ABSTRACT

Title of dissertation: IMPROVING THE ROUND COMPLEXITY OF IDEAL-CIPHER CONSTRUCTIONS

Aishwarya Thiruvengadam, Doctor of Philosophy, 2017

Dissertation directed by: Professor Jonathan Katz
Department of Computer Science

Block ciphers are an essential ingredient of modern cryptography. They are widely used as building blocks in many cryptographic constructions such as encryption schemes, hash functions etc. The security of block ciphers is not currently known to reduce to well-studied, easily formulated, computational problems. Nevertheless, modern block-cipher constructions are far from ad-hoc, and a strong theory for their design has been developed. Two classical paradigms for block cipher design are the Feistel network and the key-alternating cipher (which is encompassed by the popular substitution-permutation network). Both of these paradigms that are iterated structures that involve applications of random-looking functions/permutations over many rounds.

An important area of research is to understand the provable security guarantees offered by these classical design paradigms for block cipher constructions. This can be done using a security notion called indifferentiability which formalizes what it means for a block cipher to be ideal. In particular, this notion allows us to assert the structural robustness of a block cipher design. In this thesis, we apply the in-

differentiability notion to the two classical paradigms mentioned above and improve upon the previously known round complexity in both cases. Specifically, we make the following two contributions:

- We show that a 10-round Feistel network behaves as an ideal block cipher when the keyed round functions are built using a random oracle.

- We show that a 5-round key-alternating cipher (also known as the iterated Even-Mansour construction) with identical round keys behaves as an ideal block cipher when the round permutations are independent, public random permutations.

IMPROVING THE ROUND COMPLEXITY OF IDEAL-CIPHER CONSTRUCTIONS

by

Aishwarya Thiruvengadam

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2017

Advisory Committee:
Professor Jonathan Katz, Chair/Advisor
Professor Dana Dachman-Soled, Co-Chair
Professor Michelle Mazurek
Professor Charalampos Papamanthou
Professor Lawrence Washington

To Dad

தந்தை சொல்மிக்க மந்திரம் இல்லை

Acknowledgements

I would like to thank my advisor, Jonathan Katz, for his support and guidance throughout my PhD. He has given me the independence to grow while being available when needed. Learning from him has been a truly enriching experience and I have benefited immensely from our interactions having always come away with a new perspective. He has also been a source of valuable advice and has helped me navigate various aspects of my career. Jon's views on science, research, and teaching have helped shape my own and I am very grateful for that. His understanding during a difficult personal circumstance made my decision to continue with grad school easier; under different circumstances, I am not sure which path my life would have taken. Jon has been kind, fair, patient, and forgiving, and with his support, I have felt fearless in pursuing my research interests. I am deeply grateful to Jon and feel that I have been extremely fortunate to have him as my advisor.

I would also like to thank my advisor and mentor, Dana Dachman-Soled, for her support and encouragement. Dana has always had her door open for me and I took maximum advantage of it. Some of my most enjoyable experiences in grad school have involved staring at a white board with Dana while reasoning about a problem. My interactions with her made me feel more comfortable in my own skin and have helped me come into my own. She taught me the value of perseverance and has always been willing to engage with me and lead me in new research directions. She has provided valuable insight and advice in various aspects of research and academia. I am extremely grateful for her guidance and friendship and feel very fortunate that my years at UMD overlapped with hers, if only for a few years.

I am grateful to Babis, Larry, and Michelle for serving on my dissertation committee. I am also immensely grateful to all my co-authors and the research community and to all my teachers over the years. Thanks especially to Dana and Jon for collaboration on the work appearing in Chapter 3 and John, Yannick, and Yuanxi for collaboration on the work appearing in Chapter 4. I have also been lucky to share space at MC2 with brilliant faculty, staff and students. My gratitude to them for creating a vibrant atmosphere. I am grateful also to the faculty, students and staff at the CS department for their support, friendship and mentorship. Thanks especially to Jenny and Fatima who made any task seem like a breeze.

I have made some wonderful friends at UMD. Thank you all for being there, through the highs and the lows. Thanks especially to Shweta, Srimathy and Vikas who were incredibly supportive during difficult times. Vikas has been my rock throughout this process while going through and learning from similar experiences.

Lastly, I thank my Mom and sister who have shown incredible love, strength and support. My deepest gratitude to friends and family without whose support I could not be here today. It is humbling to know that there are those to whom I owe a deep debt of gratitude that I can never hope to repay. Finally, thanks to Dad for his immeasurable love and for being my guiding light every step of the way.

Table of Contents

List of Figures

List of Abbreviations

IC	Ideal Cipher
IEM	Iterated Even-Mansour

Chapter 1: Introduction

Block ciphers are fundamental building blocks of cryptography. They enjoy widespread use in various cryptographic constructions such as encryption schemes [4], hash functions [42,50] etc. Block ciphers take a key k and another input x and produce an output y. They are keyed objects that are intended to produce families of pseudorandom permutations [44]. Roughly speaking, a permutation is said to be "pseudorandom" when an efficient attacker interacting with it cannot tell if it is interacting with the object or a perfectly random permutation.

The design of modern block ciphers follows two main approaches [39] both of which rely on Shannon's confusion and diffusion paradigm [32,55]. In this paradigm, the input is first subjected to a *confusion* step where a (pseudo-)random function(s) is applied on part of the input. Then, in the *diffusion* step, the output of the confusion step is "mixed" such that the confusion propagates through the output bits. These two steps together form one *round* and are repeated multiple times resulting in what is expected to be a random-looking output.

Feistel Networks. Following this paradigm, one well-known approach for building practical block ciphers is to use a *Feistel network* [32], an iterated structure in which key-dependent, "random-looking" *round functions* on $\{0,1\}^n$ are applied in a sequence of rounds to yield a permutation on $\{0,1\}^{2n}$. Block ciphers that are based on Feistel networks include DES [49], FEAL [56], MISTY [46] and KASUMI [1]. In analyzing the security that Feistel networks provide, it is useful to consider an information-theoretic setting in which the round functions are instantiated by independent, truly random (keyed) functions. The purpose of such an analysis is to validate the *structural* robustness of the approach. Luby and Rackoff [44] proved that when independent, random round functions are used, a three-round Feistel network is indistinguishable from a random permutation under chosen-plaintext attacks, and a four-round Feistel network (depicted in Fig. 2.1) is indistinguishable from a random permutation under chosen plaintext/ciphertext attacks.

Key-alternating Ciphers/Iterated Even-Mansour Constructions. Another popular approach for block cipher design is by construction of *key-alternating ciphers*. Such block ciphers alternatively apply two types of transformations to the current state: the (usually bitwise) addition of a *secret* round key, and the application of a *public* permutation to the entire block. This is in particular the case of virtually all *Substitution Permutation Networks*, a popular methodology for block cipher construction, such as AES [20]. Other examples of key-alternating ciphers include block ciphers such as Serpent [9] and PRESENT [13]. This class of block ciphers has also been analyzed in an information-theoretic setting by modeling the public permutations as oracles that the adversary can only query in a black-box way (in both directions), each behaving as a perfectly random permutation. Again, this approach allows to assert the nonexistence of *generic attacks*, i.e., attacks not exploiting the particular structure of “concrete” permutations. This approach to analyzing the key-alternating ciphers dates back to Even and Mansour [30] who studied the one-round case. For this reason, such key-alternating cipher constructions with public permutations that can be queried only in a black-box manner are often called the *iterated Even-Mansour (IEM) construction*, which is the terminology we will use in the rest of this thesis.

Indistinguishability. In both the Luby-Rackoff [44] and the Even-Mansour [30] results for Feistel networks and the IEM construction respectively, the security notion considered—namely, *indistinguishability*—is one in which the key of the overall block cipher is unknown to the adversary. Indistinguishability results establish that the block ciphers satisfy the pseudorandomness property, i.e., an efficient adversary cannot tell whether it is interacting with the block cipher for an unknown key or a truly random permutation.

Even though pseudorandomness has been the primary security requirement any block cipher should satisfy, in some cases this property is not enough to establish the security of higher-level cryptosystems where the block cipher is used. For example, the security of some real-world authenticated encryption protocols such as 3GPP confidentiality and integrity protocols f8 and f9 [38] rely on the stronger block cipher security notion of *indistinguishability under related-key attacks* [5,8]. Another context where this problem arises is in block-cipher based hash functions [42,50]. In such cases, the adversary controls both the input and the key of the block cipher, and hence can exploit “known-key” or “chosen-key” attacks [10,41] 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)* [55], i.e., an oracle where each key defines an independent, random permutation. Perhaps not surprisingly, this view has turned out to be very fruitful for proving the security of constructions based on a block cipher when the pseudorandomness assumption is not enough [6,7,12,25,34,40,48,57], by using a model known as the *ideal cipher model*. However, this ultimately remains a heuristic approach, as one can construct (contrived) schemes that are secure in the ideal cipher model, but insecure for any concrete instantiation of the block cipher [11]. This means that there is no hope to formalize (let alone prove) what it means for a concrete block cipher to "behave" as an ideal cipher and that the strength of a concrete block cipher in this respect should ultimately be evaluated through cryptanalysis.

Indifferentiability. This does not mean that the provable security approach has nothing to offer regarding how to design something close to an ideal cipher. Indeed, the indifferentiability framework, introduced by Maurer et al. [47] (and popularized by Coron et al. [16]), helps formalize the notion of what it means for a block cipher to behave like an ideal cipher. More generally, it allows to assess whether a construction of some target primitive A (e.g., a block cipher) from some lower-level ideal primitive B (e.g., for the IEM construction, a small number of random permutations or for the Feistel network, a small number of random functions) is "structurally close" to the ideal version of A (e.g., an ideal cipher). Furthermore, the notion of indifferentiability is accompanied by a powerful composition theorem [47] which ensures that a large class of protocols that are provably secure when used with the ideal-A primitive, remain secure in the ideal-B model — as long as one uses a construction of A from ideal-B that is indifferentiable from ideal-A (see [24, 52] for restrictions on the applicability of the theorem).

Proving indifferentiability is more complex than proving indistinguishability: to prove indifferentiability from an ideal cipher IC of a block cipher construction $\mathcal{C}$ that relies on an ideal primitive $\mathbf{O}$, one must exhibit a *simulator* $\mathcal{S}$ such that the view of any distinguisher interacting with $(\mathcal{C}^{\mathbf{O}}, \mathbf{O})$ is indistinguishable from its view when interacting with $(\mathsf{IC}, \mathcal{S}^{\mathsf{IC}})$. Once a tentative simulator has been determined, the indifferentiability proof usually entails two technical challenges: on the one hand, proving that the simulator is never trapped into an inconsistency, and on the other hand, proving that it runs in polynomial time. Finding the right balance between these two requirements is at the heart of the design of a suitable simulator.

1.1 Contributions

Indifferentiability of Feistel Networks. In a landmark result building on [18, 37, 53], Coron et al. [17] proved that when using independent, random round functions, a 14-round Feistel network is indifferentiable from a random permutation. This suffices to show that a 14-round Feistel network with keyed round functions built using a random oracle is indifferentiable from an ideal cipher. The key question left open by the work of Coron et al. is one of *efficiency*: how many rounds are needed in order for indifferentiability to hold? It is known from prior work [17] that 5 rounds are not sufficient, while (as we have just noted) 14 rounds are. In Chapter 3, we narrow this gap and show that a 10-round Feistel network is indifferentiable from an ideal cipher.[1] The results in this chapter are based on work published in Eurocrypt 2016 [19].

Indifferentiability of IEM. Andreeva et al. [2] showed that the 5-round IEM construction with an idealized key-schedule (i.e., the function(s) mapping the master key onto the round key(s) are modeled as random oracles) is indifferentiable from an ideal cipher. Lampe and Seurin [43] showed that the 12-round IEM construction with the trivial key-schedule, i.e., in which all round keys are equal, is also indifferentiable from an ideal cipher. In both settings, the question of the exact number of rounds needed to make the IEM construction indifferentiable from an ideal cipher remained open. In Chapter 4, considering the case of non-idealized key-schedules (i.e., key derivation does not make use of an ideal primitive), we show that a 5-round IEM construction is indifferentiable from an ideal cipher. The results in this chapter are based on work to be published in Crypto 2017 [21].

Summary. In this thesis, we analyze two popular block cipher design methodologies, namely the Feistel network and the iterated Even-Mansour construction, and further our understanding on the number of rounds required for such constructions to behave like an ideal cipher under certain assumptions.

[1] A proof claiming that the 10-round Feistel is indifferentiable from an ideal cipher was published in [53] but the authors later found a distinguishing attack [54].

Chapter 2: Preliminaries

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

The Feistel Network. The r-round Feistel construction, given access to functions $\mathbf{F} = (F_1, \ldots, F_r)$ from $\{0,1\}^n$ to $\{0,1\}^n$, is defined as follows. Let (L_{i-1}, R_{i-1}) be the input to the i-th round, with (L_0, R_0) denoting the initial input. Then, the output (L_i, R_i) of the i-th round of the construction is given by $L_i := R_{i-1}$ and $R_i := L_{i-1} \oplus F_i(R_{i-1})$. So, for a r-round Feistel, if the $2n$-bit input is (L_0, R_0), then the output is given by (L_r, R_r). A Feistel network with $r = 4$ is illustrated in Figure 2.1.

The Iterated Even-Mansour Construction. Let $\mathbf{g} = (g_0, \ldots, g_r)$ be a $(r+1)$-tuple of functions from $\{0,1\}^n$ to $\{0,1\}^n$ specifying the key schedule of the IEM. Then, given access to $\mathbf{P} = (P_1, \ldots, P_r)$ of permutations over $\{0,1\}^n$, the r-round iterated Even-Mansour construction (denoted $\mathsf{EM}^{\mathbf{P}}$) maps an input $x \in \{0,1\}^n$ and

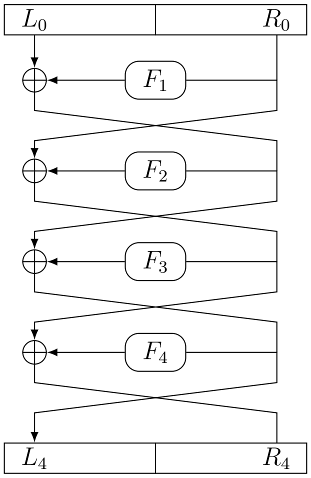

Figure 2.1: The 4-round Feistel network.

a key $k \in \{0,1\}^n$ to the output defined by

$$\mathsf{EM}^{\mathbf{P}}(k,x) = g_r(k) \oplus P_r(g_{r-1}(k) \oplus P_{r-1}(\cdots P_2(g_1(k) \oplus P_1(g_0(k) \oplus x)) \cdots)).$$

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

We observe that the first and the last key additions do not play any role for indifferentiability. This is because in the indifferentiability setting the key is just a "public" input to the construction. Hence, in Chapter 4, we focus on the slight variant of the trivial key-schedule where $g_0 = g_r = 0$ (see Fig. 2.2), but our results carry over directly to the "standard" trivial key-schedule (and more generally to any non-idealized key-schedule where the g_i's are permutations of $\{0,1\}^n$).

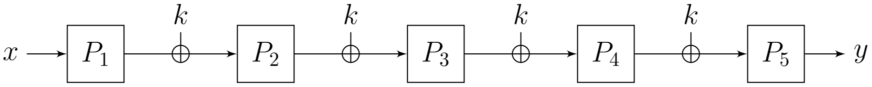

Figure 2.2: The 5-round IEM with independent permutations and identical round keys where the first and last round key additions are omitted.

Notation. Throughout, n will denote the block length of the underlying primitive used for the block cipher construction — in the case of the Feistel network, n denotes the block length of the round functions F_i, and for the IEM construction, the block length of the permutations P_i — and will play the role of the 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 algorithm $\mathcal{D}$ with oracle access to a finite list of oracles $(O_1, O_2, \ldots)$ and that outputs a single bit b, which we denote $\mathcal{D}^{O_1, O_2, \ldots} = b$.

Indifferentiability. We present the standard definition of indifferentiability below. (For clarity, we state it directly in the context of indifferentiability from an ideal cipher for a block cipher $\mathcal{C}$ instantiated from a primitive $\mathbf{O}$.) This definition is obtained by adapting the definition used by Coron et al. [17] based on the original definition of Maurer, Renner, and Holenstein [47].

Definition 1. *Let $\mathcal{C}$ be a construction that, for any n, accesses oracles $\mathbf{O} = (O_1, \ldots, O_r)$ over $\{0,1\}^n$ and implements a block cipher with key space $\{0,1\}^\kappa$ and message space $\{0,1\}^m$. We say that $\mathcal{C}$ is* indifferentiable from an ideal cipher *if there exists a simulator $\mathcal{S}$ and a polynomial t such that for all distinguishers $\mathcal{D}$ making at most $q = \mathsf{poly}(n)$ queries, $\mathcal{S}$ runs in time $t(q)$ and*

$$|\Pr[\mathcal{D}^{\mathcal{C}^{\mathbf{O}}, \mathbf{O}}(1^n) = 1] - \Pr[\mathcal{D}^{\mathsf{IC}, \mathcal{S}^{\mathsf{IC}}}(1^n) = 1]|$$

is negligible, where IC is an ideal cipher with key space $\{0,1\}^\kappa$ and message space $\{0,1\}^m$.

Note that Definition 1 allows the simulator to depend on the number of queries q. In fact, the simulators that we present do not depend on q, but these simulators are efficient only with high probability, as will become clear in the proofs. However, we can modify these simulators such that they can use the knowledge of q (as allowed in Definition 1) and abort whenever their runtime exceeds $t(q)$, thus ensuring that the simulator is efficient with probability 1.

Chapter 3: Indifferentiability of the 10-Round Feistel Network

As mentioned in Chapter 1, in the context of Feistel networks, it is known (see [17]) that one can simplify the problem of indifferentiability from an ideal cipher and focus on indifferentiability of the Feistel network when using independent, random, *unkeyed* round functions from a *public random permutation*; an ideal cipher can then be obtained by keying the round functions. Coron et al. [17] proved that a 14-round Feistel network using random, independent, round functions is indifferentiable from a random permutation. Left unresolved is the best possible efficiency of the transformation. In this chapter, we improve upon the result of Coron et al. [17] and show that 10 rounds suffice.

3.1 Overview

We first describe the proof structure used by Coron et al., and then describe how our proof differs.

3.1.1 The Techniques of Coron et al.

Consider a naive simulator for an r-round Feistel construction, which responds to distinguisher queries to each of the round functions $F_1, \ldots, F_r$, by always returning a uniformly random value. Unfortunately, there is a simple distinguisher who can distinguish oracle access to $(\mathsf{Feistel}_r^F, F)$ from oracle access to $(P, \mathcal{S}^P)$: The distinguisher will query (x_0, x_1) to the first oracle, receiving (x_r, x_{r+1}) in return and will use oracle access to the second oracle to evaluate the r-round Feistel and compute (x'_r, x'_{r+1}) on its own, creating a *chain* of queries, $(x_1, \ldots, x'_r)$. Note that in the first case $(x_r, x_{r+1}) = (x'_r, x'_{r+1})$ with probability 1, while in the second case the probability that $(x_r, x_{r+1}) = (x'_r, x'_{r+1})$ is negligible, so security is broken.

The following is an approach to fixing the above attack, which essentially gives the very high-level intuition for how a successful simulator works: If the simulator can find out the value of $P(x_0, x_1) = (x_r, x_{r+1})$ before the distinguisher queries the entire chain, then the simulator can assign values for the remaining queries $F_i(x_i)$, conditioned on the restriction $\mathsf{Feistel}_r^F(x_0, x_1) = (x_r, x_{r+1})$. More specifically, if there are two consecutive rounds $(i, i+1)$, where $i \in \{1, \ldots, r-1\}$, which have not yet been queried, the simulator can adapt its assignments to $F_i(x_i)$, $F_{i+1}(x_{i+1})$ to be consistent with $P(x_0, x_1) = (x_r, x_{r+1})$. When the simulator adapts the assignment of $F_i(x_i)$ to be consistent with a constraint $P(x_0, x_1) = (x_r, x_{r+1})$, we say that this value of $F_i(x_i)$ has been assigned via *ForceVal*. We next discuss further details of the Coron et al. [17] construction.

Partial chain detection and preemptive completion. To allow the simulator to preemptively discover $P(x_0, x_1) = (x_r, x_{r+1})$, Coron et al. fix two "detect zones" which are sets of consecutive rounds $\{1, 2, 13, 14\}$, $\{7, 8\}$. Each time the simulator assigns a value to $F_i(x_i)$, it also checks whether there exists a tuple of the form

$(x_1, x_2, x_{13}, x_{14})$ such that (1) $F_1(x_1)$, $F_2(x_2)$, $F_{13}(x_{13})$, $F_{14}(x_{14})$ have all been assigned and (2) $P(F_1(x_1) \oplus x_2, x_1) = (x_{14}, F_{13}(x_{13}) \oplus x_{14})$; or whether there exists a tuple of the form (x_7, x_8) such that $F_7(x_7)$ and $F_8(x_8)$ have both been assigned. A pair of consecutive round values (x_k, x_{k+1}) is referred to as a "partial chain" and when a new partial chain is detected according to the detect zones described above it is "enqueued for completion" and will later be dequeued and preemptively completed. When a partial chain is detected due to a detect zone that includes both x_1 and x_r, we say it is a "wraparound" chain. Note that preemptive completion of a chain can cause new chains to be detected and these will then be enqueued for completion. This means that in order to prove indifferentiability, it is necessary to argue that for x_i that fall on multiple completed chains, all restrictions on the assignment of $F_i(x_i)$ can be *simultaneously* satisfied. In particular the "undesired case" will be when some assignment $F_i(x_i)$ must be adapted via a ForceVal assignment, but an assignment to $F_i(x_i)$ has already been made previously. If such a case occurs, we say the value at an adapt position has been "overwritten." It turns out that to prove indifferentiability, it is sufficient to prove that this occurs with negligible probability.

4-**Round buffer zone.** In order to ensure that overwrites do not occur, Coron et al. [17] introduce the notion of a 4-round buffer zone. Their simulator has two 4-round buffer zones, corresponding to rounds $\{3, 4, 5, 6\}$ or $\{9, 10, 11, 12\}$. Within the buffer zones, positions $\{3, 6\}$ (respectively $\{9, 12\}$) are known as the *set uniform positions*, and positions $\{4, 5\}$ (respectively $\{10, 11\}$) are known as the *adapt posi-*

tions. Coron et al. [17] prove the following property (which we call henceforth the *strong set uniform property*): At the moment that a chain is about to be completed, the set uniform positions of the buffer zone are always unassigned. This means that the simulator will always assign uniform values to $F_3(x_3)$ and $F_6(x_6)$ (respectively $F_9(x_9)$ and $F_{12}(x_{12})$) immediately before assigning values to $F_4(x_4)$ and $F_5(x_5)$ (respectively $F_{10}(x_{10})$ and $F_{11}(x_{11})$) using ForceVal. This will ensure that ForceVal never overwrites (except with negligible probability) since $x_4 = x_2 \oplus F_3(x_3)$ is only determined at the moment $F_3(x_3)$ is assigned and so the probability that $F_4(x_4)$ has already been assigned is negligible (a similar argument holds for the other adapt positions).

