equals the number of triggered 2chains. □

Lemma 54. (*) *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 $13q^3 \geq 2^n$, so we can assume $13q^3 < 2^n$ in the following proof.

Let $\omega = (\tilde{p}_1, \dots, \tilde{p}_5, \tilde{\mathsf{ic}})$ and $\zeta(\omega) = (\tilde{t}_1, \dots, \tilde{t}_5)$. Consider the G_2-execution with footprint ω. The execution is good since $\omega \in \mathsf{FP}_2^*$. By Lemma 33, we have $|T| \leq 12q^3 + q \leq 13q^3$ at the end of the execution. Since $\tilde{\mathsf{ic}}$ contains the entries of T at the end of the execution and for any $k \in \{0,1\}^n$, $\tilde{\mathsf{ic}}(k)$ is a subset of $\tilde{\mathsf{ic}}$, we have

$$|\tilde{\mathsf{ic}}(k)| \leq |\tilde{\mathsf{ic}}| \leq 13q^3. \tag{23}$$

P_i contains every entry of $\tilde{p}_i$ because they are read in ReadTape, so at the end of the execution we have $|\tilde{p}_i| \leq |P_i| = |\tilde{t}_i|$. By (20) and (21), we have

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

where the first inequality uses (23), the second equality uses (22) and the fact that $\sum_k |\tilde{\mathsf{ic}}(k)| = |\tilde{\mathsf{ic}}|$, and the second inequality uses (23) and the assumption that $13q^3 < 2^n$. □

Lemma 55. (*) *Consider a G_3-execution with footprint $\zeta(\omega)$ for $\omega \in \mathsf{FP}_2^*$, and the G_4-execution with the same random tapes. The views of the distinguisher $\mathcal{D}$ in the two executions are identical. In particular, $\mathcal{D}$ outputs the same value in the two executions.*

Proof. Let $\zeta(\omega) = (\tilde{t}_1, \ldots, \tilde{t}_5)$ and let $\mathcal{T} = (t_1, \ldots, t_5)$ be the set of tapes used in the two executions.

We prove the views of $\mathcal{D}$ are identical by running the two executions in parallel, and prove by induction that $\mathcal{D}$ always receives the same answer in the two executions. Since $\mathcal{D}$ is deterministic, its behaviors (i.e., its queries and its output) are identical in the two executions as long as all the previous queries have the same answers.

The cipher oracle Enc/Dec is the same in G_3 and G_4, whose answers only depend on the tapes. The two executions use the same tapes, so Enc/Dec returns the same answers in the two executions.

Then we consider a distinguisher's call to Query$(i, +, x_i)$ (Query$(i, -, y_i)$ is symmetric). In the G_3-execution, SimQuery$(i, +, x_i)$ is called, which eventually returns the value of $y_i = P_i(x_i)$. The table P_i is never overwritten, so $P_i(x_i) = y_i$ at the end of the execution. Since the G_3-execution is identical to the G_2-execution with footprint ω (cf. Lemma 51), we have $P_i(x_i) = y_i$ at the end of the G_2-execution and hence $\tilde{t}_i(x_i) = y_i$ (cf. Definition 31).

In the G_4-execution, the call to Query$(i, +, x_i)$ returns $t_i(x_i)$. Since t_i is compatible with $\tilde{t}_i$, we have $t_i(x_i) = y_i$. So the call returns the same value in the two executions. □

Lemma 56. (*) *If the distinguisher $\mathcal{D}$ completes all paths, we have*

$$\Delta_{\mathcal{D}}(\mathsf{G}_2, \mathsf{G}_4) \leq 4.2 \times 10^8 q^{12}/2^n + 169q^6/2^n$$