Rigid structure. The fixed 4-round buffer zones of their simulator is used in their proof in two ways: First, since all assignments, across all completed chains are uniform except in the fixed adapt positions $\{4, 5\}$ and $\{10, 11\}$, it is easier to argue about "undesired events" occurring. In particular, since the 4-round buffer of one chain ($\{3, 4, 5, 6\}$ or $\{9, 10, 11, 12\}$) cannot overlap with the detect zone of another chain ($\{1, 2, 13, 14\}$ or $\{7, 8\}$), they can argue that if a "undesired event" occurs while detecting a chain C, then either an equivalent chain was already enqueued or that event must have been caused by a uniform setting of $F_i(x_i)$.

Bounding the simulator's runtime. The approach of Coron et al. [17] (originally introduced by Seurin [53]) is to bound the total number of partial chains that get completed by the simulator. Note that in order to create a partial chain of the form $(x_1, x_2, x_{13}, x_{14})$, it must be the case that $P(F_1(x_1) \oplus x_2, x_1) = (x_{14}, F_{13}(x_{13}) \oplus x_{14})$

and so, intuitively, the distinguisher had to query either P or P^{-1} in order to achieve this. Thus, the number of partial chains of the form $(x_1, x_2, x_{13}, x_{14})$ (i.e., wraparound chains) that will get detected and completed by the simulator is at most the total number of queries made by the distinguisher. Since there is only a single middle detect zone $\{7, 8\}$, once we have a bound on the number of wraparound chains that are completed, we can also bound the number of completed partial chains of the form (x_7, x_8).

3.1.2 Our Techniques

We next briefly discuss how our techniques differ from the techniques of Coron et al. [17] in the four main areas discussed above.

Separating detection from completion for wrap-around chains. When the distinguisher makes a query $F_i(x_i)$ to the simulator, our simulator proceeds in two phases: In the first phase, the simulator does not make any queries, but enqueues for completion all partial chains which it *predicts* will require completion. In the second phase, the simulator actually completes the chains and detects and enqueues only on the *middle* detect zone (which in our construction corresponds to rounds $\{5, 6\}$). This simplifies our proof since it means that after the set of chains has been detected in the first phase, the simulator can complete the chains in a manner that minimizes "bad interactions" between partial chains. In particular, in the second phase, the simulator first completes chains C with the property that one of the set uniform positions is "known" and hence could already been assigned (in the completion of another chain D) before the chain C gets dequeued for completion. (Although this contradicts the strong set uniform property of [17], in our proof we are able to relax this requirement. See the discussion of the *weak set uniform property* below for further details.) The simulator then proceeds to complete (and detect and enqueue) other chains. This allows us to simplify our analysis.

Relaxed properties for the 4-round buffer zone. When a partial chain is about to be completed, we allow the case that one of the set uniform positions has already been assigned to occur, as long as the adapt position adjacent to this set uniform position has not yet been assigned. Henceforth, we call this the *weak set uniform property*. We prove that the weak set uniform property holds in Claim 3.35.

Relaxed structure. Requiring only the weak set uniform property allows us to consider a more relaxed structure for detect zones and 4-round buffer zones. Instead of requiring that for every chain that gets completed the 4 round buffer positions ($\{3, 4, 5, 6\}$ or $\{9, 10, 11, 12\}$ in the case of [17]) are always unassigned, we allow more flexibility in the position of the 4-round buffer. For example, depending on whether the detected chain is of the form (x_1, x_2, x_{10}), (x_1, x_9, x_{10}), or (x_5, x_6), our 4-round buffer will be one of: $\{3, 4, 5, 6\}$ or $\{6, 7, 8, 9\}$, $\{2, 3, 4, 5\}$ or $\{5, 6, 7, 8\}$, $\{1, 2, 3, 4\}$ or $\{7, 8, 9, 10\}$, respectively. This flexibility allows us to reduce the number of rounds. However, now, the adapt zone of one chain may coincide with the detect zone of another chain. Since there are no dedicated roles for fixed positions and since partial chains in the middle detect zone are detected during the completion of other chains, we need additional bad events $\mathsf{BadlyHitFV}$ and $\mathsf{BadlyCollideFV}$ to argue that undesired effects do not occur. Furthermore, in order to prove that a new wraparound chain does not get created during the completion of other chains we introduce the new bad event BadlyCollideP.

Bounding the simulator's runtime. In comparison to the construction of Coron et al. [17], our construction has more detect zones and, moreover, for wraparound chains, we detect on partial chains consisting of three consecutive queries instead of four consecutive queries. Nevertheless, at a high-level, our proof that the simulator runtime is bounded follows very similarly to theirs. We can first bound the number of completed partial chains of the form (x_1, x_2, x_{10}) and (x_1, x_9, x_{10}) (such chains are wraparound chains since they contain both x_1 and x_{10}). Once we have done this, we again have only a single non-wraparound detect zone and so we can follow the argument of Coron et al. [17] to bound the number of completed partial chains of the form (x_5, x_6). Once we have a bound on the number of completed partial chains, it is fairly straightforward to bound the simulator complexity.

3.1.3 Related Work

Dai and Steinberger have since showed that a 8-round Feistel network is indifferentiable from an ideal cipher [22, 23]. In other related work, Ramzan and Reyzin [51] proved that a 4-round Feistel network remains indistinguishable from a random permutation even if the adversary is given access to the middle two round functions. Gentry and Ramzan [33] showed that a 4-round Feistel network can be used to instantiate the random permutation in the Even-Mansour cipher [29] and proved that such a construction is a pseudorandom permutation, even if the round functions of the Feistel network are publicly accessible. Dodis and Puniya [27] studied security of the Feistel network in a scenario where the adversary learns intermediate values when the Feistel network is evaluated, and/or when the round functions are unpredictable but not (pseudo)random.

Coron et al. [16] adapted the notion of indifferentiability to the framework of interactive Turing machines. Various relaxations of indifferentiability, such as *public indifferentiability* [27, 58], or *honest-but-curious indifferentiability* [26], have also been considered. Dodis and Puniya [26] proved that a Feistel network with super-logarithmic number of rounds is indifferentiable from an ideal cipher in the honest-but-curious setting. Mandal et al. [45] proved that the 6-round Feistel network is publicly indifferentiable from an ideal cipher.

3.2 Preliminaries

Since we are focusing on the indifferentiability of the Feistel network from a random permutation, instead of Definition 1 which presented the definition of indifferentiability from an ideal cipher, we recall the definition of *indifferentiability from a random permutation* which was used by Coron et al. [17]. (Again, based on the definition of Maurer, Renner, and Holenstein [47].)

Definition 2. *Let $\mathcal{C}$ be a construction that, for any n, accesses functions $\mathbf{F} = (F_1, \ldots, F_r)$ over $\{0,1\}^n$ and implements an invertible permutation over $\{0,1\}^{2n}$. (We stress that $\mathcal{C}$ allows evaluation of both the forward and inverse direction of the* *permutation.) We say that $\mathcal{C}$ is* indifferentiable from a random permutation *if there exists a simulator $\mathcal{S}$ and a polynomial t such that for all distinguishers $\mathcal{D}$ making at most $q = \mathsf{poly}(n)$ queries, $\mathcal{S}$ runs in time $t(q)$ and*

$$|\Pr[\mathcal{D}^{\mathcal{C}^{\mathbf{F}},\mathbf{F}}(1^n) = 1] - \Pr[\mathcal{D}^{P,\mathcal{S}^P}(1^n) = 1]|$$

is negligible, where $\mathbf{F}$ are random, independent functions over $\{0,1\}^n$ and P is a random permutation over $\{0,1\}^{2n}$. (We stress that P can be evaluated in both the forward and inverse direction.)

3.3 Our Simulator

We present an informal overview of the simulator before giving a formal description.

3.3.1 Informal Description

The queries to $F_1, \ldots, F_{10}$ are answered by the simulator through the procedure $\mathcal{S}.\mathrm{F}(i,x)$ for $i = 1, \ldots, 10$. When the distinguisher asks a query $\mathrm{F}(i,x)$, the simulator checks to see if response to the query has already been set. The queries that are already set are held in tables $G_1, \ldots, G_{10}$ as pairs (x,y) such that if $\mathrm{F}(i,x)$ is queried, and if $x \in G_i$, then y is returned as the answer to query $\mathrm{F}(i,x)$. If the query has not already been set, then the simulator adds x to the set A_i^j where j indicates the j^{th} query of the distinguisher. The simulator then checks if $i \in \{1,2,5,6,9,10\}$ (where these positions mark the endpoints of the detect zones) and, if so, checks to see if any new partial chains of the form $(x_9, x_{10}, 9)$, $(x_1, x_2, 1)$ and $(x_5, x_6, 5)$

need to be enqueued. Informally, (x_k, x_{k+1}, k) is referred to as a partial chain where (x_k, x_{k+1}) denote two consecutive round values for rounds k and $k+1$. If no new partial chains are detected, the simulator just sets the value of $G_i(x)$ uniformly at random and returns that value. If new partial chains are detected and enqueued in Q_{enq}, then the simulator evaluates these partial chains "forward" and "backward" as much as possible (without setting any new values of $G_{i'}(x')$ for any i' and x'). Say the evaluation stopped due to $x' \notin G_{i'}$ for some x' and i'. Then, the simulator adds x' to $A^j_{i'}$ and checks if $i' \in \{1, 2, 5, 6, 9, 10\}$ and, if so, detects any additional partial chains and enqueues them for completion if necessary and continues repeating the process until no more partial chains are detected.

The chains enqueued for completion during this process are enqueued in queues Q_1,Q_5,Q_6,Q_{10} and Q_{all}. Any chain that has been enqueued in Q_{enq} is also enqueued in Q_{all}. The chains enqueued in Q_b for $b \in \{1, 5, 6, 10\}$ are those that exhibit the *weak set uniform property.* Specifically, say a partial chain $C = (x_k, x_{k+1,k})$ is enqueued to be adapted at positions ℓ and $\ell + 1$ and the "set uniform" positions for C are at $\ell - 1$ and $\ell + 2$. The design of the simulator ensures that one of the "set uniform" positions is adjacent to the query that caused the chain C to be enqueued. We refer to this "set uniform" position as "good" set uniform position g and the other "set uniform" position as "bad" set uniform position b. (Thus, $g, b \in \{\ell - 1, \ell + 2\}$ by definition.) The chain C is enqueued in Q_b if at the time of enqueueing of the chain C, C can be evaluated up to the "bad" set uniform position b and the value of chain

C at b, say x_b, is such that $x_b \notin G_b$.[1]

The completion of enqueued chains starts with the completion of the chains enqueued in Q_b for $b \in \{1, 5, 6, 10\}$. This process proceeds similarly to the completion process in Coron et al. [17]. A chain C is dequeued from Q_b and the simulator evaluates the chain forward/backward up to the 4-round buffer setting $G_i(x_i)$ values uniformly for any $x_i \notin G_i$ that comes up in the evaluation. In the 4-round buffer consisting of the "set uniform" positions and the "adapt" positions, the simulator sets the values of C at the set uniform positions uniformly (if it has not already been set) and forces the values at the adapt positions such that evaluation of the Feistel is consistent with the random permutation. (Note that this could possibly lead to a value in $G_i(\cdot)$ getting overwritten. A major technical part of the proof is to show that this happens with negligible probability.) Once the simulator has completed the chain C, it places C in the set $\mathsf{CompletedChains}$ along with the chains that are obtained by evaluating C forward that are in the detect zone positions, i.e., chains of the form (x_k, x_{k+1}, k) for $k = 1, 5, 9$.

Once the simulator completes the chains enqueued in Q_b for all $b \in \{1, 5, 6, 10\}$, the simulator completes the remaining chains enqueued in Q_{all}. The completion process for the remaining chains enqueued in Q_{all} is the same as the completion

[1] There are chains that exhibit this property that are not enqueued for completion and are instead added to a set $\mathsf{MidEquivChains}$. The reason for not enqueueing these chains is only to simplify the analysis for the bound of the complexity of the simulator. We will later show that ignoring these chains does not affect the simulation and in fact, these chains will be completed at the end of the simulator's run while answering $\mathcal{D}$'s j^{th} query.

process described above except that the simulator detects additional partial chains of the form $(x_5, x_6, 5)$ during the completion and enqueues them in a queue Q_{mid}, i.e., during the completion of a chain C in Q_{all}, if an assignment occurs such that $x_k \in G_k$ for some $k \in \{5, 6\}$ due to the assignment and $x_k \notin G_k$ before the assignment, then the simulator enqueues the partial chain $(x_5, x_6, 5)$ in Q_{mid} for all $x_{k'} \in G_{k'}$ such that $k' \in \{5, 6\}$ and $k \neq k'$. (Note that the assignment could be a ForceVal assignment as well.)

Finally, the simulator completes all the chains in Q_{mid} that are not already in CompletedChains. The completion process again is the same as the process described for chains enqueued in Q_b. The simulator then returns the answer $G_i(x)$ to the query $\mathrm{F}(i, x)$.

3.3.2 Formal Description

The simulator $\mathcal{S}$ internally uses hashtables $G_1, \ldots, G_{10}$ to store the function values. Additionally, it uses sets $A_1, \ldots, A_{10}$ to detect partial chains that need to be completed; A_i stores input values that will be added to G_i in the future. The simulator uses a queue Q_{enq} to detect partial chains that need to be completed and stores a copy of Q_{enq} in a queue Q_{all} that is used during completion. Queues Q_1, Q_5, Q_6, Q_{10} are used to store the chains in Q_{enq} whose "bad" set uniform position is known at the time of detection. Additionally, a queue Q_{mid} is used to store new chains of the form $(x_5, x_6, 5)$ that are enqueued during the completion of chains from Q_{all}. A set CompletedChains is used to store the chains that have been completed already. Finally, a set MidEquivChains is used to hold chains of the form $(x_1, x_2, 1)$ and $(x_9, x_{10}, 9)$ that are detected due to $\mathrm{P}/\mathrm{P}^{-1}$ queries made by the simulator. This set is needed only for the purpose of analyzing the complexity of the simulator.

Variables: Queues $Q_{\text{enq}}, Q_{\text{all}}, Q_1, Q_5, Q_6, Q_{10}, Q_{\text{mid}}$, hashtables $G_1, \ldots, G_{10}$, sets A_i^j initialized to $\emptyset$ for $i = 1, \ldots, 10$ and $j = 1, \ldots, q$ where q is the maximum number of queries made by the distinguisher, set CompletedChains:$= \emptyset$ and set MidEquivChains $:= \emptyset$. Initialize $j := 0$.

The public procedure $\mathrm{F}(i, x)$ of the simulator provides the interface to a distinguisher.

```
procedure $\mathrm{F}(i, x)$:
    $j := j + 1$
    for $i \in \{1, \ldots, 10\}$ do
        $A_i^j := \emptyset$
    $\mathrm{F}^{\mathrm{ENQ}}(i, x)$
    while $\neg Q_{\text{enq}}$.Empty() do
        $(x_k, x_{k+1}, k, \ell, g, b) := Q_{\text{enq}}$.Dequeue()
        if $(x_k, x_{k+1}, k) \notin$ CompletedChains then
            $(x_r, x_{r+1}, r) :=$ EvalFwdEnq$(x_k, x_{k+1}, k, \ell - 2)$
            if $r + 1 = b \wedge x_{r+1} \notin G_{r+1}$ then
                $Q_b$.Enqueue$(x_k, x_{k+1}, k, \ell, g, b)$
            $(x_r, x_{r+1}, r) :=$ EvalBwdEnq$(x_k, x_{k+1}, k, \ell + 2)$
            if $r = b \wedge x_r \notin G_r$ then
                $Q_b$.Enqueue$(x_k, x_{k+1}, k, \ell, g, b)$
    for each $Q \in \langle Q_1, Q_5, Q_6, Q_{10}, Q_{\text{all}}, Q_{\text{mid}} \rangle$ do        ▷ processed in that order
        while $\neg Q$.Empty() do
            $(x_k, x_{k+1}, k, \ell, g, b) := Q$.Dequeue()
            if $(x_k, x_{k+1}, k) \notin$ CompletedChains then
                $(x_{\ell-2}, x_{\ell-1}) :=$ EvalFwdComp$(Q, x_k, x_{k+1}, k, \ell - 2)$
                $(x_{\ell+2}, x_{\ell+3}) :=$ EvalBwdComp$(Q, x_k, x_{k+1}, k, \ell + 2)$
                Adapt$(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$
                $(x_1, x_2) :=$ EvalBwdComp$(\perp, x_k, x_{k+1}, k, 1)$
                $(x_5, x_6) :=$ EvalFwdComp$(\perp, x_1, x_2, 1, 5)$
                $(x_9, x_{10}) :=$ EvalFwdComp$(\perp, x_1, x_2, 1, 9)$
                CompletedChains := CompletedChains$\cup\{(x_1, x_2, 1), (x_5, x_6, 5), (x_9, x_{10}, 9)\}$
    $\mathrm{F}^{\mathrm{COMP}}(\perp, i, x)$
    return $G_i(x)$
```

```
procedure EvalFwdEnq($x_k, x_{k+1}, k, m$):
    if $k = 5$ then
        flagForMid:= 1
    while $(k \neq m) \wedge ((k = 10) \vee (\mathrm{F}^{\mathrm{ENQ}}(k+1, x_{k+1}) \neq \perp))$ do
        if $k = 10$ then
            $(x_0, x_1) := P^{-1}(x_{10}, x_{11})$
            $k := 0$
        else
            if $k = 9 \wedge \mathsf{flagForMid} = 1$ then
                $\mathsf{MidEquivChains} := \mathsf{MidEquivChains} \cup \{(x_k, x_{k+1}, k)\}$
            $x_{k+2} := x_k \oplus G(k+1, x_{k+1})$
            $k := k + 1$
    flagForMid:= 0
    return $(x_k, x_{k+1}, k)$

procedure EvalBwdEnq($x_k, x_{k+1}, k, m$):
    if $k = 5$ then
        flagForMid:= 1
    while $(k \neq m) \wedge ((k = 0) \vee (\mathrm{F}^{\mathrm{ENQ}}(k, x_k) \neq \perp))$ do
        if $k = 0$ then
            $(x_{10}, x_{11}) := P(x_0, x_1)$
            $k := 10$
        else
            if $k = 1 \wedge \mathsf{flagForMid} = 1$ then
                $\mathsf{MidEquivChains} := \mathsf{MidEquivChains} \cup \{(x_k, x_{k+1}, k)\}$
            $x_{k-1} := x_{k+1} \oplus G(k, x_k)$
            $k := k - 1$
    flagForMid:= 0
    return $(x_k, x_{k+1}, k)$

procedure $\mathrm{F}^{\mathrm{ENQ}}(i, x)$:
    if $x \in G_i$ then
        return $G_i(x)$
    else if $x \in A_i^j$ then
        return $\perp$
    else
        $A_i^j := \{x\} \cup A_i^j$
        if $i \in \{1, 2, 5, 6, 9, 10\}$ then
            EnqNewChains($i, x$)
        return $\perp$

procedure EnqNewChains($i, x$):
    if $i = 1$ then
```

```
        **for all** $(x_9, x_{10}, x_1) \in (G_9 \cup A_9^j) \times G_{10} \times \{x\}$ **do**
            **if** $\textsc{CheckBwd}(x_{10}, G_{10}(x_{10}) \oplus x_9, x_1)$ **then**
                **if** $(x_9, x_{10}, 9) \notin \mathsf{MidEquivChains}$ **then**
                    $Q_{\text{enq}}.\textsc{Enqueue}(x_9, x_{10}, 9, 3, 2, 5)$
                    $Q_{\text{all}}.\textsc{Enqueue}(x_9, x_{10}, 9, 3, 2, 5)$
    **if** $i = 2$ **then**
        **for all** $(x_{10}, x_1, x_2) \in (G_{10} \cup A_{10}^j) \times G_1 \times \{x\}$ **do**
            **if** $\textsc{CheckFwd}(x_2 \oplus G_1(x_1), x_1, x_{10})$ **then**
                **if** $(x_1, x_2, 1) \notin \mathsf{MidEquivChains}$ **then**
                    $Q_{\text{enq}}.\textsc{Enqueue}(x_1, x_2, 1, 4, 3, 6)$
                    $Q_{\text{all}}.\textsc{Enqueue}(x_1, x_2, 1, 4, 3, 6)$
    **if** $i = 5$ **then**
        **for all** $(x_5, x_6) \in \{x\} \times (G_6 \cup A_6^j)$ **do**
            $Q_{\text{enq}}.\textsc{Enqueue}(x_5, x_6, 5, 2, 4, 1)$
            $Q_{\text{all}}.\textsc{Enqueue}(x_5, x_6, 5, 2, 4, 1)$
    **if** $i = 6$ **then**
        **for all** $(x_5, x_6) \in (G_5 \cup A_5^j) \times \{x\}$ **do**
            $Q_{\text{enq}}.\textsc{Enqueue}(x_5, x_6, 5, 8, 7, 10)$
            $Q_{\text{all}}.\textsc{Enqueue}(x_5, x_6, 5, 8, 7, 10)$
    **if** $i = 9$ **then**
        **for all** $(x_9, x_{10}, x_1) \in \{x\} \times G_{10} \times (G_1 \cup A_1^j)$ **do**
            **if** $\textsc{CheckBwd}(x_{10}, G_{10}(x_{10}) \oplus x_9, x_1)$ **then**
                **if** $(x_9, x_{10}, 9) \notin \mathsf{MidEquivChains}$ **then**
                    $Q_{\text{enq}}.\textsc{Enqueue}(x_9, x_{10}, 9, 6, 8, 5)$
                    $Q_{\text{all}}.\textsc{Enqueue}(x_9, x_{10}, 9, 6, 8, 5)$
    **if** $i = 10$ **then**
        **for all** $(x_{10}, x_1, x_2) \in \{x\} \times G_1 \times (G_2 \cup A_2^j)$ **do**
            **if** $\textsc{CheckFwd}(x_2 \oplus G_1(x_1), x_1, x_{10})$ **then**
                **if** $(x_1, x_2, 1) \notin \mathsf{MidEquivChains}$ **then**
                    $Q_{\text{enq}}.\textsc{Enqueue}(x_1, x_2, 1, 7, 9, 6)$
                    $Q_{\text{all}}.\textsc{Enqueue}(x_1, x_2, 1, 7, 9, 6)$

**procedure** $\textsc{CheckFwd}(x_0, x_1, x_{10})$:
    $(x'_{10}, x'_{11}) := P(x_0, x_1)$
    **return** $x'_{10} \overset{?}{=} x_{10}$

**procedure** $\textsc{CheckBwd}(x_{10}, x_{11}, x_1)$:
    $(x'_0, x'_1) := P^{-1}(x_{10}, x_{11})$
    **return** $x'_1 \overset{?}{=} x_1$

**procedure** $\textsc{EvalFwdComp}(Q, x_k, x_{k+1}, k, m)$:
    **while** $k \neq m$ **do**
        **if** $k = 10$ **then**
```

```
            $(x_0, x_1) := P^{-1}(x_{10}, x_{11})$
            $k := 0$
        else
            $x_{k+2} := x_k \oplus \mathrm{F}^{\mathrm{COMP}}(Q, k+1, x_{k+1})$
            $k := k + 1$
    return $(x_m, x_{m+1})$

procedure $\textsc{EvalBwdComp}(Q, x_k, x_{k+1}, k, m)$:
    while $k \neq m$ do
        if $k = 0$ then
            $(x_{10}, x_{11}) := P(x_0, x_1)$
            $k := 10$
        else
            $x_{k-1} := x_{k+1} \oplus \mathrm{F}^{\mathrm{COMP}}(Q, k, x_k)$
            $k := k - 1$
    return $(x_m, x_{m+1})$

procedure $\mathrm{F}^{\mathrm{COMP}}(Q, i, x)$:
    if $x \notin G_i$ then
        $G_i(x) \leftarrow \{0,1\}^n$
        if $Q \neq \perp \wedge Q = Q_{\mathrm{all}} \wedge i \in \{5, 6\}$ then
            $\textsc{EnqNewMidChains}(i, x)$
    return $G_i(x)$

procedure $\textsc{EnqNewMidChains}(i, x)$:
    if $i = 5$ then
        for all $(x_5, x_6) \in \{x\} \times G_6$ do
            $Q_{\mathrm{mid}}.\textsc{Enqueue}(x_5, x_6, 5, 2, 4, 1)$
    if $i = 6$ then
        for all $(x_5, x_6) \in G_5 \times \{x\}$ do
            $Q_{\mathrm{mid}}.\textsc{Enqueue}(x_5, x_6, 5, 8, 7, 10)$

procedure $\textsc{Adapt}(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$:
    $\mathsf{flagMidAdapt0} := 0$
    $\mathsf{flagMidAdapt1} := 0$
    $\mathrm{F}^{\mathrm{COMP}}(Q, \ell - 1, x_{\ell-1})$
    $x_\ell := x_{\ell-2} \oplus G_{\ell-1}(x_{\ell-1})$
    if $(Q = Q_{\mathrm{all}}) \wedge (\ell = 5 \vee \ell = 6) \wedge (x_\ell \notin G_\ell)$ then
        $\mathsf{flagMidAdapt0} := 1$
    $\mathrm{F}^{\mathrm{COMP}}(Q, \ell + 2, x_{\ell+2})$
    $x_{\ell+1} := x_{\ell+3} \oplus G_{\ell+2}(x_{\ell+2})$
    if $(Q = Q_{\mathrm{all}}) \wedge (\ell + 1 = 5 \vee \ell + 1 = 6) \wedge (x_{\ell+1} \notin G_{\ell+1})$ then
        $\mathsf{flagMidAdapt1} := 1$
    ForceVal($x_\ell, x_{\ell+1} \oplus x_{\ell-1}, \ell$)
    if flagMidAdapt0 $= 1$ then
        EnqNewMidChains($\ell, x_\ell$)
    ForceVal($x_{\ell+1}, x_\ell \oplus x_{\ell+2}, \ell + 1$)
    if flagMidAdapt1 $= 1$ then
        EnqNewMidChains($\ell + 1, x_{\ell+1}$)

procedure ForceVal($x, y, \ell$):
    $G_\ell(x) := y$
```

3.4 Proof of Indifferentiability

Let $\mathsf{Feistel}$ denote the 10-round Feistel construction, let $\mathbf{F}$ be 10 independent random functions with domain and range $\{0,1\}^n$, and let P denote a random permutation on $\{0,1\}^{2n}$. We let $\mathcal{S}$ denote the simulator from the previous section. This section is dedicated to proving the following result:

Theorem 3.1. *The probability that a distinguisher $\mathcal{D}$ making at most q queries outputs 1 in an interaction with $(P, \mathcal{S}^P)$ and the probability that it outputs 1 in an interaction with $(\mathsf{Feistel}^{\mathbf{F}}, \mathbf{F})$ differ by at most $O(q^{12}/2^n)$. Moreover, $\mathcal{S}$ runs in time polynomial in q.*

For the remainder of the chapter, fix some distinguisher $\mathcal{D}$ making at most q queries.

3.4.1 Proof Overview

Our proof structure utilizes four hybrid experiments $H_1, \ldots, H_4$ as in the proof of Coron et al. [17]. Hybrid H_1 denotes the scenario in which $\mathcal{D}$ interacts with

$(P, \mathcal{S}^P)$, and H_4 denotes the scenario in which $\mathcal{D}$ interacts with $(\mathsf{Feistel}^{\mathbf{F}}, \mathbf{F})$. To prove indifferentiability, we show that the difference between the probability $\mathcal{D}$ outputs 1 in H_1 and the probability $\mathcal{D}$ outputs 1 in H_4 is at most $O(q^12/2^n)$.

In H_2, we replace the random permutation P in H_1 with a two-sided random function R that also implements $R.\textsc{CheckFwd}$ and $R.\textsc{CheckBwd}$. Following Coron et al. [17], we first upper bound the simulator complexity in hybrid H_2. In order to bound the simulator's complexity in H_1, we would like to argue that the *simulator*'s views are indistinguishable in H_1 and H_2. However, in H_1 the simulator itself implements $\textsc{CheckFwd}$ and $\textsc{CheckBwd}$, while in H_2 the procedures $R.\textsc{CheckFwd}$ and $R.\textsc{CheckBwd}$ are provided by the two-sided random function R. Thus, we introduce an additional hybrid $H_{1.5}$ in which $\mathcal{D}$ interacts with $(P, \hat{\mathcal{S}}^{P^+})$, where P^+ denotes a random permutation equipped with additional procedures $\textsc{CheckFwd}$ and $\textsc{CheckBwd}$ that are implemented as in the simulator $\mathcal{S}$. The simulator $\hat{\mathcal{S}}$ in $H_{1.5}$ implements the procedures $\textsc{CheckFwd}$ and $\textsc{CheckBwd}$ by simply calling $P^+.\textsc{CheckFwd}$ and $P^+.\textsc{CheckBwd}$. Thus, the difference in hybrids H_1 and $H_{1.5}$ is purely conceptual. To bound the simulator's complexity in H_1 we then argue that (1) the number of queries made by the simulator is essentially the same in H_1 and $H_{1.5}$; and (2) the views of the simulator in $H_{1.5}$ and H_2 are indistinguishable.

Next, we define certain low-probability events (referred to as "bad events") that can occur in an execution of H_2, and show that if these events do not occur in an execution of H_2, then we can prove certain "good" properties; in particular, we can prove that for every call to $\textsc{ForceVal}(x, \cdot, j)$ that occurs in the execution, we have that $x \notin G_j$ before the call. If this is true, we say that "ForceVal does not overwrite." This is the main technical part of the proof.

In H_3, we replace the two-sided random function R from H_2 with the 10-round Feistel construction $\mathsf{Feistel}$. This implies that the distinguisher $\mathcal{D}$ interacts with $(\mathsf{Feistel}, \hat{\mathcal{S}}^{\mathsf{Feistel}^+})$, where $\mathsf{Feistel}^+$ (similar to P^+) denotes the $\mathsf{Feistel}$ construction with additional procedures CheckFwd and CheckBwd, and the Feistel construction and the simulator share the same randomness. Given the property that ForceVal does not "overwrite," we prove that the distinguishing advantage of $\mathcal{D}$ is exactly the probability with which the "bad events" occur by explicitly mapping the randomness used in H_2 to the randomness used in H_3. The mapping and the proof follow exactly along the lines of the proof in Coron et al. [17].

Finally, in H_4, the distinguisher accesses the random functions $\mathbf{F}$ directly instead of accessing them through the simulator.

3.4.2 Indistinguishability of the First and Second Experiments

Recall that in experiment H_1, the distinguisher $\mathcal{D}$ interacts with $(P, \mathcal{S}^P)$. We define an intermediate hybrid experiment $H_{1.5}$ in which $\mathcal{D}$ interacts with $(P, \hat{\mathcal{S}}^{P^+})$, where P^+ is a random permutation that provides procedures CheckFwd and CheckBwd that are implemented as in the simulator $\mathcal{S}$. Also, the simulator $\hat{\mathcal{S}}$ in experiment $H_{1.5}$ differs from $\mathcal{S}$ in that $\hat{\mathcal{S}}$.CheckFwd simply calls P^+.CheckFwd and $\hat{\mathcal{S}}$.CheckBwd simply calls P^+.CheckBwd.

The difference between experiments H_1 and $H_{1.5}$ is only conceptual, as all we have done is to move the CheckFwd and CheckBwd procedures from the simulator into an oracle to which the simulator has access. The following is thus immediate.

Lemma 3.2. *The probability that $\mathcal{D}$ outputs 1 in H_1 is equal to the probability it outputs 1 in $H_{1.5}$. Moreover, the number of queries $\mathcal{S}$ makes to P plus the number of times $\mathcal{S}$ internally runs* CheckFwd *and* CheckBwd *in H_1 is equal to the number of queries $\hat{\mathcal{S}}$ makes to P^+ in $H_{1.5}$.*

Experiment H_2 differs from $H_{1.5}$ in that we replace the random permutation P^+ with a random two-sided function R. This two-sided function maintains a hashtable p containing elements of the form $(\downarrow, x_0, x_1)$ and $(\uparrow, x_{10}, x_{11})$ that can be mapped to each other. Whenever the procedure $R.\mathrm{P}(x_0, x_1)$ is queried, R checks if $(\downarrow, x_0, x_1) \in p$ and if so, answers accordingly, i.e., it returns $(x_{10}, x_{11}) = p(\downarrow, x_0, x_1)$. Otherwise, an independent uniform output (x_{10}, x_{11}) is picked and $(\downarrow, x_0, x_1)$ as well as $(\uparrow, x_{10}, x_{11})$ are added to p, mapping to each other. Queries to the procedure $R.\mathrm{P}^{-1}(\cdot)$ are answered analogously.

In addition to procedures P and P^{-1}, R contains the following two procedures: CheckFwd(x_0, x_1, x_{10}) and CheckBwd(x_{10}, x_{11}, x_1).[2] The former, procedure CheckFwd(x_0, x_1, x_{10}), works as follows: If $(\downarrow, x_0, x_1) \in p$, it returns true if $(\downarrow, x_0, x_1)$ maps to (x_{10}, x_{11}) for some value of $x_{11} \in \{0,1\}^n$ and false otherwise. The latter procedure CheckBwd(x_{10}, x_{11}, x_1) works as follows: If $(\uparrow, x_{10}, x_{11}) \in p$, it returns true if $(\uparrow, x_{10}, x_{11})$ maps to (x_0, x_1) for some value of $x_0 \in \{0,1\}^n$ and false

[2] This is similar to the check procedure in [17].

otherwise.

The pseudocode for the two-sided random function R, using hashtable p, is as follows:

```
procedure P($x_0, x_1$):
    if $(\downarrow, x_0, x_1) \notin p$ then
        $(x_{10}, x_{11}) \xleftarrow{\$} \{0,1\}^{2n}$
        $p(\downarrow, x_0, x_1) := (x_{10}, x_{11})$
        $p(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$ // May overwrite an entry
    return $p(\downarrow, x_0, x_1)$

procedure P$^{-1}$($x_{10}, x_{11}$):
    if $(\uparrow, x_{10}, x_{11}) \notin p$ then
        $(x_0, x_1) \xleftarrow{\$} \{0,1\}^{2n}$
        $p(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$
        $p(\downarrow, x_0, x_1) := (x_{10}, x_{11})$ // May overwrite an entry
    return $p(\uparrow, x_{10}, x_{11})$

procedure CheckFwd($x_0, x_1, x_{10}$):
    if $(\downarrow, x_0, x_1) \in p$ then
        $(x'_{10}, x'_{11}) := p(\downarrow, x_0, x_1)$
        return $x'_{10} \overset{?}{=} x_{10}$
    return false

procedure CheckBwd($x_{10}, x_{11}, x_1$):
    if $(\uparrow, x_{10}, x_{11}) \in p$ then
        $(x'_0, x'_1) := p(\uparrow, x_{10}, x_{11})$
        return $x'_1 \overset{?}{=} x_1$
    return false
```

Figure 3.1: Random two-sided function R.

Next, we show that H_2 and $H_{1.5}$ are indistinguishable.

Indistinguishability of H_2 and $H_{1.5}$. We first bound the effect of replacing P^+ with R as a function of the total number of queries q' made to this oracle. Let $\mathcal{S}'$ consist of both the simulator and $\mathcal{D}$ and the number of queries to $\mathcal{S}'$ correspond to the queries to P/P^{-1} and CheckFwd/CheckBwd made by both the simulator and the distinguisher. To conclude the proof of indistinguishability, we bound the number of queries to the oracle as a function of the total number of queries q made by $\mathcal{D}$.

Theorem 3.3. *Let $\mathcal{S}'$ be an algorithm that makes at most q' queries to an oracle. Then the difference between the probability that $\mathcal{S}'$ outputs 1 when interacting with P^+, and the probability that $\mathcal{S}'$ outputs 1 when interacting with R is at most $12q'^2/2^n$.*

Proof. In order to prove this theorem, we consider experiments $E_0, E_1, \ldots, E_3$, where E_0 corresponds to $\mathcal{S}'$ interacting with P^+, and E_3 corresponds to $\mathcal{S}'$ interacting with R.

Experiment E_0: $\mathcal{S}'$ interacts with P^+.
Experiment E_1: $\mathcal{S}'$ interacts with P_1 where procedure P_1.P is defined as follows:

```
procedure $P_1$.P$(x_0, x_1)$
    if $(\downarrow, x_0, x_1) \notin p$ then
        $(x_{10}, x_{11}) \overset{\$}{\leftarrow} \{0,1\}^{2n}$
        if $(\uparrow, x_{10}, x_{11}) \in p$ then
            $(x_{10}, x_{11}) \overset{\$}{\leftarrow} \{0,1\}^{2n} \setminus \{(x'_{10}, x'_{11}) | (\uparrow, x'_{10}, x'_{11}) \in p\}$
        $p(\downarrow, x_0, x_1) := (x_{10}, x_{11})$
        $p(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$
    return $p(\downarrow, x_0, x_1)$
```

Procedure P_1.P^{-1} is defined analogously. Procedures P_1.CheckFwd and P_1.CheckBwd are defined as follows.

```
procedure CheckFwd$(x_0, x_1, x_{10})$:
    if $(\downarrow, x_0, x_1) \in p$ then
        $(x'_{10}, x'_{11}) := p(\downarrow, x_0, x_1)$
        return $x'_{10} \overset{?}{=} x_{10}$
    $(x'_{10}, x'_{11}) :=$ P$(x_0, x_1)$ //Note that procedure $P_1$.P is called
    return $x'_{10} \overset{?}{=} x_{10}$
```

```
procedure $\textsc{CheckBwd}(x_{10}, x_{11}, x_1)$:
    if $(\uparrow, x_{10}, x_{11}) \in p$ then
        $(x'_0, x'_1) := p(\uparrow, x_{10}, x_{11})$
        return $x'_1 \overset{?}{=} x_1$
    $(x'_0, x'_1) := \mathrm{P}^{-1}(x_{10}, x_{11})$ //Note that procedure $P_1.\mathrm{P}^{-1}$ is called
    return $x'_1 \overset{?}{=} x_1$
```

Claim 3.4. *The probabilities that $\mathcal{S}'$ outputs 1 in E_0 and E_1 are identical.*

Proof. The values assigned through procedures P and P^{-1} are chosen uniformly from the set of values that have not been assigned so far in both E_0 and E_1. Also, the procedures CheckFwd and CheckBwd are implemented in a similar manner in both experiments. So, experiments E_0 and E_1 behave identically. □

Experiment E_2: $\mathcal{S}'$ interacts with P_2 where the procedures $P_2.\mathrm{P}$ and $P_2.\mathrm{P}^{-1}$ are defined exactly as $R.\mathrm{P}$ and $R.\mathrm{P}^{-1}$ respectively, while procedures P_2.CheckFwd and P_2.CheckBwd are defined as in P_1.CheckFwd and P_1.CheckBwd respectively.

Claim 3.5. *The probability that $\mathcal{S}'$ outputs 1 in E_1 differs from the probability that it outputs 1 in E_2 by at most $q'^2/2^{2n}$.*

Proof. The proof of the claim follows exactly along the lines of [17, Lemma 3.8]. □

Experiment E_3: $\mathcal{S}'$ interacts with R.

Claim 3.6. *The probability that $\mathcal{S}'$ outputs 1 in E_2 differs from the probability that it outputs 1 in E_3 by at most $11q'^2/2^n$.*

Proof. In order to prove this claim, we introduce events BadCheckForward, BadCheckBackward, BadOverwrite, and BadBackwardQuery and prove that these events occur with probability at most $11q'^2/2^n$. We proceed to argue that if none of these events occur the experiments E_2 and E_3 behave exactly the same.

The event $\mathsf{BadCheckForward}$ occurs if $P'.\textsc{CheckFwd}$ returns true in the last line in E_2. Similarly, the event $\mathsf{BadCheckBackward}$ occurs if $P'.\textsc{CheckBwd}$ returns true in the last line in E_2. The probability of the events $\mathsf{BadCheckForward}$ and $\mathsf{BadCheckBackward}$ is at most $q'/2^n$.

The event $\mathsf{BadOverwrite}$ occurs if in either E_2 or in E_3, in any call to the procedures P or P^{-1}, an entry of p is overwritten. The probability that $\mathsf{BadOverwrite}$ occurs in E_2 and E_3 is at most $2(q')^2/2^{2n}$.

The event $\mathsf{BadBackwardQuery}$ occurs if in E_2 either one of the two following events occur. (1) There exists $(x_0, x_1), (x_{10}^*, x_{11}^*)$ such that all of the following hold:

(i) The query $\mathrm{P}(x_0, x_1)$ is issued in the last line of a $\textsc{CheckFwd}$ query, and $p(\downarrow, x_0, x_1)$ is set to (x_{10}^*, x_{11}^*).

(ii) A query $\mathrm{P}^{-1}(x_{10}^*, x_{11}^*)$ or $\textsc{CheckFwd}(x_0, x_1, x_{10}^*)$ or $\textsc{CheckBwd}(x_1, x_{10}^*, x_{11}^*)$ is issued after (1).

(iii) The query $\mathrm{P}(x_0, x_1)$ is not issued by the distinguisher between point (i) and point (ii).

(2) There exist $(x_0^*, x_1^*), (x_{10}, x_{11})$ such that all the following hold:

(i) The query $\mathrm{P}^{-1}(x_{10}, x_{11})$ is issued in the last line of a $\textsc{CheckBwd}$ query, and $p(\uparrow, x_{10}, x_{11})$ is set to (x_0^*, x_1^*).

(ii) After (i), a query $\mathrm{P}(x_0^*, x_1^*)$ or $\textsc{CheckFwd}(x_0^*, x_1^*, x_{10})$ or $\textsc{CheckBwd}(x_1^*, x_{10}, x_{11})$

is issued.

(iii) The query $\mathrm{P}^{-1}(x_{10}, x_{11})$ is not issued by the distinguisher between point (i) and point (ii).

Consider the probability with which the event ($\mathsf{BadBackwardQuery} \wedge \neg\mathsf{BadCheckForward} \wedge \neg\mathsf{BadCheckBackward}$) occurs. To analyze the probability of this event, let us consider the first case by which $\mathsf{BadBackwardQuery}$ can occur. Consider any pair $(x_0, x_1, x^*_{10}, x^*_{11})$ such that condition (i) of event (1) holds. Since $\mathsf{BadCheckForward}$ does not occur, we have that the $\textsc{CheckFwd}$ query returns false. Now, as long as the queries $\mathrm{P}(x_0, x_1)$, $\mathrm{P}^{-1}(x^*_{10}, x^*_{11})$, $\textsc{CheckFwd}(x_0, x_1, x^*_{10})$ or $\textsc{CheckBwd}(x_1, x^*_{10}, x^*_{11})$ are not made by the distinguisher, the value (x^*_{10}, x^*_{11}) is distributed uniformly in the set of all pairs (x'_{10}, x'_{11}) for which $\textsc{CheckFwd}(x_0, x_1, x'_{10})$ and $\textsc{CheckBwd}(x_1, x'_{10}, x'_{11})$ were not queried. Thus, the probability that in a single query, the distinguisher queries (a) $\mathrm{P}^{-1}(x^*_{10}, x^*_{11})$, is at most $\frac{q'}{2^{2n}-q'}$, (b) $\textsc{CheckFwd}(x_0, x_1, x^*_{10})$, is at most $\frac{q'}{2^n-q'}$, or (c) $\textsc{CheckBwd}(x_1, x^*_{10}, x^*_{11})$ is at most $\frac{q'}{2^{2n}-q'}$. By a similar argument for event (2) and by assuming $q' < 2^n/2$ and since there are at most q' queries made by the distinguisher, the probability of ($\mathsf{BadBackwardQuery} \wedge \neg\mathsf{BadCheckForward} \wedge \neg\mathsf{BadCheckBackward}$) is at most $2 * (2(q')^2/2^{2n} + 2q'^2/2^n) \leq 8(q')^2/2^n$.

If the bad events defined above do not occur, the experiments E_2 and E_3 behave identically. □

Theorem 3.3 follows from Claims 3.4–3.6. □

Theorem 3.3 bounds the effect of replacing P^+ with R in terms of the total number of queries q' made to the P^+/R oracle. In the following sequence of claims, we show that $q' = O(q^6)$ (where, recall, q is the total number of queries made by the original distinguisher $\mathcal{D}$).