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 51, for $\omega \in \mathsf{FP}_2^*(1)$ we have $\zeta(\omega) \in \mathsf{FP}_3^*(1)$. Moreover, by Lemma 55, 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

$$\begin{aligned}\sum_{\omega \in \mathsf{FP}_2^*(1)} \Pr_{\mathsf{G}_3}[\zeta(\omega)] &\geq \sum_{\omega \in \mathsf{FP}_2^*(1)} \Pr_{\mathsf{G}_2}[\omega] \cdot \big(1 - \frac{169q^6}{2^n}\big)\\ &= \Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)] \cdot \big(1 - \frac{169q^6}{2^n}\big)\\ &\geq \Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)] - \frac{169q^6}{2^n} \end{aligned} \tag{24}$$

where the first inequality uses Lemma 54, where the equality uses (19), 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 (24),

$$\begin{aligned}\Delta_{\mathcal{D}}(\mathsf{G}_2, \mathsf{G}_4) &\leq \Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)] + \big(1 - \Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*]\big) - \big(\Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)] - \frac{169q^6}{2^n}\big)\\ &= 1 - \Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*] + 169q^6/2^n\\ &\leq 4.2 \times 10^8 q^{12}/2^n + 169q^6/2^n\end{aligned}$$

where the last inequality uses Theorem 44. □

Lemma 57. *For an arbitrary distinguisher $\mathcal{D}$ with q queries in each position, we have*

$$\Delta_{\mathcal{D}}(\mathsf{G}_2, \mathsf{G}_4) \leq 1.73 \times 10^{12} q^{12}/2^n.$$

Proof. By Lemma 48, there exists a distinguisher $\mathcal{D}'$ equivalent to $\mathcal{D}$ that completes all paths and makes at most $2q$ queries in each position. Since $\mathcal{D}'$ is subject to Lemma 56, we have

$$\begin{aligned}\Delta_{\mathcal{D}}(\mathsf{G}_2, \mathsf{G}_4) = \Delta_{\mathcal{D}'}(\mathsf{G}_2, \mathsf{G}_4) &\leq 4.2 \times 10^8 \cdot (2q)^{12}/2^n + 169(2q)^6/2^n\\ &\leq 1.73 \times 10^{12} q^{12}/2^n.\end{aligned}$$ □

5.7 Indistinguishability of G_1 and G_4

Theorem 58. *Any distinguisher with q queries cannot distinguish G_1 from G_4 with advantage more than $2 \times 10^{12} q^{12}/2^n$.*

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

$$\begin{aligned}\Delta_{\mathcal{D}}(\mathsf{G}_1, \mathsf{G}_4) &= \Delta_{\mathcal{D}}(\mathsf{G}_1, \mathsf{G}_2) + \Delta_{\mathcal{D}}(\mathsf{G}_2, \mathsf{G}_4) \\ &\leq 4.3 \times 10^8 q^{12}/2^n + 1.73 \times 10^{12} q^{12}/2^n \\ &\leq 2 \times 10^{12} q^{12}/2^n\end{aligned}$$

where the first inequality uses Lemmas 46 and 57. □

Acknowledgments

We thank Dana Dachman-Soled and Jonathan Katz for discussions that led to the termination argument used in this work. The first author was supported by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003. The second author was partially supported by the French Agence Nationale de la Recherche through the BRUTUS project under Contract ANR-14-CE28-0015. The third author was supported by National Natural Science Foundation of China Grant 20131351464. 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] Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the Indifferentiability of Key-Alternating Ciphers. In Ran Canetti and Juan A. Garay, editors, *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] Elena Andreeva, Andrey Bogdanov, and Bart Mennink. Towards Understanding the Known-Key Security of Block Ciphers. In Shiho Moriai, editor, *Fast Software Encryption - FSE 2013*, volume 8424 of *LNCS*, pages 348–366. Springer, 2013.

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

[4] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated Key Exchange Secure against Dictionary Attacks. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.

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

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

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

[8] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009.

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

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

[11] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, François-Xavier Standaert, John P. Steinberger, and Elmar Tischhauser. Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations - (Extended Abstract). In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 45–62. Springer, 2012.