Claim 3.7. *In H_2, the simulator dequeues from the queue Q_{enq} a partial chain of the form $(x_1, x_2, 1, \ell, g, b)$ such that $(x_1, x_2, 1) \notin \mathsf{CompletedChains}$ at most q times in total.*

Proof. Consider such a dequeue call and let $(x_1, x_2, 1, \ell, g, b)$ be the chain dequeued from Q_{enq}. The chain must have been enqueued in Q_{enq} when $\textsc{CheckFwd}(x_2 \oplus G_1(x_1), x_1, x_{10}) = \mathsf{true}$ for a fixed x_{10} in lines 75 and 95 of the simulator. Since $G_1(x_1)$ is never overwritten (as 1 is not an adapt position for any chain), there is a tuple (x_0, x_1, x_{10}) where $x_0 = x_2 \oplus G_1(x_1)$ such that $\textsc{CheckFwd}(x_2 \oplus G_1(x_1), x_1, x_{10}) = \mathsf{true}$ when the chain was enqueued. This implies that there must have been a call either to $\mathrm{P}(x_0, x_1)$ such that (x_{10}, x_{11}) was chosen uniformly in line 3 of R, or to $\mathrm{P}^{-1}(x_{10}, x_{11})$ such that (x_0, x_1) was chosen uniformly in line 9 of R for some $x_{11} \in \{0,1\}^n$. This call to P or P^{-1} was made by the distinguisher or the simulator. If $(x_1, x_2, 1) \notin \mathsf{CompletedChains}$, we claim that this call was made by the distinguisher. This is because, by definition, the simulator queries P or P^{-1} only in the following cases:

1. in a call to $\textsc{EvalFwdEnq}$ or $\textsc{EvalBwdEnq}$
2. in a call to $\textsc{EvalFwdComp}$ or $\textsc{EvalBwdComp}$

We first consider the case where the simulator queried $\mathrm{P}/\mathrm{P}^{-1}$ during a call to $\textsc{EvalFwdEnq}(x_k, x_{k+1}, k, m)$ or $\textsc{EvalBwdEnq}(x_k, x_{k+1}, k, m)$. If $k \in \{1, 9\}$, then the query to $\mathrm{P}/\mathrm{P}^{-1}$ is such that the triple $(\uparrow, x_{10}, x_{11})$ or $(\downarrow, x_0, x_1)$ is already in p. This is because CheckFwd/CheckBwd must have returned true for these chains to have been enqueued. If $k = 5$, then the partial chain resulting from a query to $\mathrm{P}/\mathrm{P}^{-1}$ is added to the set MidEquivChains and chains belonging to that set are not enqueued by definition of the simulator.

In case 2, the partial chain $(x_1, x_2, 1)$ resulting from a query to $\mathrm{P}/\mathrm{P}^{-1}$ is immediately added to CompletedChains by definition of the simulator. So, if $(x_1, x_2, 1) \notin$ CompletedChains when it is dequeued, the call to $\mathrm{P}/\mathrm{P}^{-1}$ is due to a query made by the distinguisher. We also claim that there is a unique distinguisher query corresponding to the chain $(x_1, x_2, 1)$. Say this is not the case and there is another chain $(x'_1, x'_2, 1)$ that has been enqueued such that $\textsc{CheckFwd}(x'_2 \oplus G_1(x'_1), x'_1, x'_{10}) = \mathsf{true}$ due to the same distinguisher query. If $\mathcal{D}$'s query was $(\downarrow, x_0, x_1)$ that returned (x_{10}, x_{11}), we have that $x'_{10} = x_{10}, x_1 = x'_1$ and $G_1(x'_1) \oplus x'_2 = x_2 \oplus G_1(x_1)$, giving $x_2 = x'_2$. If $\mathcal{D}$'s query was $(\uparrow, x_{10}, x_{11})$ that returned (x_0, x_1), we have the same result again. So, there is a unique distinguisher query corresponding to the chain $(x_1, x_2, 1)$. Since $\mathcal{D}$ makes at most q queries, we have that there are most q such partial chains dequeued. □

The following can be proved similarly to Claim 3.7.

Claim 3.8. *In H_2, the simulator dequeues from the queue Q_{enq} a partial chain of the form $(x_9, x_{10}, 9, \ell, g, b)$ such that $(x_9, x_{10}, 9) \notin$ CompletedChains at most q times in total.*

For a hashtable G we let $|G|$ denote the number of entries in G.

Claim 3.9. *In H_2, at most $90q^2$ partial chains of the form $(x_5, x_6, 5)$ are enqueued.*

Proof. Before we bound the number of partial chains of the form $(x_5, x_6, 5)$, we will bound the size of $|G_5|$ and $|G_6|$. The size of G_5 can increase only in the following ways: (a) the distinguisher queries $\mathrm{F}(5, \cdot)$, (b) during the completion of a $(x_9, x_{10}, 9)$ partial chain, (c) during the completion of a $(x_1, x_2, 1)$ partial chain, or (d) during the completion of a $(x_5, x_6, 5)$ partial chain where $x_5 \notin G_5$ and $x_5 \in A_5^j$.

There are at most q queries that the distinguisher makes. The simulator only completes $(x_9, x_{10}, 9)$ such that $(x_9, x_{10}, 9) \notin \mathsf{CompletedChains}$. By Claim 3.7, we have that there are at most q such chains dequeued from Q_{enq} and hence at most q such chains can be dequeued from the queues Q_1, Q_5, Q_6, Q_{10} and Q_{all}. Similarly, by Claim 3.8, we have that there are at most q completions of a partial chain of the form $(x_1, x_2, 1)$.

For the last case, a value $x_5 \in \{0,1\}^n$ gets added to A_5^j for $j = 1, \ldots, q$ only when $x_5 \notin G_5$. So, a bound on $\sum_{i=1}^{j} A_5^j$ gives a bound on the number of values x_5 added to G_5 due to a completion of a $(x_5, x_6, 5)$ partial chain. A value $x_5 \in \{0,1\}^n$ gets added to A_5^j for $j = 1, \ldots, q$ only in one of the following ways: (a) the distinguisher queried $\mathrm{F}(5, x_5)$ (b) the simulator made a call to EvalFwdEnq or EvalBwdEnq during the completion of a $(x_9, x_{10}, 9)$ chain where $(x_9, x_{10}, 9) \notin$ $\mathsf{CompletedChains}$ (c) the simulator made a call to EvalFwdEnq or EvalBwdEnq during the completion of a $(x_1, x_2, 1)$ chain where $(x_1, x_2, 1) \notin \mathsf{CompletedChains}$. Thus, we have $\sum_{i=1}^{j} A_5^j \leq 3q$ by Claims 3.7 and 3.8. So, putting everything together, we have that $|G_5| \leq 6q$ and by a similar argument, $|G_6| \leq 6q$.

Now, the number of partial chains of the form $(x_5, x_6, 5)$ that are enqueued in Q_{all} can be bounded by $\sum_{j=1}^{q} |A_5^j| \cdot |G_6 \cup A_6^j| + \sum_{j=1}^{q} |G_5 \cup A_5^j| \cdot |A_6^j|$ and hence can be bounded by $54q^2$. And, the number of partial chains of the form $(x_5, x_6, 5)$ that are enqueued in Q_{mid} can be bounded by $|G_5| \cdot |G_6|$ and hence can be bounded by $36q^2$. □

Claim 3.10. *In H_2, we have $|G_i| \leq 93q^2$ for all i.*

Proof. The size of G_i can only increase in the following ways: (a) the distinguisher queries $\mathrm{F}(i, \cdot)$, (b) during the completion of a $(x_9, x_{10}, 9)$ partial chain, (c) during the completion of a $(x_1, x_2, 1)$ partial chain, or (d) during the completion of a $(x_5, x_6, 5)$ partial chain.

There are at most q distinguisher queries and at most q completions each of partial chains of the form $(x_9, x_{10}, 9)$ and $(x_1, x_2, 1)$ by Claims 3.7 and 3.8. There are at most $90q^2$ completions of a $(x_5, x_6, 5)$ partial chain by Claim 3.9. So, $|G_i| \leq 90q^2 + 3q \leq 93q^2$. □

Claim 3.11. *In H_2, the simulator makes at most $644q^2$ queries to $R.\mathrm{P}$ and $R.\mathrm{P}^{-1}$.*

Proof. The simulator makes calls to $R.\mathrm{P}/R.\mathrm{P}^{-1}$ only in procedures EvalFwdEnq, EvalBwdEnq, EvalFwdComp, and EvalBwdComp. By the bounds on the number of partial chains that are enqueued for completion given in Claims 3.7, 3.8, and 3.9 and by definition of the simulator, the number of calls to $\mathrm{P}/\mathrm{P}^{-1}$ is bounded by $92q^2 \cdot 7 = 644q^2$. □

Claim 3.12. *In H_2, the simulator makes at most $3,166,000q^6$ queries to the procedures R.CheckFwd and R.CheckBwd.*

Proof. The number of $\textsc{CheckFwd}$ queries made by the simulator is bounded by $\sum_{j=1}^{q} |(G_{10} \cup A_{10}^{j}| \times |G_1| \times |A_2^{j}|) + \sum_{j=1}^{q} |(G_2 \cup A_2^{j}| \times |G_1| \times |A_{10}^{j}|)$.

A value $x_{10} \in \{0,1\}^n$ gets added to A_{10}^{j} for $j = 1, \ldots, q$ only in one of the following ways: (a) the distinguisher queried $\mathrm{F}(10, x_{10})$, (b) the simulator made a call to $\textsc{EvalFwdEnq}$ or $\textsc{EvalBwdEnq}$ during the completion of a $(x_1, x_2, 1)$ chain where $(x_1, x_2, 1) \notin \mathsf{CompletedChains}$, or (c) the simulator made a call to $\textsc{EvalFwdEnq}$ or $\textsc{EvalBwdEnq}$ during the completion of a $(x_5, x_6, 5)$ chain.

There are at most q distinguisher queries, and by Claims 3.8 and 3.9, there are at most q and $90q^2$ such calls to $\textsc{EvalFwdEnq}$ and $\textsc{EvalBwdEnq}$. So, $\sum_{j=1}^{q} A_{10}^{j} \leq 90q^2 + 2q \leq 92q^2$. In a similar manner, we can bound $\sum_{j=1}^{q} A_2^{j} \leq 90q^2 + 2q \leq 92q^2$. So, the number of $\textsc{CheckFwd}$ queries made by the simulator is bounded by $1582860q^6$. By a similar argument, the number of $\textsc{CheckBwd}$ queries made by the simulator is bounded by $1582860q^6$. Hence, there are at most $O(q^6)$ queries to $R.\textsc{CheckFwd}$ and $R.\textsc{CheckBwd}$. □

Corollary 3.13. *In $H_{1.5}$ and H_1, the simulator makes at most $3.2 \times 10^6 q^6$ queries to P^+ and P respectively, except with probability at most $\frac{10^{15} q^{12}}{2^n}$.*

Proof. For the sake of contradiction, assume that there exists a distinguisher $\mathcal{D}$ that makes at most q queries such that in $H_{1.5}$ the simulator makes more than $3.2 \times 10^6 q^6$ queries to P^+ with probability greater than $\frac{10^{15} q^{12}}{2^n}$. By Claims 3.11 and 3.12, the simulator makes at most $3,166,644q^6$ queries to R in H_2. Now, consider a distinguisher $\mathcal{S}'$ that consists of $\mathcal{D}$ and the simulator together. $\mathcal{S}'$ outputs 1 if $\mathcal{D}$ and the simulator make more than $3.2 \times (10q)^6$ queries. Then, $\mathcal{S}'$ issues at most

$q' = 3.2 \times (10q)^6$ queries and distinguishes R from P^+ with probability greater than $\frac{10^{15}q^{12}}{2^n} \geq \frac{12\cdot(3.2\times(10q)^6)^2}{2^n}$ which contradicts Theorem 3.3. Hence, the simulator makes at most $3.2 \times 10^6 q^6$ queries to P^+ except with probability $\frac{10^{15}q^{12}}{2^n}$. Combining this with Lemma 3.2, we get the result. □

Lemma 3.14. *The probability that $\mathcal{D}$ outputs 1 in H_1 differs from the probability that it outputs 1 in H_2 by at most $\frac{2\cdot 10^{15}q^{12}}{2^n}$.*

Proof. By Lemma 3.2, we have that the distinguisher $\mathcal{D}$ outputs 1 in H_1 and $H_{1.5}$ with the same probability. Now, for the sake of contradiction, assume that $\mathcal{D}$ distinguishes $H_{1.5}$ from H_2 with advantage greater than $\frac{2\cdot 10^{15}q^{12}}{2^n}$. We construct a distinguisher $\mathcal{S}'$ that consists of $\mathcal{D}$ and the simulator where $\mathcal{S}'$ makes at most $3.3 \times (10q)^6$ queries and distinguishes R from P^+. $\mathcal{S}'$ works as follows: $\mathcal{S}'$ outputs 1 if the number of queries issued by $\mathcal{D}$ and the simulator exceeds $3.3 \times (10q)^6$; otherwise, it outputs whatever is output by $\mathcal{D}$. By Corollary 3.13, the simulator makes at most $3.2 \times (10q)^6$ queries in $H_{1.5}$ except with probability $\frac{10^{15}q^{12}}{2^n}$ and by Claims 3.11 and 3.12, the simulator makes at most $3,166,644q^6$ queries in H_2. So, distinguisher $\mathcal{S}'$ making at most $3.3 \times (10q)^6$ queries can distinguish R and P^+ with probability greater than $\frac{10^{15}q^{12}}{2^n} \geq \frac{12\cdot(3.3\times(10q)^6)^2}{2^n}$ which contradicts Theorem 3.3. □

By extending the analysis, we also bound the overall running time of the simulator.

Claim 3.15. *In H_2, the simulator runs in time $O(q^6)$.*

Proof. We analyze the running time of the procedures of $\hat{\mathcal{S}}$. Procedure $\mathrm{F}^{\mathrm{ENQ}}$ runs in time $O(1)$ except for the run-time of procedure EnqNewChains. Procedures

EvalFwdEnq and EvalBwdEnq run in time $O(1)$ except for the run-time of procedure $\mathrm{F}^{\mathrm{ENQ}}$. Procedures EvalFwdComp and EvalBwdComp run in time $O(1)$ except for the run-time of procedure $\mathrm{F}^{\mathrm{COMP}}$. Procedure Adapt runs in time $O(1)$ except for the run-time of procedures $\mathrm{F}^{\mathrm{COMP}}$ and EnqNewMidChains. Procedure $\mathrm{F}^{\mathrm{COMP}}$ runs in time $O(1)$ except for the run-time of procedure EnqNewMidChains. Procedures CheckFwd, CheckBwd, and ForceVal run in time $O(1)$.

Procedure EnqNewChains runs in time $O(q^4)$ by the bounds on $|G_i|$ given by Claim 3.10 and by the bounds on $\sum_{j=1}^{q} |A_i^j|$ for $i \in \{1, 2, 5, 6, 9, 10\}$ derived in Claims 3.9 and 3.12. Procedure EnqNewMidChains runs in time $O(q^2)$ by the bounds on $|G_5|$ and $|G_6|$ derived in Claim 3.9.

Calls to $\mathrm{F}^{\mathrm{ENQ}}$ are made either by a direct query by the distinguisher, or in calls to EvalFwdEnq or EvalBwdEnq. There are at most $O(q^2)$ calls to EvalFwdEnq and EvalBwdEnq by the bounds established on the number of partial chains enqueued in Q_{enq} given by Claims 3.7, 3.8, and 3.9. So, the maximum number of calls to $\mathrm{F}^{\mathrm{ENQ}}$ is bounded by $O(q^2)$. Calls to EnqNewChains are made only during calls to $\mathrm{F}^{\mathrm{ENQ}}$. So, there are at most $O(q^2)$ calls to EnqNewChains. Calls to CheckFwd and CheckBwd are only made in EnqNewChains - so there are at most $O(q^6)$ calls to CheckFwd and CheckBwd.

There are at most $O(q^2)$ calls to EvalFwdComp, EvalBwdComp, and Adapt by the bounds on the number of partial chains enqueued in Q_{enq} and Q_{mid} given by Claims 3.7, 3.8, and 3.9. There are at most $O(q^2)$ calls to $\mathrm{F}^{\mathrm{COMP}}$ since $\mathrm{F}^{\mathrm{COMP}}$ is called only in EvalFwdComp, EvalBwdComp, Adapt, or as a result of a direct distinguisher query. Calls to EnqNewMidChains are made only during calls to $\mathrm{F}^{\mathrm{COMP}}$ and Adapt. So, there are at most $O(q^2)$ calls to EnqNewMidChains. There are at most $O(q^2)$ calls to ForceVal since ForceVal is called only during Adapt. Putting all of this together, the simulator runs in time $O(q^6)$. □

Corollary 3.16. *In $H_{1.5}$ (and hence H_1), the simulator runs for at most $O(q^6)$ steps and makes at most $3.2 \times (10q)^6$ queries except with probability at most $\frac{10^{15}q^{12}}{2^n}$.*

Proof. By Claims 3.11 and 3.12, the simulator makes at most $3.2 \times (10q)^6$ queries to R in H_2 and by Claim 3.15, runs in time at most $r(q) \in O(q^6)$ in H_2. For the sake of contradiction, assume that there exists a distinguisher $\mathcal{D}$ that makes at most q queries such that in $H_{1.5}$ the simulator runs in time greater than $r(q)$ or makes more than $3.2 \times (10q)^6$ queries to P^+ with probability greater than $\frac{10^{15}q^{12}}{2^n}$. Now, consider a distinguisher $\mathcal{S}'$ which aims to distinguish P^+ and R making only $r'(q)$ queries. The distinguisher $\mathcal{S}'$ consists of $\mathcal{D}$ and the simulator together. $\mathcal{S}'$ outputs 1 if the simulator runs for more than $r(q)$ steps or if $\mathcal{D}$ and the simulator make more than $3.3 \times (10q)^6$ queries. Then, $\mathcal{S}'$ issues at most $3.3 \times (10q)^6$ queries and distinguishes R from P^+ with probability greater than $\frac{10^{15}q^{12}}{2^n} \geq \frac{12\cdot(3.3\times(10q)^6)^2}{2^n}$ which contradicts Theorem 3.3. Combining this with Lemma 3.2, we get the result. □

3.4.3 Properties of the Second Experiment

Before we define the third hybrid experiment, we introduce some definitions and establish some properties of executions in the second experiment H_2. The definitions here follow closely along the lines of the definitions in [17]. A *partial* *chain* is a triple $(x_k, x_{k+1}, k) \in \{0,1\}^n \times \{0,1\}^n \times \{0, \ldots, 10\}$. If $C = (x_k, x_{k+1}, k)$ is a partial chain, we let $C[1] = x_k$, $C[2] = x_{k+1}$, and $C[3] = k$.

Definition 3. *Fix tables $G = \hat{\mathcal{S}}.G$ and $p = R.p$ in an execution of H_2, and let $C = (x_k, x_{k+1}, k)$ be a partial chain. We define functions* $\mathsf{next}, \mathsf{prev}, \mathsf{val}^+, \mathsf{val}^-$*, and* val *as follows:*

```
procedure $\mathsf{next}(x_k, x_{k+1}, k)$:
    if $k < 10$ then
        if $x_{k+1} \notin G_{k+1}$ then return $\perp$
        $x_{k+2} := x_k \oplus G_{k+1}(x_{k+1})$
        return $(x_{k+1}, x_{k+2}, k+1)$
    else if $k = 10$ then
        if $(\uparrow, x_{10}, x_{11}) \notin p$ then return $\perp$
        $(x_0, x_1) := p(\uparrow, x_{10}, x_{11})$
        return $(x_0, x_1, 0)$

procedure $\mathsf{prev}(x_k, x_{k+1}, k)$:
    if $k > 0$ then
        if $x_k \notin G_k$ then return $\perp$
        $x_{k-1} := x_{k+1} \oplus G_k(x_k)$
        return $(x_{k-1}, x_k, k-1)$
    else if $k = 0$ then
        if $(\downarrow, x_0, x_1) \notin p$ then return $\perp$
        $(x_{10}, x_{11}) := p(\downarrow, x_0, x_1)$
        return $(x_{10}, x_{11}, 10)$

procedure $\mathsf{val}_i^+(C)$:
    while $(C \neq \perp) \wedge (C[3] \notin \{i-1, i\})$ do
        $C := \mathsf{next}(C)$
    if $C = \perp$ then return $\perp$
    if $C[3] = i$ then return $C[1]$
    else return $C[2]$

procedure $\mathsf{val}_i^-(C)$:
    while $(C \neq \perp) \wedge (C[3] \notin \{i-1, i\})$ do
        $C := \mathsf{prev}(C)$
    if $C = \perp$ then return $\perp$
    if $C[3] = i$ then return $C[1]$
    else return $C[2]$
```

```
procedure $\mathsf{val}_i(C)$:
    if $\mathsf{val}_i^+(C) \neq \perp$ then return $\mathsf{val}_i^+(C)$
    else return $\mathsf{val}_i^-(C)$
```

We say that $\perp \notin G_i$ for $i \in \{1, \ldots, 10\}$. So, if $\mathsf{val}_i(C) \notin G_i$, then either $\mathsf{val}_i(C) = \perp$ or $\mathsf{val}_i(C) \neq \perp$ and $\mathsf{val}_i(C) \notin G_i$.

Definition 4. *For a given set of tables G, p, two partial chains C, D are* equivalent *(denoted $C \equiv D$) if they are in the reflexive, transitive closure of the relations given by* next *and* prev*.*

So, two chains C and D are equivalent if $C = D$, or if D can be obtained by applying next and prev finitely many times to C.

Definition 5. *The set of* table-defined *chains contains all chains C for which* $\mathsf{next}(C) \neq \perp$ *and* $\mathsf{prev}(C) \neq \perp$*.*

Definition 6. *A chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ is called an* enqueued *chain if C is enqueued for completion. For such an enqueued chain, we define* $\mathsf{next}(C)$ *as the procedure* next *applied to the partial chain (x_k, x_{k+1}, k), i.e.,* $\mathsf{next}(C) := \mathsf{next}(x_k, x_{k+1}, k)$*. The procedures* prev*,* val^+*,* val^-*, and* val *on an enqueued chain C are defined in a similar manner.*

Definition 7. *The set Q^*_{all} contains chains that are enqueued in Q_{all} but not in Q_1, Q_5, Q_6, or Q_{10}.*

Definition 8. *We say that a* uniform assignment to *$G_k(x_k)$ occurs when the simulator sets $G_k(x_k)$ through an assignment $G_k(x_k) \leftarrow \{0, 1\}^n$, i.e., a uniform value is* *chosen from the set of n-bit strings and $G_k(x_k)$ is assigned that value.*

A uniform assignment to $G_k(x_k)$ occurs in line 125 of the simulator's execution. In particular, if $G_k(x_k)$ is set through a ForceVal$(x_k, \cdot, k)$ call, then it is not a uniform assignment.