[12] Dan Boneh and Victor Shoup. A Graduate Course in Applied Cryptography. Available at http://toc.cryptobook.us.

[13] Shan Chen, Rodolphe Lampe, Jooyoung Lee, Yannick Seurin, and John P. Steinberger. Minimizing the Two-Round Even-Mansour Cipher. In Juan A. Garay and Rosario Gennaro, editors, *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] Shan Chen and John Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *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] Benoît Cogliati and Yannick Seurin. On the Provable Security of the Iterated Even-Mansour Cipher against Related-Key and Chosen-Key Attacks. In Elisabeth Oswald and Marc Fischlin, editors, *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] Benoît Cogliati and Yannick Seurin. Strengthening the Known-Key Security Notion for Block Ciphers. In Thomas Peyrin, editor, *Fast Software Encryption - FSE 2016*, volume 9783 of *LNCS*, pages 494–513. Springer, 2016.

[17] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005.

[18] Jean-Sébastien Coron, Thomas Holenstein, Robin Künzler, Jacques Patarin, Yannick Seurin, and Stefano Tessaro. How to Build an Ideal Cipher: The Indifferentiability of the Feistel Construction. *J. Cryptology*, 29(1):61–114, 2016.

[19] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *LNCS*, pages 1–20. Springer, 2008.

[20] Yuanxi Dai and John P. Steinberger. Indifferentiability of 8-Round Feistel Networks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology -*

CRYPTO 2016 (Proceedings, Part I), volume 9814 of *LNCS*, pages 95–120. Springer, 2016. Full version available at http://eprint.iacr.org/2015/1069.

[21] Grégory Demay, Peter Gazi, Martin Hirt, and Ueli Maurer. Resource-Restricted Indifferentiability. In Thomas Johansson and Phong Q. Nguyen, editors, *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] Anand Desai. The Security of All-or-Nothing Encryption: Protecting against Exhaustive Key Search. In Mihir Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *LNCS*, pages 359–375. Springer, 2000.

[23] Yevgeniy Dodis, Martijn Stam, John P. Steinberger, and Tianren Liu. Indifferentiability of Confusion-Diffusion Networks. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 (Proceedings, Part II)*, volume 9666 of *LNCS*, pages 679–704. Springer, 2016.

[24] Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-Mansour Scheme Revisited. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 336–354. Springer, 2012.

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

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

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

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

[29] Chun Guo and Dongdai 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] Chun Guo and Dongdai 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] Viet Tung Hoang and Stefano Tessaro. Key-Alternating Ciphers and Key-Length Extension: Exact Bounds and Multi-user Security. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 (Proceedings, Part I)*, volume 9814 of *LNCS*, pages 3–32. Springer, 2016.

[32] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The Equivalence of the Random Oracle Model and the Ideal Cipher Model, Revisited. In Lance Fortnow and Salil P. Vadhan, editors, *Symposium on Theory of Computing - STOC 2011*, pages 89–98. ACM, 2011. Full version available at http://arxiv.org/abs/1011.1264.

[33] Tetsu Iwata and Tadayoshi Kohno. New Security Proofs for the 3GPP Confidentiality and Integrity Algorithms. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption - FSE 2004*, volume 3017 of *LNCS*, pages 427–445. Springer, 2004.

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

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

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

[37] Rodolphe Lampe, Jacques Patarin, and Yannick Seurin. An Asymptotically Tight Security Analysis of the Iterated Even-Mansour Cipher. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 278–295. Springer, 2012.

[38] Rodolphe Lampe and Yannick Seurin. How to Construct an Ideal Cipher from a Small Set of Public Permutations. In Kazue Sako and Palash Sarkar, editors, *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] Avradip Mandal, Jacques Patarin, and Yannick Seurin. On the Public Indifferentiability and Correlation Intractability of the 6-Round Feistel Construction. In Ronald Cramer, editor, *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] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *Theory of Cryptography Conference- TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, 2004.

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

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

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

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

[45] John 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] Robert 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.