Definition 9. *We say that a* uniform assignment to p *occurs in a call to* $R.\mathrm{P}(x_0, x_1)$ *if* $(\downarrow, x_0, x_1) \notin p$ *when the call is made and* $p(\downarrow, x_0, x_1)$ *is set through the assignment* $p(\downarrow, x_0, x_1) := (x_{10}, x_{11})$ *where* (x_{10}, x_{11}) *is chosen uniformly from the set of* $2n$*-bit strings.*

Similarly, it occurs in a call to $R.\mathrm{P}^{-1}(x_{10}, x_{11})$ *if* $(\uparrow, x_{10}, x_{11}) \notin p$ *when the call is made and* $p(\uparrow, x_{10}, x_{11})$ *is set through the assignment* $p(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$ *where* (x_0, x_1) *is chosen uniformly from the set of* $2n$*-bit strings.*

A uniform assignment to $p(\downarrow, x_0, x_1)$ occurs in line 4 of R in Figure 2 and a uniform assignment to $p(\uparrow, x_{10}, x_{11})$ occurs in line 10 of R in Figure 2.

In the remainder of the section, we let $T = O(q^2)$ be an upper bound on the sizes of G_i and p as well as the upper bound on the number of enqueued chains and hence the number of calls to the Adapt procedure in an execution of H_2. A bound on T is derived from Claims 3.7–3.11.

Bad executions. We define a set of "bad" events, and show that these occur with negligible probability. Following that, we analyze execution of the experiment assuming that none of these bad events occur.

Definition 10. *We say that event* BadP *occurs in* H_2 *if either:*

- *Immediately after choosing* (x_{10}, x_{11}) *in a call to* $R.\mathrm{P}(\cdot,\cdot)$, *either* $(\uparrow, x_{10}, x_{11}) \in p$ *or* $x_{10} \in G_{10}$.

- *Immediately after choosing* (x_0, x_1) *in a call to* $R.\mathrm{P}^{-1}(\cdot,\cdot)$, *either* $(\downarrow, x_0, x_1) \in p$ *or* $x_1 \in G_1$.

Lemma 3.17. *The probability of event* BadP *in* H_2 *is at most* $2\,T^2/2^n$.

Proof. The proof follows exactly as in [17, Lemma 3.18]. □

Definition 11. *We say that event* $\mathsf{BadlyHit}^+$ *occurs in* H_2 *if either:*

- *Immediately after a uniform assignment to* $G_k(x_k)$, *there is a partial chain* (x_k, x_{k+1}, k) *such that* $\mathsf{prev}(\mathsf{prev}(x_k, x_{k+1}, k)) \neq \perp$.

- *Immediately after a uniform assignment to* $G_k(x_k)$, *there is a partial chain* $(x_{k-1}, x_k, k-1)$ *such that* $\mathsf{next}(\mathsf{next}(x_{k-1}, x_k, k-1)) \neq \perp$.

and the relevant partial chain is either table-defined or an enqueued chain in Q_{all}.

Lemma 3.18. *The probability of event* $\mathsf{BadlyHit}^+$ *in* H_2 *is at most* $40\,T^3/2^n$.

Proof. Consider the case where a uniform assignment to $G_k(x_k)$ occurs and immediately after the assignment, there exists an enqueued chain $C = (x_k, x_{k+1}, k)$ in Q_{all} such that $\mathsf{prev}(\mathsf{prev}(x_k, x_{k+1}, k)) \neq \perp$. For this to occur, $x_{k-1} := x_{k+1} \oplus G_k(x_k)$ should take one of T values (for $k \in \{2, \ldots, 10\}$, x_{k-1} should be such that $x_{k-1} \in G_{k-1}$ and for $k = 1$, $(\downarrow, x_{k-1}, x_k)$ should be in p). The probability of this is at most $T/2^n$. The analysis for the case where C is table-defined is exactly the same. There are at most T enqueued chains and at most T options for x_{k+1} such that C is table-defined. So, the total probability of the first case is $2T^2/2^n$. The second case can be analyzed in a similar fashion. So, the total probability of $\mathsf{BadlyHit}^+$ for a uniform assignment $G_k(x_k)$ is $4T^2/2^n$. Since there can be at most $10T$ such assignments, the probability of $\mathsf{BadlyHit}^+$ is $40T^3/2^n$. □

Definition 12. *We say that event* $\mathsf{BadlyCollide}^+$ *occurs in* H_2 *if a uniform assignment to* $G_i(x_i)$ *is such that there exist two partial chains* C *and* D *such that for some* $\ell \in \{0,\ldots,11\}$ *and* $\sigma, \rho \in \{+,-\}$ *all the following are true:*

- *Immediately before the assignment,* C *and* D *are not equivalent.*
- *Immediately before the assignment,* $\mathsf{val}^\sigma_\ell(C) = \perp$ *or* $\mathsf{val}^\rho_\ell(D) = \perp$.
- *Immediately after the assignment,* $\mathsf{val}^\sigma_\ell(C) = \mathsf{val}^\rho_\ell(D) \neq \perp$.

and one of the following is true:

- *Immediately after the assignment,* C *and* D *are table-defined.*
- *Immediately after the assignment,* C *is table-defined and* D *is a chain enqueued in* Q_{all}.
- C *and* D *are chains enqueued in* Q_{all}.

Lemma 3.19. *The probability of event* $(\mathsf{BadlyCollide}^+ \wedge \neg\mathsf{BadlyHit}^+ \wedge \neg\mathsf{BadP})$ *in* H_2 *is at most* $21160\,T^5/2^n$.

Proof. Let C and D be the partial chains in Definition 12.

Case 1: After the assignment C and D are table-defined. The proof for this case follows exactly as in [17, Lemma 3.21].

Case 2: C is a chain enqueued in Q_{all} and D is table-defined after the assignment. Consider the case that the enqueued chain C is of the form (x_k, x_{k+1}, k) where $x_k \in A_k^j$ and $x_{k+1} \in A_{k+1}^j$ for some $j \in \{1, \ldots, q\}$. Let $\mathsf{val}_\ell^-(C) = \mathsf{val}_\ell^-(D) =\perp$ before the assignment and $\mathsf{val}_\ell^-(C) = \mathsf{val}_\ell^-(D) \neq\perp$ after the assignment. For $\mathsf{val}_\ell^-(C)$ to change, we must have $i = k$. Since $\mathsf{BadlyHit}^+$ does not occur, we have that $\mathsf{val}_{\ell+1}^-(C) = \mathsf{val}_{\ell+1}^-(D)$ and $\ell + 1 = k$. But, since C is not equivalent to D before the assignment, it cannot be the case that $\mathsf{val}_\ell^-(C) = \mathsf{val}_\ell^-(D) \neq\perp$ after the assignment and hence this case has probability 0.

Let $\mathsf{val}_\ell^-(C) =\perp$ and $\mathsf{val}_\ell^-(D) \neq\perp$ before the assignment and $\mathsf{val}_\ell^-(C) = \mathsf{val}_\ell^-(D) \neq\perp$ after the assignment. For $\mathsf{val}_\ell^-(C)$ to change $i = k$. Since $\mathsf{BadlyHit}^+$ does not occur and $\mathsf{val}_\ell^-(D)$ does not change due to the assignment, we have that $x_{k+1} \oplus G_k(\mathsf{val}_k^-(C)) = \mathsf{val}_{k-1}^-(D)$ where $\ell + 1 = k$. The probability of this is $1/2^n$.

Let $\mathsf{val}_\ell^+(C) = \mathsf{val}_\ell^-(D) =\perp$ before the assignment and $\mathsf{val}_\ell^+(C) = \mathsf{val}_\ell^-(D) \neq\perp$ afterward. For $\mathsf{val}_\ell^+(C)$ to change after the assignment, it must be the case that $\ell = k + 2$ and $i = k + 1$ since $\mathsf{BadlyHit}^+$ does not occur. But for $\mathsf{val}_\ell^-(D)$ to change when $\ell = k+2$, we need $i = k+3$ since $\mathsf{BadlyHit}^+$ does not occur as D is table-defined after the assignment. So, we see that $\mathsf{val}_\ell^+(C)$ cannot change in this case.

Let $\mathsf{val}_\ell^+(C) =\perp$ and $\mathsf{val}_\ell^-(D) \neq\perp$ before the assignment and $\mathsf{val}_\ell^+(C) = \mathsf{val}_\ell^-(D) \neq\perp$ after the assignment. For $\mathsf{val}_\ell^+(C)$ to change $i = k + 1$. Since $\mathsf{BadlyHit}^+$ does not occur and $\mathsf{val}_\ell^-(D)$ does not change due to the assignment, we have that $x_k \oplus G_{k+1}(\mathsf{val}_{k+1}^+(C)) = \mathsf{val}_{k+2}^-(D)$ where $\ell = k + 2$. The probability of this is $1/2^n$.

The remaining four cases follow similarly. The case for $x_k \in G_k, x_{k+1} \in A_{k+1}^j$, and $x_k \in A_k^j, x_{k+1} \in G_{k+1}$, and $x_k \in G_k, x_{k+1} \in G_{k+1}$ where $C = (x_k, x_{k+1}, k)$ is the enqueued chain, follow in a similar fashion. So, the total probability of this event can be bounded as follows. There are at most T enqueued chains and at most $11T^2$ table-defined chains before the assignment and there are at most $2T$ chains that were not table-defined before the assignment but were table-defined after. There are at most $10T$ uniform assignments to $G_i(x_i)$ and there are at most 4 possibilities for σ, ρ. Thus, the probability is at most $10T \cdot T \cdot (11T^2 + 2T) \cdot 4 \cdot 4/2^n \leq 2080T^4/2^n$.

Case 3: C and D are chains enqueued in Q_{all}. The proof follows similar to the proof of Case 2. □

Definition 13. *We say that event* BadlyCollideP *occurs in H_2 if either:*

- *A uniform assignment $p(\downarrow, x_0, x_1) := (x_{10}, x_{11})$ is such that there exist partial chains C and D such that for some $\sigma, \rho \in \{+, -\}$ the following are all true:*
  - *Immediately before the assignment, C and D are not equivalent.*
  - *Immediately before the assignment, $\mathsf{val}^{\sigma}_{10}(C) = \perp$ or $\mathsf{val}^{\rho}_{10}(D) = \perp$.*
  - *Immediately after the assignment, $\mathsf{val}^{\sigma}_{10}(C) = \mathsf{val}^{\rho}_{10}(D) = x_{10} \neq \perp$.*

  and one of the following conditions hold:

  - *Before the assignment, C and D are chains in Q^*_{all}.*
  - *Immediately after the assignment, C and D are table-defined.*
  - *Before the assignment, C is a chain enqueued in Q_{all} and immediately after the assignment, D is table-defined.*

- *A uniform assignment $p(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$ is such that there exist two* *partial chains C and D such that for some $\sigma, \rho \in \{+, -\}$ the following are all true:*

- *Immediately before the assignment, C and D are not equivalent.*
- *Immediately before the assignment, $\mathsf{val}_1^{\sigma}(C) = \perp$ or $\mathsf{val}_1^{\rho}(D) = \perp$.*
- *Immediately after the assignment, $\mathsf{val}_1^{\sigma}(C) = \mathsf{val}_1^{\rho}(D) = x_1 \neq \perp$.*

and one of the following conditions hold:

- *Before the assignment, C and D are chains in Q^*_{all}.*
- *Immediately after the assignment, C and D are table-defined.*
- *Before the assignment, C is a chain enqueued in Q_{all} and immediately after the assignment, D is table-defined.*

Lemma 3.20. *The probability of event* $\mathsf{BadlyCollideP}$ *in H_2 is at most $314T^5/2^n$.*

Proof. Consider the case that after a uniform choice of (x_0, x_1) leading to an assignment $p(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$, the event $\mathsf{BadlyCollideP}$ occurs. The value $\mathsf{val}_1^-(C)$ for a chain C does not change due to the assignment since it is a $p(\uparrow, x_{10}, x_{11})$ assignment and $\mathsf{val}_1^-(C)$ can change only due to a $p(\downarrow, x_0, x_1)$ assignment by definition of $\mathsf{val}^-(\cdot)$.

Suppose that $\mathsf{val}_1^+(C) = \perp$ and $\mathsf{val}_1^-(D) \neq \perp$ before the assignment, and after the assignment $\mathsf{val}_1^+(C) = \mathsf{val}_1^-(D) = x_1$. The value $\mathsf{val}_1^-(D)$ does not change due to the assignment as mentioned above. So, the probability that $\mathsf{val}_1^+(C) = \mathsf{val}_1^-(D) = x_1$ is 2^{-n}.

Suppose that $\mathsf{val}_1^+(C) = \mathsf{val}_1^+(D) = \perp$ before the assignment and after the assignment $\mathsf{val}_1^+(C) = \mathsf{val}_1^+(D) = x_1$. For this to happen, $\mathsf{val}_{10}(C) = \mathsf{val}_{10}(D) = x_{10}$ and $\mathsf{val}_{11}(C) = \mathsf{val}_{11}(D) = x_{11}$ implying that C and D are equivalent chains. So, the probability of this event is 0.

Suppose that $\mathsf{val}_1^+(C) = \perp$ and $\mathsf{val}_1^+(D) \neq \perp$ before the assignment and after the assignment $\mathsf{val}_1^+(C) = \mathsf{val}_1^+(D) = x_1$. Now, the value of $\mathsf{val}_1^+(D)$ stays the same after the assignment (even if BadP occurs). So, the probability that $\mathsf{val}_1^+(C) = \mathsf{val}_1^+(D) = x_1$ is 2^{-n}.

The analysis for the other case follows similarly. There are at most T assignments of the form $p(\uparrow, x_{10}, x_{11})$ or $p(\downarrow, x_0, x_1)$. There are at most $11T^2$ possibilities for a chain to be table-defined before the assignment and T possibilities for a chain to be table-defined after the assignment but not before. There are at most T chains enqueued for completion in Q_{all}. So, the probability of the event $\mathsf{BadlyCollideP}$ is at most $\frac{T\cdot((11T^2+T)^2+T^2+T\cdot(11T^2+T))\cdot 2}{2^n}$. □

Definition 14. *We say that event* $\mathsf{BadlyHitFV}$ *occurs in* H_2 *if in a uniform assignment to* $G_s(x_s)$ *that occurs in a call to* $\textsc{Adapt}(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$*, for some* $s \in \{g, b\}$*, one of the following happens (where we let* $C = (x_{\ell-2}, x_{\ell-1}, \ell-2)$*):*

- $s = \ell + 2$ *and the following holds:*
  - *Immediately before the assignment,* $\mathsf{val}_{\ell+1}^-(C) = \perp$*.*
  - *Immediately after the assignment,* $\mathsf{val}_{\ell+1}^-(C) \neq \perp$*.*
  - *Immediately after the assignment,* $y := \mathsf{val}_{\ell-1}(C) \oplus \mathsf{val}_{\ell+1}^-(C)$ *is such that* $x'_{\ell+1} \oplus x'_{\ell-1} = y$ *for some* $x'_{\ell+1} \in G_{\ell+1}$ *and* $x'_{\ell-1} \in G_{\ell-1}$*.*

- *$s = \ell - 1$ and the following holds:*
  - *Immediately before the assignment, $\mathsf{val}^+_\ell(C) =\perp$.*
  - *Immediately after the assignment, $\mathsf{val}^+_\ell(C) \neq\perp$.*
  - *Immediately after the assignment, $y := \mathsf{val}_{\ell+2}(C) \oplus \mathsf{val}^+_\ell(C)$ is such that $x'_{\ell+2} \oplus x'_\ell = y$ for some $x'_{\ell+2} \in G_{\ell+2}$ and $x'_\ell \in G_\ell$.*

Lemma 3.21. *The probability of event* $\mathsf{BadlyHitFV}$ *in H_2 is at most $2T^3/2^n$.*

Proof. Consider the first case where $s = \ell+2$. Note that for a chain C with $s = \ell+2$ the "value" at the adapt position $\ell+1$ is set as $\mathsf{val}_{\ell+1}(C) := \mathsf{val}_{\ell+3}(C) \oplus G_s(\mathsf{val}_s(C))$ where $\mathsf{val}_{\ell+3}(C) \neq\perp$ is one of the arguments to ADAPT. Since the assignment to $G_s(x_s)$ happens inside the ADAPT call, $\mathsf{val}^-_{\ell+1}(C) =\perp$ until the assignment and $\mathsf{val}^-_{\ell+1}(C) \neq\perp$ immediately after the assignment.

Now, $y := \mathsf{val}_{\ell-1}(C) \oplus \mathsf{val}^-_{\ell+1}(C)$. Note that $\mathsf{val}_{\ell-1}(C) \neq\perp$ since $\mathsf{val}_{\ell-1}(C) = x_{\ell-1}$ is one of the arguments of the ADAPT procedure. So, for $y := \mathsf{val}_{\ell-1}(C) \oplus \mathsf{val}_{\ell+3}(C) \oplus G_s(\mathsf{val}_s(C))$ to be such that $y = x'_{\ell-1} \oplus x'_{\ell+1}$, where $x'_{\ell-1} \in G_{\ell-1}$ and $x'_{\ell+1} \in G_{\ell+1}$, the value y needs to take one of $T^2/2^n$ values. Note that there are at most T such calls to ADAPT by assumption. So, the probability of the first case is at most $T^3/2^n$.

The analysis for the second case is analogous. □

Definition 15. *We say that event* $\mathsf{BadlyCollideFV}$ *occurs in H_2 if in a uniform assignment to $G_s(x_s)$ that occurs in a call to* ADAPT$(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$, *for some $s \in \{g, b\}$, the following happens (where we let $C = (x_{\ell-2}, x_{\ell-1}, \ell-2)$ and*

D is a chain in Q^*_{all}):

- *$s = \ell + 2$, and for some $(k, k') \in \{(\ell - 1, \ell + 1), (\ell + 1, \ell - 1)\}$ the following holds:*
    - *Immediately before the assignment, $\mathsf{val}^-_{\ell+1}(C) = \perp$ and $\mathsf{val}_k(D) \neq \perp$.*
    - *Immediately after the assignment, $\mathsf{val}^-_{\ell+1}(C) \neq \perp$.*
    - *Immediately after the assignment, $y := \mathsf{val}_{\ell-1}(C) \oplus \mathsf{val}^-_{\ell+1}(C)$ is such that $x \oplus y = \mathsf{val}_k(D)$ for some $x \in G_{k'}$.*
- *$s = \ell - 1$, and for some $(k, k') \in \{(\ell, \ell + 2), (\ell + 2, \ell)\}$ the following holds:*
    - *Immediately before the assignment, $\mathsf{val}^+_{\ell}(C) = \perp$ and $\mathsf{val}_k(D) \neq \perp$.*
    - *Immediately after the assignment, $\mathsf{val}^+_{\ell}(C) \neq \perp$.*
    - *Immediately after the assignment, $y := \mathsf{val}_{\ell+2}(C) \oplus \mathsf{val}^+_{\ell}(C)$ is such that $x \oplus y = \mathsf{val}_k(D)$ for some $x \in G_{k'}$.*

Lemma 3.22. *The probability of event* $\mathsf{BadlyCollideFV}$ *in H_2 is at most $4\,T^3/2^n$.*

Proof. Consider the first case where $s = \ell + 2$. Note that during the ADAPT call the "value" at the adapt position $\ell + 1$ is set as $\mathsf{val}_{\ell+1}(C) := \mathsf{val}_{\ell+3}(C) \oplus G_s(\mathsf{val}_s(C))$ where $\mathsf{val}_{\ell+3}(C) \neq \perp$ is one of the arguments to ADAPT. Since the assignment to $G_s(x_s)$ happens inside the ADAPT call, $\mathsf{val}^-_{\ell+1}(C) = \perp$ until the assignment and $\mathsf{val}^-_{\ell+1}(C) \neq \perp$ immediately after the assignment.

Now, $y := \mathsf{val}_{\ell-1}(C) \oplus \mathsf{val}^-_{\ell+1}(C)$. Note that $\mathsf{val}_{\ell-1}(C) \neq \perp$ since it is one of the arguments of the ADAPT procedure. Also note that if $\mathsf{val}_k(D) \neq \perp$ before the assignment, then $\mathsf{val}_k(D)$ does not change due to the assignment. Say $k = \ell - 1$ and $k' = \ell{+}1$. So, for $y := \mathsf{val}_{\ell-1}(C) \oplus \mathsf{val}_{\ell+3}(C) \oplus G_s(x_s)$ to be such that $y = x \oplus \mathsf{val}_{\ell-1}(D)$ where $x \in G_{\ell+1}$, the value y would have to take one of $T^2/2^n$ values. Similarly for the case where $k = \ell + 1$ and $k' = \ell - 1$. So, for a single call to Adapt where $s = \ell + 2$, we have that the probability that the event occurs is $2T^2/2^n$. There are at most T calls to Adapt by assumption and hence, the probability of the first case is at most $2T^3/2^n$.

The analysis for the second case is analogous. □

We say that an execution of H_2 is *good* if none of the events BadP, $\mathsf{BadlyHit}^+$, $\mathsf{BadlyCollide}^+$, $\mathsf{BadlyCollideP}$, $\mathsf{BadlyHitFV}$ or $\mathsf{BadlyCollideFV}$ occur. Lemmas 3.17–3.22 thus imply:

Lemma 3.23. *The probability that an execution of H_2 is good is at least $1 - O(T^5)/2^n$.*

Properties of good executions. We prove that during a good execution of H_2, every call to ForceVal$(x, \cdot, a)$ is such that $x \notin G_a$, i.e., a ForceVal call does not "overwrite".

Before we proceed with the proofs, we introduce some notation. For a chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ that is enqueued for completion, the *adapt positions* are at ℓ, $\ell + 1$. These positions are those where the simulator uses ForceVal$(\cdot, \cdot, \ell)$ and ForceVal$(\cdot, \cdot, \ell + 1)$ to force the values at $G_\ell(\cdot)$ and $G_{\ell+1}(\cdot)$. Also, for the chain C, the "set uniform" positions are at $\ell - 1$, $\ell + 2$. (These are the buffer zones that surround the adapt positions.) One of these "set uniform" positions is adjacent to the query that caused the chain to be enqueued and this position is denoted by g and referred to as the "good" set uniform position. The other "set uniform" position is referred to as the "bad" set uniform position. Note that $g, b \in \{\ell-1, \ell+2\}$ and $g \neq b$; Let a be the adapt position that is adjacent to a "bad" set uniform position. So, if $b = \ell - 1$, then $a = \ell$; else, if $b = \ell + 2$, $a = \ell + 1$. Consider a call $\textsc{Adapt}(x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$. If $b = \ell - 1$: if $x_{\ell-1} \in G_{\ell-1}$, then x_a is defined as $x_a = x_\ell := x_{\ell-2} \oplus G_{\ell-1}(x_{\ell-1})$; else, x_a is defined as $x_a = x_\ell = \perp$. Else if $b = \ell + 2$: if $x_{\ell+2} \notin G_{\ell+2}$, then x_a is defined as $x_a = x_{\ell+1} := x_{\ell+3} \oplus G_{\ell+2}(x_{\ell+2})$; else, x_a is defined as $x_a = x_{\ell+1} = \perp$.

Also, for a chain C enqueued in Q_b we say *adapting is safe* if just before the call to $\textsc{Adapt}$ for C, we have $x_g \notin G_g$ and $x_a \notin G_a$. Analogously, for a chain C in Q^*_{all} or Q_{mid} we say *adapting is safe* if just before the call to $\textsc{Adapt}$ for C, we have $x_{\ell-1} \notin G_{\ell-1}$ and $x_{\ell+2} \notin G_{\ell+2}$. Also, we loosely use the statement $C \in \mathsf{CompletedChains}$ where $C = (x_k, x_{k+1}, k, \ell, g, b)$ to mean that $(x_k, x_{k+1}, k) \in \mathsf{CompletedChains}$.

To prove that $\textsc{ForceVal}$ does not "overwrite," we will prove that for every call to $\textsc{Adapt}$ that occurs during the completion of a chain $C = (x_k, x_{k+1}, k, \ell, g, b)$, we have $\mathsf{val}_g(C) \notin G_g$ before the call and if C is enqueued in Q_b, $\mathsf{val}_a(C) \notin G_a$ before the call; else, $\mathsf{val}_b(C) \notin G_b$ before the call. I.e., every call to $\textsc{Adapt}$ is "safe." In order to prove the above statements, we prove that at the time a chain C is enqueued in Q_{all}, $\mathsf{val}_g(C) = \perp$ and if C is a chain enqueued in Q_b for some $b \in \{1, 5, 6, 10\}$, then $\mathsf{val}_b(C) \notin G_b$; else, $\mathsf{val}_b(C) = \perp$ when C was enqueued. Similarly, if a chain C is enqueued in Q_{mid}, then we prove that $\mathsf{val}_g(C) = \perp$ and $\mathsf{val}_b(C) = \perp$ just before the assignment that precedes C being enqueued occurs. We also need to prove properties of equivalent chains in order to prove that if a chain equivalent to C has been completed before C, then $C \in \mathsf{CompletedChains}$ when it is dequeued. All of this put together will help us prove that FORCEVAL does not "overwrite" (Theorem 3.38). While the structure explained above is similar to the structure of the proof in [17], the major difference is in how we prove the properties of chains at the time they are enqueued. This is due to the fact that we separate enqueueing from completion in our simulation.

Properties of Equivalent Chains

Claim 3.24. *Consider a good execution of H_2. Suppose that at some point in the execution, two partial chains C and D are equivalent. Then there exists a sequence of partial chains $C_1, \ldots, C_r$ such that*

- *$C = C_1$ and $D = C_r$, or else $D = C_1$ and $C = C_r$,*
- *for $r \geq 2$, $C_i = \mathsf{next}(C_{i-1})$ and $C_{i-1} = \mathsf{prev}(C_i)$ for all $i \in \{2, \ldots, r\}$,*
- *for $r \geq 3$, $C_2, \ldots, C_{r-1}$ is table-defined,*
- *$D = (\mathsf{val}_j^{\rho}(C), \mathsf{val}_{j+1}^{\rho}(C), j)$ where $\mathsf{val}_j^{\rho}(C) \neq \perp$ and $\mathsf{val}_{j+1}^{\rho}(C) \neq \perp$ for some $\rho \in \{+, -\}$,*
- *$C = (\mathsf{val}_k^{\sigma}(D), \mathsf{val}_{k+1}^{\sigma}(D), k)$ where $\mathsf{val}_k^{\sigma}(D) \neq \perp$ and $\mathsf{val}_{k+1}^{\sigma}(D) \neq \perp$ for some $\sigma \in \{+, -\}$.*

Proof. By definition, $C \equiv D$ implies that we can apply next and prev finitely many times to get D from C. Since BadP does not occur, we have that the relation

$\equiv$ is symmetric and hence, $\equiv$ is an equivalence relation. The chains $C_1, \ldots, C_r$ represent the sequence where either next or prev is repeatedly applied to C or D to derive the shortest sequence and since BadP does not occur, if $C_i = \mathsf{next}(C_{i-1})$ then $C_{i-1} = \mathsf{prev}(C_i)$ and vice versa. Since each C_i for all $i \in \{2, \ldots, r-1\}$ is such that $\mathsf{next}(C_i) \neq \perp$ and $\mathsf{prev}(C_i) \neq \perp$, C_i is table-defined for all $i \in \{2, \ldots, r-1\}$. The last two bullet points follow from the definition of the procedures $\mathsf{val}^+(\cdot)$ and $\mathsf{val}^-(\cdot)$ and the existence of the sequence of chains $C_1, \ldots, C_r$. □

Claim 3.25. *Consider some point in a good execution of H_2 and assume that $x \notin G_j$ before every call to* ForceVal$(x, \cdot, j)$ *prior to this point in the execution. Then, if the partial chains $C = (x_k, x_{k+1}, k)$ with $k \in \{1, 5, 9\}$ and $D = (x'_m, x'_{m+1}, m)$ with $m \in \{1, 5, 9\}$ are equivalent at this point in the execution, then $C \in$* CompletedChains *if and only if $D \in$* CompletedChains.

Proof. Consider the case where C has just been placed in CompletedChains after being adapted. By Claim 3.24, the chains equivalent to C at this point are exactly those chains $(\mathsf{val}_i(C), \mathsf{val}_{i+1}(C), i)$ where $i \in \{0, \ldots, 10\}$. Hence, if C and D are equivalent and $k, m \in \{1, 5, 9\}$, then both $C, D \in$ CompletedChains (by definition of the simulator). Since BadP does not occur and ForceVal does not overwrite (by assumption in the lemma), $\mathsf{val}_i(C)$ does not change during a good execution of H_2, and hence this property continues to hold even after $C, D \in$ CompletedChains. □

Properties of Enqueued Chains

Recall that $\{1, 5, 6, 10\}$ are "bad" set uniform positions.

Claim 3.26. *Say a chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ is enqueued to be completed in Q_b. Then at the time C is enqueued, $\mathsf{val}_g(C) = \perp$ and $\mathsf{val}_b(C) \notin G_b$.*

Proof. Say $C = (x_5, x_6, 5, 2, 4, 1)$ where $g = 4$ and $b = 1$. Such a chain C is enqueued in Q_1 only if $x_5 \notin G_5$, by construction of the simulator. Otherwise, EnqNewChains$(5, x_5)$ is not called. Since $x_5 \notin G_5$ when enqueued, $\mathsf{val}_4^-(C) = \perp$ at the time C is enqueued.

Similarly, by construction of the simulator, a chain C is enqueued in Q_b only if $\mathsf{val}_b(C) \neq \perp$ and $\mathsf{val}_b(C) \notin G_b$. So, we have $\mathsf{val}_4(C) = \perp$ and $\mathsf{val}_1(C) \notin G_1$ at the time the chain C is enqueued. The other cases are analogous. □

Effects of a Call to ForceVal

For the following claims, note that $g, b \in \{\ell - 1, \ell + 2\}$ and $g \neq b$.

Claim 3.27. *In a good execution of H_2, let $x_{\ell-1} \notin G_{\ell-1}$ (respectively $x_{\ell+2} \notin G_{\ell+2}$) immediately before a call* Adapt$(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$. *Then, before the call to* ForceVal$(x_\ell, \cdot, \ell)$ *(respectively* ForceVal$(x_{\ell+1}, \cdot, \ell+1)$*) in that* Adapt *call, we have $x_\ell \notin G_\ell$ (respectively $x_{\ell+1} \notin G_{\ell+1}$).*

Proof. The proof follows exactly as in Lemma [17, Lemma 3.26(a)]. □

Corollary 3.28. *In a good execution of H_2, consider a call to* Adapt$(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$ *and assume that adapting was safe for all chains C that were dequeued from Q_1, Q_5, Q_6, Q_{10}, Q^*_{all}, or Q_{mid} before this* Adapt *call. Then, before the call to* ForceVal$(x_\ell, \cdot, \ell)$ *and* ForceVal$(x_{\ell+1}, \cdot, \ell+1)$ *that occurs in the call*

ADAPT$(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$, we have $x_\ell \notin G_\ell$ and $x_{\ell+1} \notin G_{\ell+1}$ respectively.

Proof. The proof follows immediately from Claim 3.27. □

Claim 3.29. *In a good execution of H_2, suppose that $x_{\ell-1} \notin G_{\ell-1}$ (resp., $x_{\ell+2} \notin G_{\ell+2}$) immediately before a call* ADAPT$(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$*. Then, if C is a table-defined chain before the call to* ADAPT, $\mathsf{val}_i(C)$ *for $i \in \{1, \ldots, 10\}$ stays constant during the call to* FORCEVAL$(x_\ell, \cdot, \ell)$ *(resp.,* FORCEVAL$(x_{\ell+1}, \cdot, \ell+1)$*).*

Proof. The proof follows exactly as in Lemma [17, Lemma 3.26(b)]. □

Claim 3.30. *Consider a good execution of H_2. Suppose that $x_{\ell-1} \notin G_{\ell-1}$ (respectively, $x_{\ell+2} \notin G_{\ell+2}$) immediately before a call* ADAPT$(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$*. Then, if C is a chain enqueued in Q_{all},* $\mathsf{val}_i(C)$ *for $i \in \{1, \ldots, 10\}$ stays constant during the call to* FORCEVAL$(x_\ell, \cdot, \ell)$ *(respectively,* FORCEVAL$(x_{\ell+1}, \cdot, \ell+1)$*) that occurs in the* ADAPT *call.*

Proof. Consider the case that C is equivalent to the chain being adapted. If C is equivalent to the chain being adapted, then $\mathsf{val}_i(C)$ stays constant during the calls to FORCEVAL by definition of the procedure.

Consider the case where the chain C is not equivalent to the chain being adapted. Then $\mathsf{val}_i(C)$ can change during the call to FORCEVAL$(x_\ell, \cdot, \ell)$ only if $\mathsf{val}_\ell^\rho(C) = x_\ell$ for some $\rho \in \{+, -\}$. This implies that $\mathsf{BadlyCollide}^+$ occurred on the uniform assignment to $G_{\ell-1}(x_{\ell-1})$ that happened in the ADAPT call. This is because C is a chain enqueued for completion in Q_{all}, and C is not equivalent to the chain

$D = (x_{\ell-2}, x_{\ell-1}, \ell - 2)$ by assumption. Now, before the assignment $\mathsf{val}^{+}_{\ell}(D) = \perp$, and after the assignment $\mathsf{val}^{\rho}_{\ell}(C) = \mathsf{val}^{+}_{\ell}(D) \neq \perp$ and D is table-defined. A similar argument works for the call to $\textsc{ForceVal}(x_{\ell+1}, \cdot, \ell+1)$ when $x_{\ell+2} \notin G_{\ell+2}$. □

Claim 3.31. *In a good execution of H_2, consider a call to $\textsc{Adapt}(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$ for some $Q \in \{Q_1, Q_5, Q_6, Q_{10}\}$. Assume that adapting was safe for all chains C that were dequeued from Q_1, Q_5, Q_6, Q_{10} before this* Adapt *call. If $x_a \notin G_a$ and $x_g \notin G_g$ (where a is the adapt position adjacent to the "bad" set uniform position) before the* Adapt *call, then if C is a chain enqueued in Q_{all}, $\mathsf{val}_i(C)$ for $i \in \{1, \ldots, 10\}$ stays constant during the call to $\textsc{ForceVal}(x_a, \cdot, a)$ that occurs in the* Adapt *call.*

Proof. Consider the case where C is equivalent to the chain being adapted. Then, $\mathsf{val}_i(C)$ stays constant during the call to $\textsc{ForceVal}(x_a, \cdot, a)$ by definition of the procedure.

Consider a chain C that is not equivalent to the chain $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ being adapted. Assume the lemma has held until the Adapt call of the chain D currently being adapted i.e. $\textsc{ForceVal}(x_{a'}, \cdot, a')$ did not affect $\mathsf{val}_i(C)$ for any chain C such that C is a chain enqueued in Q_{all} and a' is the adapt position adjacent to the "bad" set uniform b' of a chain D' enqueued in $Q_{b'}$ and dequeued before the current Adapt call.

Let D be the chain enqueued in Q_b during whose completion the Adapt call occurred. Now, by construction of the simulator, D is equivalent to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ and hence not equivalent to C. For $\mathsf{val}_i(C)$ to change during the call to $\textsc{ForceVal}(x_a, \cdot, a)$

that occurs during the Adapt call, it must be the case that $\mathsf{val}_a(C) = \mathsf{val}_a(D) = x_a$. For a chain D enqueued in Q_b, we have $\mathsf{val}_b(D) \notin G_b$ and $\mathsf{val}_g(D) =\perp$ when D was enqueued by Claim 3.26. So, $\mathsf{val}_a(D) =\perp$ when chain D was enqueued.

Consider the case where just before the Adapt call of D, we have $x_b \notin G_b$. Then, $\mathsf{val}_a(D) =\perp$ since $x_b \notin G_b$ and $x_g \notin G_g$ by assumption. So, if $\mathsf{val}_a(C) = \mathsf{val}_a(D) \neq\perp$ before the call to ForceVal$(x_a, \cdot, a)$, then $\mathsf{BadlyCollide}^+$ occurred on the uniform assignment to $G_b(x_b)$. Consider next the case where just before the Adapt call of D, we have $x_b \in G_b$. So, if $\mathsf{val}_a(C) = \mathsf{val}_a(D) \neq\perp$ before the ForceVal call, then this occurred during the completion of a chain E that was dequeued before D. Then, E was dequeued from $Q_{b'}$ for some $b' \in \{1, 5, 6, 10\}$ by construction of the simulator. Consider the last assignment that happened before $\mathsf{val}_a(C) = \mathsf{val}_a(D) \neq\perp$ was true. As stated above, this assignment must have happened during the completion of a chain E that was dequeued before D. This assignment can either be (a) a $p(\uparrow, x_{10}, x_{11})$ or $p(\downarrow, x_0, x_1)$ assignment, but then BadP occurred, (b) a ForceVal assignment, but by assumption that the lemma has held so far and by Claim 3.30, this cannot be true, or (c) a uniform assignment to $G_j(x_j)$, but then $\mathsf{BadlyCollide}^+$ occurred.

So, if C is a chain enqueued for completion in Q_{all}, $\mathsf{val}_i(C)$ for $i \in \{1, \ldots, 10\}$ stays constant during the call to ForceVal$(x_a, \cdot, a)$. □

Additional Properties of Enqueued Chains

For the following claim, if a chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ is enqueued in Q_{mid}, then the assignment $G_i(x_i)$ that precedes C being enqueued happens either in lines 125, 147, or 150 of the simulator's execution.

Claim 3.32. *In a good execution of H_2, if at the time a chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ is enqueued in Q_{mid}, no chain equivalent to C has been enqueued for completion and adapting was safe for every chain dequeued from Q_1, Q_5, Q_6, Q_{10} or Q^*_{all} so far, then $\mathsf{val}_g(C) = \perp$ and $\mathsf{val}_b(C) = \perp$ just before the assignment $G_i(x_i)$ that precedes C being enqueued. Also, $\mathsf{val}_9(C) = \mathsf{val}_2(C) = \perp$ just before the assignment $G_i(x_i)$ that precedes C being enqueued.*

Proof. Say a chain $C = (x_5, x_6, 5, 2, 4, 1)$ is enqueued in Q_{mid} with $g = 4$ and $b = 1$. Then, the assignment $G_5(x_5)$ that precedes the enqueueing of C is such that $x_5 \notin G_5$ before the assignment, by construction of the simulator. Otherwise, EnqNewMidChains$(5, x_5)$ is not called. Hence, $\mathsf{val}_4^-(C) = \perp$ just before the assignment $G_5(x_5)$ that precedes C being enqueued. Also, since $\mathsf{val}_4^-(C) = \perp$, we have $\mathsf{val}_1^-(C) = \perp$.

Before we prove $\mathsf{val}_4^+(C) = \perp$ and $\mathsf{val}_1^+(C) = \perp$ (and hence, $\mathsf{val}_4(C) = \perp$ and $\mathsf{val}_1(C) = \perp$), we make the following observation. If a partial chain $(x_5, x_6, 5)$ is enqueued in Q_{mid} such that no equivalent chain has been enqueued previously, by construction of the simulator, either (1) $\mathsf{val}_5(D) = x_5$ for a chain D belonging to Q^*_{all} where $\mathsf{val}_5(D) = \perp$ when D was enqueued or (2) $\mathsf{val}_6(E) = x_6$ for a chain E enqueued in Q^*_{all} where $\mathsf{val}_6(E) = \perp$ when E was enqueued or (3) both. In other words, either $x_5 \notin G_5 \cup A_5^t$ or $x_6 \notin G_6 \cup A_6^t$ or both when $Q_{\text{enq}}.\textsc{Empty}() = \textsf{true}$ in line 6 of the simulator's execution after $\mathcal{D}$'s t^{th} query.

Consider a chain $C = (x_5, x_6, 5, 2, 4, 1)$ that was enqueued in Q_{mid} such that no chain equivalent to C was enqueued previously. Such a chain C is enqueued in Q_{mid}, when $x_6 \in G_6$, $\textsf{val}_5(C) = \textsf{val}_5(D) = x_5$ and $x_5 \in G_5$ right before C was enqueued (and not earlier) where D is a chain belonging to Q_{all}^* and $x_5 \in G_5$ due to the completion of D.

For $\textsf{val}_1(C) \neq \perp$ at the time of the assignment that precedes the enqueueing of C, we need $\textsf{val}_1^+(C) \neq \perp$. Then, in particular, we have that $x_7 := \textsf{val}_7(C) \in G_7$ and $x_8 := \textsf{val}_8(C) \in G_8$ (otherwise, $\textsf{val}_9^+(C) = \perp$ implying that $\textsf{val}_1^+(C) = \perp$).

Consider the partial chains $C = (x_5, x_6, 5)$, $C_1 = (x_6, x_7, 6)$ and $C_2 = (x_7, x_8, 7)$. For $\textsf{val}_9^+(C) \neq \perp$ just before the assignment that precedes the enqueueing of C, we need (1) $C_1 = \textsf{next}(C)$, $C_2 = \textsf{next}(C_1)$ (and hence, $x_6 \in G_6$ and $x_7 \in G_7$), (2) $x_5 = \textsf{val}_5(D)$ for a chain D in Q_{all}^*, and (3) $x_8 \in G_8$ or $x_8 = \textsf{val}_8(E)$ for a chain E enqueued in Q_{all}. Note that this condition is not true at the time the simulator finished enqueueing chains in Q_{all} since we have either $x_5 \notin G_5 \cup A_5^t$ or $x_6 \notin G_6 \cup A_6^t$ or both. Hence, the conditions must have been met during the completion of chains in Q_{all}. Consider the last assignment that was made before all the above conditions were met.

Consider the case when the last assignment (such that all the conditions listed above were met immediately after this assignment) happened, the chain C_1 was already table-defined. Now, if the assignment was a $\mathrm{P}/\mathrm{P}^{-1}$ assignment, then $\textsf{BadP}$ occurred. It cannot be a $\textsc{ForceVal}$ assignment since $\textsc{ForceVal}$ does not change the value of a chain enqueued in Q_{all} by Claims 3.30 and 3.31. If it were a uniform assignment to $G_i(x_i)$, then $\mathsf{BadlyCollide}^+$ occurred.

Consider the case when the last assignment (such that all the conditions listed above were met immediately after this assignment) happened, the chain C_1 was not table-defined before the assignment but table-defined immediately after. Recall that if $C_1 = (x_6, x_7, 6)$ is table-defined then $x_6 \in G_6$ and $x_7 \in G_7$. So, the assignment was either to $G_6(x_6)$ or $G_7(x_7)$.

Consider the case that the last assignment (such that all the conditions listed above were met immediately after this assignment) set $G_7(x_7)$. If this were a uniform assignment to $G_7(x_7)$, then $\mathsf{BadlyCollide}^+$ occurred since $C_1(\equiv C)$ and E are not equivalent as no chain equivalent to C has been enqueued previously. If this were a ForceVal assignment, then $\mathsf{BadlyCollideFV}$ occurred. This is because 7 is an adapt position only for partial chains that are either of the form (a) $X = (x_9, x_{10}, 9)$ such that $(x_9, x_{10}, 9, 6, 8, 5)$ belongs to Q^*_{all}, or (b) $Y = (x_1, x_2, 1)$ such that $(x_1, x_2, 1, 7, 9, 6)$ is enqueued in Q_6. For case (a), by assumption for chains in Q^*_{all}, we have $\mathsf{val}_5(X) \notin G_5$ before the Adapt call for such a chain. For case (b), the adapt position 7 is adjacent to the "bad" set uniform position 6. By assumption for chains enqueued in Q_6, we have $\mathsf{val}_9(Y) \notin G_9$ before the Adapt call for such a chain. Hence, $\mathsf{BadlyCollideFV}$ occurred due to the assignment $G_5(\mathsf{val}_5(X))$ or $G_9(\mathsf{val}_9(Y))$ that occurs in the Adapt call. The analysis for the case when $G_6(x_6)$ is set is similar. So, the above conditions are not met for a chain C to be enqueued in Q_{mid}. Hence, for such a chain $C = (x_5, x_6, 5, 2, 4, 1)$, $\mathsf{val}^+_9(C) =\perp$ just before the assignment that caused C to be enqueued. Since $\mathsf{val}^+_9(C) =\perp$ and $\mathsf{val}^-_4(C) =\perp$

before the assignment, we have $\mathsf{val}_4(C) =\perp$, $\mathsf{val}_9(C) =\perp$ and $\mathsf{val}_1(C) =\perp$ just before the assignment that precedes C being enqueued. The analysis for the case where $C = (x_5, x_6, 5, 8, 7, 10)$ is analogous. □

Claim 3.33. *Consider a good execution of H_2. Just before the execution of line 27 during the simulator's execution, if adapting was safe for every chain dequeued from Q_1,Q_5,Q_6,Q_{10}, Q^*_{all} or Q_{mid} so far, then it holds that:*

(i) *if $x_9 \in G_9$, $x_{10} \in G_{10}$, $x_1 \in G_1$ such that $R.\textsc{CheckBwd}(x_{10}, x_9 \oplus G_{10}(x_{10}), x_1) =$ true, then $(x_9, x_{10}, 9) \in \mathsf{CompletedChains}$.*

(ii) *if $x_1 \in G_1$, $x_2 \in G_2$, $x_{10} \in G_{10}$ such that $R.\textsc{CheckFwd}(x_2 \oplus G_1(x_1), x_1, x_{10}) =$ true, then $(x_1, x_2, 1) \in \mathsf{CompletedChains}$.*

(iii) *if $x_5 \in G_5$, $x_6 \in G_6$, then $(x_5, x_6, 5) \in \mathsf{CompletedChains}$.*

Proof. We start by proving (i). For a triple (x_9, x_{10}, x_1), we say that the "condition holds" if (x_9, x_{10}, x_1) is such that $x_9 \in G_9$, $x_{10} \in G_{10}$, $x_1 \in G_1$ and $R.\textsc{CheckBwd}(x_{10}, x_9 \oplus G_{10}(x_{10}), x_1) = \mathsf{true}$. Also, we refer to the partial chain $(x_9, x_{10}, 9)$ as the partial chain associated with the triple (x_9, x_{10}, x_1). So, our aim is to prove that for every triple (x_9, x_{10}, x_1) such that the condition holds, the associated partial chain $(x_9, x_{10}, 9) \in \mathsf{CompletedChains}$. Assume the claim holds right before (and hence immediately after) line 27 of the simulator's execution while answering the distinguisher's $(t-1)^{th}$ query to $\mathrm{F}(\cdot, \cdot)$. Let the distinguisher ask its t^{th} query $\mathrm{F}(k, x)$. The aim is to prove that at line 27 of the simulator's execution while answering the distinguisher's t^{th} query to $\mathrm{F}(\cdot, \cdot)$, if a triple $T^* = (x_9, x_{10}, x_1)$ is such that the condition holds, then the partial chain $C^* = (x_9, x_{10}, 9)$ associated with the triple is such that $C^* \in \mathsf{CompletedChains}$. Note that the distinguisher could have made queries to $\mathrm{P}/\mathrm{P}^{-1}$ between the $(t-1)^{th}$ and t^{th} queries to $\mathrm{F}(\cdot,\cdot)$; but if those queries resulted in the condition holding, then BadP occurred.

Suppose that there exists a triple T^* such that the condition holds at line 27 of the simulator's execution while answering the distinguisher's t^{th} query. If the condition held at the end of simulator's execution while answering the previous distinguisher query, then by the assumption that the claim has held so far, the partial chain C^* associated with the triple T^* is such that $C^* \in \mathsf{CompletedChains}$. If the condition held at the end of the simulator's execution of the current query t (and not at the end of the previous query), we differentiate cases where the associated partial chain C^* was enqueued for completion during the simulator's execution while answering the t^{th} query and when it was not.

Consider the case where a chain equivalent to C^* was enqueued in Q_{all} during the simulator's execution while answering the distinguisher's current query. If $C^* = (x_9, x_{10}, 9)$ was enqueued during the t^{th} query, then $(x_9, x_{10}, 9) \in \mathsf{CompletedChains}$ by construction of the simulator. Note also that chains in $\mathsf{MidEquivChains}$ are not enqueued for completion by the simulator. By definition of the set $\mathsf{MidEquivChains}$, these chains are such that they are equivalent to a chain of the form $(x_5, x_6, 5)$ that has been enqueued for completion. Since BadP does not occur and ForceVal does not overwrite, the equivalence holds when $(x_5, x_6, 5) \in \mathsf{CompletedChains}$ and hence, by Claim 3.25, such a chain in $\mathsf{MidEquivChains}$ is placed in $\mathsf{CompletedChains}$ as well. By the same argument, if a chain equivalent to C^* has been enqueued for completion, then $C^* \in \mathsf{CompletedChains}$ by the end of the simulator's execution of the current query. So, if a chain equivalent to C^* was enqueued for completion or was in $\mathsf{MidEquivChains}$ during the simulator's execution while answering the current query t, then $C^* \in \mathsf{CompletedChains}$.

Consider the case where no chain equivalent to C^* was enqueued in Q_{all} and $C^* \notin \mathsf{MidEquivChains}$ during the simulator's execution while answering the distinguisher's current query. We differentiate between the cases where (1) $C = \mathsf{next}(C^*) \neq \perp$ and $\mathsf{next}(C) \neq \perp$ when $Q_{\text{enq}}.\textsc{Empty}() = \texttt{true}$ in line 6 of the simulator's execution when answering the distinguisher's t^{th} query and (2) when this is not the case.

Consider the case when $C = \mathsf{next}(C^*) \neq \perp$ and $\mathsf{next}(C) \neq \perp$ at the time the simulator stops enqueueing chains in Q_{all}, i.e., when $Q_{\text{enq}}.\textsc{Empty}() = \texttt{true}$ in line 6 of the simulator's execution when answering the distinguisher's t^{th} query. This implies that $x_{10} \in G_{10}$ and $(\uparrow, x_{10}, x_{11}) \in P$, where $x_{11} := x_9 \oplus G_{10}(x_{10})$, and hence $C = (x_{10}, x_{11}, 10)$ is table-defined at the time the simulator stops enqueueing chains in Q_{all}. Since the triple T^* is such that the associated partial chain $C^* = (x_9, x_{10}, 9)$ was not enqueued for completion and not in $\mathsf{MidEquivChains}$, we have that either (a) $x_9 \notin G_9 \cup A_9^t$ or (b) $x_1 \notin G_1 \cup A_1^t$ when $Q_{\text{enq}}.\textsc{Empty}() = \texttt{true}$ in line 6. For the condition to hold, we need $x_1 \in G_1$ and $x_9 \in G_9$, and hence we have that the condition does not hold for the triple T^* when $Q_{\text{enq}}.\textsc{Empty}() = \texttt{true}$ in line 6. Consider the case where $x_1 \notin G_1 \cup A_1^t$. For $x_1 \in G_1$ to be true by the end of the simulator's execution while answering the distinguisher's t^{th} query, it must be the case that $\mathsf{val}_1(D) = \mathsf{val}_1(C) = x_1$ at some point for a chain D that has been enqueued in Q_{all} or Q_{mid}. Before analyzing the case that $\mathsf{val}_1(D) = \mathsf{val}_1(C) = x_1$ occurs, we make the following observations. First, C and D are not equivalent as $C \equiv C^*$ and no chain equivalent to C^* (including itself) has been enqueued. Second, for all chains D that have been enqueued in Q_{all}, $\mathsf{val}_1(D) \neq x_1$ when D was enqueued since $x_1 \notin G_1 \cup A_1^t$. Now, if $\mathsf{val}_1(D) \neq x_1$ and $\mathsf{val}_1(D) \neq \perp$, it cannot be that $\mathsf{val}_1(D) = x_1$ at a later point since ForceVal does not overwrite and BadP does not occur. Hence, if $\mathsf{val}_1(D) = x_1$ at a later point, then $\mathsf{val}_1(D) = \perp$ when enqueued. Similarly, for all chains D that have been enqueued in Q_{mid} it holds that $\mathsf{val}_1(D) = \perp$ just before the assignment that precedes the enqueueing of D by Claim 3.32. Since $\mathsf{BadlyHit}^+$ and $\mathsf{BadlyHitFV}$ do not occur, $\mathsf{val}_1(D) = \perp$ at the time D is enqueued. Now, if $\mathsf{val}_1(D) = \mathsf{val}_1(C) = x_1$, then this is during the completion of some chain E during the simulator's execution while answering the distinguisher's t^{th} query. Consider the last assignment before $\mathsf{val}_1(D) = \mathsf{val}_1(C) = x_1$ was true. This cannot be a uniform assignment to $G_i(x_i)$ since then $\mathsf{BadlyCollide}^+$ occurred. This cannot be due to a uniform assignment to P since then BadP or $\mathsf{BadlyCollideP}$ occurred. This cannot be a ForceVal assignment since that would contradict Claims 3.29, 3.30 or 3.31. The analysis for the case where $x_9 \notin G_9 \cup A_9^t$ when the simulator stops enqueueing chains in Q_{all} is analogous. So, if C was table-defined when the simulator stops enqueueing chains in Q_{all}, then the condition does not hold for the triple T^* at the end of the simulator's execution of the current query.

Consider next the case when either $\mathsf{next}(C^*) = \perp$, or $C = \mathsf{next}(C^*) \neq \perp$ and $\mathsf{next}(C) = \perp$ at the time the simulator stops enqueueing chains in Q_{all}, i.e., when $Q_{\text{enq}}.\text{Empty}() = \mathsf{true}$ in line 6 of the simulator's execution when answering the distinguisher's t^{th} query. Now if the triple $T^* = (x_9, x_{10}, x_1)$ is such that the condition holds by the end of the simulator's execution of the current query, then it must be the case that $\mathsf{next}(C^*) \neq \perp$ and $\mathsf{next}(\mathsf{next}(C^*)) \neq \perp$ by the end of the simulator's execution. In particular, this means that the partial chain $\mathsf{next}(C^*) = C = (x_{10}, x_{11}, 10)$ where $x_{11} := x_9 \oplus G_{10}(x_{10})$ is table-defined (with $\mathsf{val}_1(C) = x_1$) by the end of the simulator's execution. Note that at the moment C becomes table-defined either $x_1 \notin G_1$ or $x_9 \notin G_9$ as otherwise either BadP or $\mathsf{BadlyHit}^+$ occurred. Furthermore, immediately before the assignment that causes C to be table-defined, we have either $\mathsf{val}_1(C) = \perp$ or $\mathsf{val}_9(C) = \perp$ and immediately after the assignment, we have $\mathsf{val}_9(C) \neq \perp$ and $\mathsf{val}_1(C) \neq \perp$ by definition. Say $\mathsf{val}_1(C) = \perp$ immediately before the assignment that caused C to be table-defined and $\mathsf{val}_1(C)(= x_1) \neq \perp$ immediately after. For $x_1 \in G_1$ to be true by the end of the simulator's execution while answering the distinguisher's t^{th} query, it must be the case that $\mathsf{val}_1(D) = \mathsf{val}_1(C) = x_1$ at some point for a chain D that has been enqueued in Q_{all} or Q_{mid}. Consider the last assignment before $\mathsf{val}_1(D) = \mathsf{val}_1(C) = x_1$ was true. The rest of the analysis proceeds similarly to the analysis above. The case when $\mathsf{val}_9(C) = \perp$ immediately before the assignment that caused C to be table-defined and $\mathsf{val}_9(C)(= x_9) \neq \perp$ immediately after follows in a similar fashion. So, if $\mathsf{next}(C^*) = \perp$ or if $\mathsf{next}(C^*) \neq \perp$ and $\mathsf{next}(\mathsf{next}(C^*)) = \perp$ when the simulator stops enqueueing chains in Q_{all}, then the condition also does not hold for the triple T^* at the end of the simulator's execution of the current query. Summarizing, if a chain equivalent to C^* was not enqueued in Q_{all} and $C^* \notin \mathsf{MidEquivChains}$ during the simulator's execution while answering the distinguisher's current query, then the condition does not hold for the triple T^* at the end of the simulator's execution of the current query.

The proof of (ii) follows exactly along the lines of the proof of (i) given above.

The proof of (iii) is as follows. Let $\mathcal{D}$ ask its t^{th} query $\mathrm{F}(k, x)$. Just before the simulator returns $G_k(x)$ in line 27, let the lemma be false and let this be the first time that the lemma does not hold implying that there exist $x_5 \in G_5$ and $x_6 \in G_6$ such that $(x_5, x_6, 5) \notin \mathsf{CompletedChains}$.

If the lemma has held so far, in particular it has held right before (and immediately after) line 27 of the simulator's execution while answering $\mathcal{D}$'s $(t-1)^{th}$ query to $\mathrm{F}(\cdot, \cdot)$. Note that the distinguisher could have made queries to $\mathrm{P}/\mathrm{P}^{-1}$ between the $(t-1)^{th}$ and t^{th} queries to $\mathrm{F}(\cdot, \cdot)$, but those queries cannot result in $x_5 \in G_5$ or $x_6 \in G_6$.

So, $x_5 \in G_5$, $x_6 \in G_6$ are such that $(x_5, x_6, 5) \notin \mathsf{CompletedChains}$ happened during the simulator's execution while answering $\mathcal{D}$'s t^{th} query. Now, if $(x_5, x_6, 5)$ were enqueued for completion during the t^{th} query then $(x_5, x_6, 5) \in \mathsf{CompletedChains}$. If a chain equivalent to $(x_5, x_6, 5)$ were enqueued for completion during the t^{th} query, then $(x_5, x_6, 5) \in \mathsf{CompletedChains}$. This is because equivalent chains are placed in $\mathsf{CompletedChains}$ simultaneously, since BadP does not occur and FORCEVAL does not overwrite. So, for $x_5 \in G_5$, $x_6 \in G_6$ to be such that $(x_5, x_6, 5) \notin \mathsf{CompletedChains}$, the simulator did not enqueue this partial chain. (Note that chains of the type $(x_5, x_6, 5)$ are not added to $\mathsf{MidEquivChains}$.)

Let $x_6 \in G_6$, and say an assignment occurs such that before the assignment $x_5 \notin G_5$, but after the assignment $x_5 \in G_5$ leading to the creation of a partial chain of the form $(x_5, x_6, 5)$ with $x_5 \in G_5, x_6 \in G_6$. (The analysis for the other case is analogous.) Such an assignment can happen only by completion of a chain in Q_1, Q_5, Q_6, Q_{10} or completion of a chain in Q^*_{all}. We analyze these next.

Case 1: An assignment happens to $G_5(x_5)$ during the completion of a chain C enqueued in Q_b where $b \in \{1, 5, 6, 10\}$ and $x_6 \in G_6$ before this assignment. Now, if $x_6 \in G_6$ before the assignment causing $x_5 \in G_5$, then either $x_6 \in G_6$ before $\mathcal{D}$'s t-th query or $x_6 \in G_6$ due to the completion of a chain D enqueued in Q_1, Q_5, Q_6, Q_{10} and dequeued before C. Again, by construction of the simulator, chains C that are enqueued in Q_b are such that either $\mathsf{val}_5(C) \in A^t_5$ or $\mathsf{val}_5(C) \in G_5$ at the time C was enqueued and similarly, chains D that are enqueued in Q_b are such that either $\mathsf{val}_6(D) \in A^t_6$ or $\mathsf{val}_6(D) \in G_6$ at the time D was enqueued. Since BadP does not occur and ForceVal does not overwrite, $\mathsf{val}_5(C) = x_5 \in A^t_5$ (since $x_5 \notin G_5$ before this assignment) and $\mathsf{val}_6(D) = x_6 \in G_6 \cup A^t_6$. Thus, $(x_5, x_6, 5)$ is enqueued for completion by construction of the simulator.

Case 2: An assignment happens to $G_5(x_5)$ during the completion of a chain C in Q^*_{all} and $x_6 \in G_6$ before this assignment. If $x_6 \in G_6 \cup A^t_6$ and $x_5 \in A^t_5$ when the simulator enqueues chains in Q_{all}, then $(x_5, x_6, 5)$ is enqueued for completion in Q_{all}. Else, $(x_5, x_6, 5)$ is enqueued for completion in Q_{mid}.

This completes the proof. □

Claim 3.34. *Consider a good execution of H_2. If a chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ belongs to Q^*_{all} such that at the time C is enqueued, adapting was safe for every chain dequeued from Q_1,Q_5,Q_6,Q_{10}, Q^*_{all} or Q_{mid} so far, then $\mathsf{val}_b(C) =\perp$ and $\mathsf{val}_g(C) =\perp$ at the time C is enqueued.*

Proof. Say $C = (x_9, x_{10}, 9, 3, 2, 5)$ is enqueued where the query preceding the chain's enqueueing is $G_1(x_1)$ where $\mathsf{val}_1(C) = x_1$. Then, by definition of the simulator, $x_1 \notin G_1$ as otherwise, EnqNewChains$(1, x_1)$ is not called. So, $\mathsf{val}_2^+(C) = \perp$. Now, we claim that $\mathsf{val}_5^-(C) \notin G_5$. This is because if $\mathsf{val}_5^-(C) \in G_5$, then $\mathsf{val}_6^-(C) \in G_6$ since otherwise, $\mathsf{val}_5^-(C) = \perp$. This implies that the partial chain $(x_5, x_6, 5)$, where $x_5 = \mathsf{val}_5^-(C)$ and $x_6 = \mathsf{val}_6^-(C)$, is such that $x_5 \in G_5$ and $x_6 \in G_6$. Hence, by Lemma 3.33, we have that $(x_5, x_6, 5) \in \mathsf{CompletedChains}$ since no new G_i assignments have been issued between the moment the simulator returned the answer (in line 27 of its execution) and the moment when chain C is enqueued in Q_{all}. However, since BadP does not occur, this means that $x_1 \in G_1$ contradicting the first statement. Thus, we have that $\mathsf{val}_5^-(C) \notin G_5$. Now, $\mathsf{val}_5^+(C) = \perp$ since $\mathsf{val}_2^+(C) = \perp$. So, $\mathsf{val}_5(C) \notin G_5$.

Since C is not enqueued in Q_1,Q_5,Q_6,Q_{10}, we have $\mathsf{val}_5(C) = \perp$ when C is enqueued. So $\mathsf{val}_2(C) = \perp$ and $\mathsf{val}_5(C) = \perp$, where $g = 2$ and $b = 5$. The other cases are analogous. □

ForceVal$(x, \cdot, j)$ does not Overwrite $G_j(x)$

Lemma 3.35. *Consider a good execution of H_2. Let $C = (x_k, x_{k+1}, k, \ell, g, b)$ be a partial chain enqueued in Q_1,Q_5,Q_6, or Q_{10}. At the moment $C = (x_k, x_{k+1}, k, \ell, g, b)$ is dequeued, assume that adapting was safe for every chain C' in Q^*_{all} or Q_{mid} dequeued so far. Then,*

- *At the moment $C = (x_k, x_{k+1}, k, \ell, g, b)$ is dequeued, $C \in \mathsf{CompletedChains}$, or*

- *Just before the call to* ADAPT *for C, $\mathsf{val}_g(C) \notin G_g$ and $\mathsf{val}_a(C) \notin G_a$ (where a is the adapt position adjacent to the "bad" set uniform position b).*

Proof of Lemma 3.35. Assume the lemma has held until the moment that some chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ is dequeued. Note that if the lemma has held until now we have that for every call to FORCEVAL$(x, \cdot, j)$ so far, $x \notin G_j$ by Corollary 3.28.

Consider the case that at the moment C was dequeued there is a chain D equivalent to C that was dequeued before C. Now, if D was dequeued before C, then $D \in \mathsf{CompletedChains}$ by construction of the simulator. If C and D are equivalent chains such that $D \in \mathsf{CompletedChains}$, then $C \in \mathsf{CompletedChains}$ by Claim 3.25.

Next consider the case where no chain equivalent to C was dequeued before C was dequeued. Say $C \notin \mathsf{CompletedChains}$ when dequeued. If we prove $\mathsf{val}_g(C) \notin G_g$ and $\mathsf{val}_a(C) \notin G_a$ at the time C was dequeued, we have that $\mathsf{val}_g(C) \notin G_g$ and $\mathsf{val}_a(C) \notin G_a$ just before the call to ADAPT for C since otherwise BadP or $\mathsf{BadlyHit}^+$ occurred.

By Claim 3.26, we have that $\mathsf{val}_g(C) = \perp$ at the time C was enqueued. If $\mathsf{val}_g(C) \in G_g$ at the time C was dequeued, then this was due to the completion of a chain D which was enqueued in $Q_{b'}$ where $b' \in \{1, 5, 6, 10\}$ due to the same distinguisher query as C and dequeued(and completed) before C such that $\mathsf{val}_g(C) = \mathsf{val}_g(D) \neq \perp$.

Consider the last assignment that was made before $\mathsf{val}_g(C) = \mathsf{val}_g(D) \neq \perp$ was true. This cannot have been a uniform assignment to $G_i(x_i)$ since that implies that

$\mathsf{BadlyCollide}^+$ occurred. This is because C and D are not equivalent(by assumption) and C and D are both enqueued for completion in Q_{all} and either $\mathsf{val}_g(C) =\perp$ or $\mathsf{val}_g(D) =\perp$ before the assignment(otherwise this is not the last assignment before $\mathsf{val}_g(C) = \mathsf{val}_g(D) \neq\perp$) and $\mathsf{val}_g(C) = \mathsf{val}_g(D) \neq\perp$ after the assignment.

The assignment cannot have been of the form $p(\downarrow, x_0, x_1) = (x_{10}, x_{11})$ or $p(\uparrow, x_{10}, x_{11}) = (x_0, x_1)$ since then BadP occurred. The assignment cannot have been a ForceVal query. This is because from Claims 3.31 and 3.30 we have that ForceVal does not change $\mathsf{val}_i(C)$ for a chain C enqueued in Q_{all} (including those enqueued in Q_1, Q_5, Q_6, Q_{10}) during completion of chains in Q_1, Q_5, Q_6, Q_{10}.

Now, consider the argument for $\mathsf{val}_a(C) \notin G_a$ when C is dequeued. By Claim 3.26, we have that $\mathsf{val}_b(C) \notin G_b$ and $\mathsf{val}_g(C) =\perp$ at the time C was enqueued, implying that $\mathsf{val}_a(C) =\perp$ when C was enqueued (where a is the adapt position adjacent to "bad" set uniform position). The argument for this case follows similar to the one above for $\mathsf{val}_g(C)$. □

Lemma 3.36. *Consider a good execution of H_2. Let $C = (x_k, x_{k+1}, k, \ell, g, b)$ be a partial chain in Q^*_{all}. At the moment $C = (x_k, x_{k+1}, k, \ell, g, b)$ is dequeued, assume that adapting was safe for every chain C' in Q_{mid} dequeued so far. Then,*

- *At the moment C is dequeued, $C \in \mathsf{CompletedChains}$ or,*
- *Just before the call to* Adapt *for C, $\mathsf{val}_{\ell-1}(C) \notin G_{\ell-1}$ and $\mathsf{val}_{\ell+2}(C) \notin G_{\ell+2}$.*

Proof. Recall that $g, b \in \{\ell - 1, \ell + 2\}$ and $g \neq b$. Assume the lemma has held until the moment that some chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ is dequeued. Note that if the lemma has held until now we have that for every call to $\textsc{ForceVal}(x, \cdot, j)$ so far, $x \notin G_j$ by Lemma 3.35 and Corollary 3.28.

Consider the case that at the moment C was dequeued, a chain D equivalent to C was dequeued before C. Now, if D was dequeued before C, then $D \in \mathsf{CompletedChains}$ by construction of the simulator. If C and D are equivalent chains such that $D \in \mathsf{CompletedChains}$, then $C \in \mathsf{CompletedChains}$ by Claim 3.25.

Consider the case that no chain equivalent to C was dequeued before C was dequeued. Note also that if we prove $\mathsf{val}_{\ell-1}(C) \notin G_{\ell-1}$ and $\mathsf{val}_{\ell+2}(C) \notin G_{\ell+2}$ at the time C was dequeued, we have that $\mathsf{val}_{\ell-1}(C) \notin G_{\ell-1}$ and $\mathsf{val}_{\ell+2}(C) \notin G_{\ell+2}$ just before the call to $\textsc{Adapt}$ for C since otherwise BadP or $\mathsf{BadlyHit}^+$ occurs.

From Claim 3.34, we have that $\mathsf{val}_{\ell-1}(C) =\perp$ and $\mathsf{val}_{\ell+2}(C) =\perp$ at the time the chain C is enqueued. Let us consider the case where $\mathsf{val}_{\ell-1}(C) \in G_{\ell-1}$ at the time C was dequeued. If $\mathsf{val}_{\ell-1}(C) \in G_{\ell-1}$ at the time C was dequeued, then this was due to the completion of a chain D which was enqueued in Q_{all} (including Q_1,Q_5,Q_6,Q_{10}) as a result of the same distinguisher query as C and was dequeued(and completed) before C such that $\mathsf{val}_{\ell-1}(C) = \mathsf{val}_{\ell-1}(D) \neq\perp$ at the time C was dequeued.

Consider the last assignment that was made before $\mathsf{val}_{\ell-1}(C) = \mathsf{val}_{\ell-1}(D) \neq\perp$ was true. This cannot have been a uniform assignment to $G_i(x_i)$ since that implies that $\mathsf{BadlyCollide}^+$ occurred. This is because C and D are not equivalent(by assumption) and either $\mathsf{val}_{\ell-1}(C) =\perp$ or $\mathsf{val}_{\ell-1}(D) =\perp$ before the assignment(otherwise this is not the last assignment before $\mathsf{val}_{\ell-1}(C) = \mathsf{val}_{\ell-1}(D) \neq\perp$) and $\mathsf{val}_{\ell-1}(C) = \mathsf{val}_{\ell-1}(D) \neq\perp$ after the assignment.

The assignment cannot have been of the form $p(\downarrow, x_0, x_1) = (x_{10}, x_{11})$ or

$p(\uparrow, x_{10}, x_{11}) = (x_0, x_1)$ since then BadP or $\mathsf{BadlyCollideP}$ occurred. The assignment cannot have been a ForceVal assignment since ForceVal does not change $\mathsf{val}_i(C)$ for a chain C enqueued in Q_{all} during completion of chains in Q_1,Q_5,Q_6,Q_{10} by Claims 3.31 and 3.30 and 3.35. Similarly, ForceVal that occurs during the completion of chains in Q_{all} does not change $\mathsf{val}_i(C)$ for a chain C enqueued in Q_{all} by Claim 3.30 as the lemma has held thus far.

The argument for $\mathsf{val}_{\ell+2}(C) \notin G_{\ell+2}$ when C was dequeued is analogous. □

Lemma 3.37. *Consider a good execution of H_2. Let $C = (x_k, x_{k+1}, k, \ell, g, b)$ be a partial chain enqueued in Q_{mid}. Then,*

- *At the moment C is dequeued, $C \in \mathsf{CompletedChains}$, or*
- *Just before the call to* Adapt *for C, $\mathsf{val}_{\ell-1}(C) \notin G_{\ell-1}$ and $\mathsf{val}_{\ell+2}(C) \notin G_{\ell+2}$.*

Proof. Again, we assume the lemma has held until the moment that some chain $C = (x_k, x_{k+1}, k, \ell, g, b)$ is dequeued. Note that if the lemma has held until now we have that for every call to $\text{ForceVal}(x, \cdot, j)$ so far, $x \notin G_j$ by Lemmas 3.35, 3.36 and Corollary 3.28.

Case 1: Consider the case that when a chain C is enqueued in Q_{mid}, there is an equivalent chain that has been enqueued previously. We analyze two subcases. First, consider the case that when a chain C is enqueued in Q_{mid}, there is an equivalent chain D enqueued previously and $D \in \mathsf{CompletedChains}$ when C is enqueued. If C equivalent to D and $D \in \mathsf{CompletedChains}$, then by Claim 3.25, we have that $C \in \mathsf{CompletedChains}$.

Consider the case that when a chain C is enqueued in Q_{mid}, no equivalent chain belongs to $\mathsf{CompletedChains}$ but there is a chain D equivalent to C that has been enqueued in Q_{all} or Q_{mid}. Now, if C and D are equivalent when C is enqueued, then there exists a sequence of chains $C_1, \ldots, C_r$ given by Claim 3.24. Since BadP does not occur and ForceVal does not overwrite until the moment C is dequeued, we have that C and D are equivalent when D is placed in $\mathsf{CompletedChains}$. So, by Claim 3.25, $C \in \mathsf{CompletedChains}$ and hence, $C \in \mathsf{CompletedChains}$ at the moment it is dequeued.

Case 2: Consider the case that no chain equivalent to C was enqueued before C was enqueued. Note also that if we prove $\mathsf{val}_{\ell-1}(C) \notin G_{\ell-1}$ and $\mathsf{val}_{\ell+2}(C) \notin G_{\ell+2}$ at the time C was dequeued, we have that $\mathsf{val}_{\ell-1}(C) \notin G_{\ell-1}$ and $\mathsf{val}_{\ell+2}(C) \notin G_{\ell+2}$ just before the call to Adapt for C since otherwise BadP or $\mathsf{BadlyHit}^+$ occurs.

Since no chain equivalent to C was enqueued previously, from Claim 3.32, we have that $\mathsf{val}_{\ell-1}(C) = \perp$ and $\mathsf{val}_{\ell+2}(C) = \perp$ immediately before the assignment that caused the chain C is enqueued. Let us consider the case where $\mathsf{val}_{\ell-1}(C) \in G_{\ell-1}$ at the time C was dequeued. This could not have happened at the time C was enqueued since then $\mathsf{BadlyHit}^+$ or $\mathsf{BadlyHitFV}$ occurred. If $\mathsf{val}_{\ell-1}(C) \in G_{\ell-1}$ at the time C was dequeued, then this was due to the completion of a chain D which was enqueued before C was enqueued and dequeued(and completed) after C was enqueued. By construction of the simulator, this means that D has to be enqueued in Q_{all} (but not in Q_1,Q_5,Q_6,Q_{10}) or Q_{mid}. Note also that C and D have to be such that $\mathsf{val}_{\ell-1}(C) = \mathsf{val}_{\ell-1}(D) \neq \perp$ at the time C was dequeued.

Consider the last assignment that was made before $\mathsf{val}_{\ell-1}(C) = \mathsf{val}_{\ell-1}(D) \neq \perp$

was true. This cannot have been a uniform assignment to $G_i(x_i)$ since that implies that $\mathsf{BadlyCollide}^+$ occurred. This is because C and D are not equivalent(by assumption) and either $\mathsf{val}_{\ell-1}(C) =\perp$ or $\mathsf{val}_{\ell-1}(D) =\perp$ before the assignment(otherwise this is not the last assignment before $\mathsf{val}_{\ell-1}(C) = \mathsf{val}_{\ell-1}(D) \neq\perp$) and $\mathsf{val}_{\ell-1}(C) = \mathsf{val}_{\ell-1}(D) \neq\perp$ after the assignment.

The assignment cannot have been of the form $p(\downarrow, x_0, x_1) = (x_{10}, x_{11})$ or $p(\uparrow, x_{10}, x_{11}) = (x_0, x_1)$ since then BadP or $\mathsf{BadlyCollideP}$ occurred. The assignment cannot have been a ForceVal query since ForceVal does not change $\mathsf{val}_i(C)$ for a table-defined chain C by Claim 3.29.

The argument for $\mathsf{val}_{\ell+2}(C) \notin G_{\ell+2}$ when C was dequeued is analogous. □

Theorem 3.38 (No overwrites). *In a good execution of H_2, we have $x \notin G_j$ before any call to* ForceVal$(x, \cdot, j)$.

Proof. Combining the result of Lemmas 3.35, 3.36 and 3.37 with Corollary 3.28, we have that for every call to ForceVal$(x, \cdot, j)$, $x \notin G_j$ before the call. □

As in Coron et al. [17], the distinguisher *completes all chains*, if, at the end of the execution, it emulates a call to EvalFwdComp$(x_0, x_1, 0, 10)$ for all queries to $\mathrm{P}(x_0, x_1)$ or to $(x_0, x_1) = \mathrm{P}^{-1}(x_{10}, x_{11})$ that it made during the execution.

Lemma 3.39. *Consider a good execution of H_2 in which the distinguisher completes all chains. Suppose that during the execution $\mathrm{P}(x_0, x_1)$, respectively $\mathrm{P}^{-1}(x_{10}, x_{11})$, is queried by the simulator or the distinguisher. Then, $p(\downarrow, x_0, x_1) = (\mathsf{val}^+_{10}(x_0, x_1, 0), \mathsf{val}^+_{11}(x_0, x_1, 0))$, respectively $p(\uparrow, x_{10}, x_{11}) = (\mathsf{val}^-_0(x_{10}, x_{11}, 10), \mathsf{val}^-_1(x_{10}, x_{11}, 10))$ at the end of the execution.*

Proof. The simulator queries $\mathrm{P}/\mathrm{P}^{-1}$ either (1) when it is enqueueing chains in Q_{all} or (2) when it is completing chains in Q_1, Q_5, Q_6, Q_{10}, Q^*_{all} and Q_{mid}. In the second case, when the simulator has completed the chain, we have the result and it holds even at the end of the execution as ForceVal does not overwrite by Theorem 3.38 and BadP does not occur. In the first case, where the simulator queries $\mathrm{P}/\mathrm{P}^{-1}$ when it is enqueueing chains in Q_{all}, the simulator either enqueues the chain in Q_{all} or places it in MidEquivChains. If the simulator enqueues the chain in Q_{all}, at the time the simulator completes the chain, the lemma holds and we have the result at the end of the execution as well since BadP does not occur and ForceVal does not overwrite. If the simulator places the chain in MidEquivChains, then an equivalent chain C was enqueued for completion in Q_{all}. Again, since BadP does not occur and ForceVal does not overwrite by Theorem 3.38, we have that the equivalence holds until $C \in$ CompletedChains and so the lemma holds at the end of the execution as well.

Consider the case where the query $\mathrm{P}(x_0, x_1)$ was made by the distinguisher. Since the distinguisher completes all chains, it eventually queries x_5 and x_6 corresponding to the Feistel evaluation of $(x_0, x_1, 0)$ at some point. This implies that $x_5 \in G_5$ and $x_6 \in G_6$ at the end of the execution. One of these two values are queried later by the distinguisher, say x_6, and at the moment if $(x_5, x_6, 5) \notin$ CompletedChains, it is enqueued and completed by the simulator and again, by the argument above, the lemma holds. If $(x_5, x_6, 5) \in$ CompletedChains, then the lemma holds right after that completion as BadP does not occur and ForceVal does not overwrite. □

In the following lemma, we make the randomness p used by R explicit. The randomness p is a list containing $2 \cdot 2^{2n}$ strings of length $2n$. Then R runs deterministically given p. So, whenever the procedure $R.\mathrm{P}(x_0, x_1)$ is queried, R checks if $(\downarrow, x_0, x_1) \in p$ and if so, answers accordingly. Otherwise, R reads $(x_{10}, x_{11}) := p(\downarrow, x_0, x_1)$, and then $(\downarrow, x_0, x_1)$ as well as $(\uparrow, x_{10}, x_{11})$ are added to p, mapping to each other. The procedure $R.\mathrm{P}^{-1}(x_{10}, x_{11})$ is implemented analogously.

Lemma 3.40. *Consider a good execution of H_2 in which the distinguisher completes all chains. Then, the number of calls to* Adapt *by the simulator equals the number of queries to $p(\cdot, \cdot, \cdot)$ made by R.*

Proof. The number of queries to $p(\cdot, \cdot, \cdot)$ equals half the number of entries in the table P since BadP does not occur. And for each call to Adapt, there is a corresponding entry in p that was read while evaluating forward/backward; also two calls to Adapt procedure can't share the same entry as otherwise BadP occurred or ForceVal has overwritten contradicting Theorem 3.38.

For a query in $p(\cdot, \cdot, \cdot)$, consider the case that the query was made in a call to P by the simulator. Then, this call was either made while the simulator was completing a chain, in which case we consider the Adapt call that was made right after this query, or, this call was made while the simulator was enqueueing a chain C in Q_{all}. For this case, consider the chains equivalent to C from this moment until a chain D equivalent to C is dequeued for completion where D is the first chain equivalent to C to be dequeued. Since no value is overwritten and BadP does not occur, we can associate the Adapt call that is called during the completion of chain

D to the p query.

Consider the case that the distinguisher made a query to $p(\cdot,\cdot,\cdot)$. Since the distinguisher completes all chains, it eventually makes the corresponding queries to the Feistel and say x_5 and x_6 are the corresponding Feistel queries. One of them has to be queried last by the distinguisher and at this moment consider the chain $(x_5, x_6, 5)$. If $(x_5, x_6, 5) \in \mathsf{CompletedChains}$, then consider the first chain equivalent to $(x_5, x_6, 5)$ that was dequeued and we associate the p query to the Adapt call that occurred during the completion of this chain. If $(x_5, x_6, 5) \notin \mathsf{CompletedChains}$, then at the moment $(x_5, x_6, 5)$ is dequeued, consider the first chain equivalent to $(x_5, x_6, 5)$ that was dequeued and we associate the p query to the Adapt call that occurred during the completion of this chain. $\square$

3.4.4 Indistinguishability of the Second and Third Experiments

We now describe experiment H_3. Here, we explicitly consider the randomness z where z is a table containing an independent uniform bitstring of length n for each $i \in \{1,\ldots,10\}$ and $x_i \in \{0,1\}^n$. Whenever the Feistel construction needs to query the i-th round function on x, it uses the value $z(i,x)$ instead. In $H_3(z)$, the two-sided random function is replaced by the 10-round Feistel construction $\mathsf{Feistel}(z)$, and the distinguisher $\mathcal{D}$ interacts with $(\mathsf{Feistel}(z), \hat{\mathcal{S}}(z)^{\mathsf{Feistel}^+(z)})$. The construction $\mathsf{Feistel}^+(z)$ defined below contains additional procedures CheckFwd and CheckBwd that the simulator has access to. Note that the randomness z used by $\mathsf{Feistel}$ and $\hat{\mathcal{S}}$ is the same and the simulator answers queries to the round functions by running the procedure $\hat{\mathcal{S}}.\mathrm{F}(i, x)$ and whenever it needs to set $G_j(x_j)$ to a random value, it uses the value $z(j, x_j)$ instead. The Feistel construction $\mathsf{Feistel}(z)$ is defined as follows:

```
procedure $\mathrm{P}(x_0, x_1)$:
    for $i := 2$ to $11$ do
        $x_i := x_{i-2} \oplus z(i-1, x_{i-1})$
    $p(\downarrow, x_0, x_1) := (x_{10}, x_{11})$
    $p(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$
    return $(x_{10}, x_{11})$

procedure $\mathrm{P}^{-1}(x_{10}, x_{11})$:
    for $i := 9$ to $0$ do
        $x_i := x_{i+2} \oplus z(i+1, x_{i+1})$
    $p(\downarrow, x_0, x_1) := (x_{10}, x_{11})$
    $p(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$
    return $(x_0, x_1)$
```

The construction $\mathsf{Feistel}^{+}(z)$ that the simulator has access to contains the following CheckFwd and CheckBwd procedures in addition to the procedures $\mathsf{Feistel}.\mathrm{P}$ and $\mathsf{Feistel}.\mathrm{P}^{-1}$.

```
procedure CheckFwd$(x_0, x_1, x_{10})$:
    if $(\downarrow, x_0, x_1) \in p$ then
        $(x'_{10}, x'_{11}) := p(\downarrow, x_0, x_1)$
        return $x'_{10} \overset{?}{=} x_{10}$
    return false

procedure CheckBwd$(x_{10}, x_{11}, x_1)$:
    if $(\uparrow, x_{10}, x_{11}) \in p$ then
        $(x'_0, x'_1) := p(\uparrow, x_{10}, x_{11})$
        return $x'_1 \overset{?}{=} x_1$
    return false
```

Mapping randomness of H_2 to randomness of H_3. Before we describe the mapping, we make the randomness r used by R explicit. The randomness r is a list containing $2 \cdot 2^{2n}$ strings of length $2n$. Then R runs deterministically given r.

With regard to the simulator, let f be a table containing an independent uniform bitstring of length n for each $i \in \{1, \ldots, 10\}$ and $x_i \in \{0,1\}^n$. Then whenever the simulator $\hat{\mathcal{S}}$ needs to set $G_i(x_i)$ to a random value, it uses the value $f(i, x_i)$ instead.

We define a map τ which maps a pair of tables (f, r) where f is the randomness used by the simulator $\hat{\mathcal{S}}$ and r is the randomness used by the random two-sided function R either to the symbol λ in case (f, r) does not lead to a good execution of H_2, or to a partial table z. The description of the map follows along the lines of the map in [17]. A partial table $z : \{1, \ldots, 10\} \times \{0,1\}^n \to \{0,1\}^n \cup \{\perp\}$ either has an actual entry for a pair (i, x), or a symbol $\perp$ which indicates that the entry is unused. This map will be such that $H_2(f, r)$ and $H_3(\tau(f, r))$ have "exactly the same behaviour" for good (f, r) (where a "good" (f, r) leads to a good execution of H_2). Whenever we refer to executions of $H_2(f, r)$ and $H_3(z)$ below, we assume that they are executed for the same distinguisher.

Definition 16. *The function $\tau(f, r)$ is defined as follows: If (f, r) is good, run a simulation of H_2 in which the distinguisher completes all chains. Consider the tables G at the end of this execution, and for any i and x let $z(i, x) := G_i(x)$ in case $x \in G_i$ and $z(i, x) :=\perp$ otherwise. If (f, r) is not good, let $\tau(f, r) := \lambda$.*

Lemma 3.41. *The probability that a distinguisher $\mathcal{D}$ outputs 1 in H_2 differs at most by $O(q^{10})/2^n$ from the probability that it outputs 1 in H_3.*

Proof. The proof of this lemma follows exactly along the lines of the proof of [17, Lemma 3.37]. So, the probability that a distinguisher $\mathcal{D}$ outputs 1 in H_2 differs from the probability that it outputs 1 in H_3 by twice the probability that an execution of H_2 is not good. By Lemma 3.23, this is given by $O(q^{10})/2^n$. □

3.4.5 Indistinguishability of the Third and Fourth Experiments

In H_3, the distinguisher accesses the random functions through the simulator. In experiment H_4, the distinguisher can instead access them directly.

Lemma 3.42. *Suppose that in $H_3(z)$ the simulator $\hat{\mathcal{S}}(z)$ eventually answers a query* $\mathrm{F}(i, x)$*. Then that query is answered with $z(i, x)$.*

Proof. The proof follows exactly as in [17, Lemma 3.38]. □

Lemma 3.43 (Indistinguishability of H_3 and H_4)**.** *The probability that a distinguisher outputs 1 in H_3 differs by at most by $O(q^{10})/2^n$ from the probability that it outputs 1 in H_4.*

Proof. The proof follows exactly along the lines of [17, Lemma 3.38]. □

Chapter 4: Indifferentiability of 5-Round Iterated Even-Mansour

In this chapter, we prove that the 5-round iterated Even-Mansour (IEM) construction with a trivial key-schedule, i.e., where all round keys are equal,[1] is indifferentiable from an ideal cipher. As mentioned in Chapter 2, this implies that the 5-round iterated Even-Mansour construction with a non-idealized key schedule is indifferentiable from an ideal cipher.

Our 5-round feasibility result improves *both* on the 5-round result of Andreeva et al. [2] for the IEM construction with an idealized key-schedule and on the 12-round feasibility result of Lampe and Seurin [43] for the IEM construction with the trivial key-schedule. Our simulator runs in time $O(q^5)$ and makes $O(q^5)$ ideal cipher queries. We remark that the security bound presented in this chapter can be improved with a more fined-grained analysis of "bad events" (see [21]).

[1] Actually we consider a slight variant of the trivial key-schedule where the first and last round keys are omitted, but both our negative and positive results are straightforward to extend to the "standard" trivial key-schedule. See Section 4.2 for a discussion.

4.1 Overview

4.1.1 Our Techniques

Our 5-round simulator follows the traditional "chain detection/completion" paradigm, pioneered by Coron et al. [17, 18, 37] for proving indifferentiability of the Feistel construction, which has since then been used for the IEM construction as well [2, 43]. (See Section 3.1 for an overview of their techniques.) However, our simulator 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 new 5-round simulator detects and completes *any* path of length three, where a path is a sequence of adjacent (round) permutation queries "chained" by the same key as in the IEM construction (and which might wrap around the ideal cipher). In contrast, the 12-round simulator of Lampe and Seurin [43] used a much more parsimonious chain detection strategy (inherited from [17, 18, 37, 53]) which allowed for a much simpler termination argument.

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

The most innovative part of the 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 [17, 18, 37, 53] the proof is "bootstrapped" by proving that certain types of paths (namely those that wrap around the IC) will be detected and completed only if the distinguisher made the corresponding IC query. Hence, assuming the distinguisher makes at most q queries, at most q such paths will be completed. In virtually all previous indifferentiability proofs, this fact easily allows to upper bound the size of permutations "history" for all other "detect zones" used by the simulator, and hence to upper bound the total number of paths that will ever be detected and completed. This is not the case for our 5-round simulator, since the "at most q wrapping paths" trick only allows 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. In order to push the argument further, we need to prove a structural property of the history of adjacent permutations P_2 and P_4. In more detail, this property is that the table maintaining answers of the simulator for P_2 (respectively, P_4) never contains four distinct input/output pairs $(x^{(i)}, y^{(i)})$, $1 \leq i \leq 4$, such that $\bigoplus_{1\leq i\leq 4}(x^{(i)} \oplus y^{(i)}) = 0$. It is rather straightforward to prove that such input/output pairs are unlikely to exist if the simulator sets answers at random, but answers are sometimes "adapted" when completing a path, which makes the proof much more complicated.

4.1.2 Related Work

As mentioned in Chapter 1, Andreeva et al. [2] and Lampe and Seurin [43] showed that the 5-round IEM construction with an idealized key-schedule, and the 12-round IEM construction with the trivial key-schedule, respectively, are indifferentiable from an ideal cipher. 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 [14, 31], known-key [3, 15], or chosen-key attacks [14,35]. A recent preprint shows that the 3-round IEM construction with a (non-invertible) idealized key-schedule is indifferentiable from an ideal cipher [36].

4.2 Preliminaries

As mentioned in Chapter 2, the iterated Even-Mansour construction EM for a trivial key-schedule is defined as follows: for an input $x \in \{0,1\}^n$ and a key $k \in \{0,1\}^n$,

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

where P_i is a permutation over $\{0,1\}^n$. See Fig. 2.2 for an illustration of a 5-round iterated Even-Mansour construction.

Indifferentiability. We recall the standard definition of indifferentiability presented in Definition 1 and for clarity, we state it directly for the IEM construction.

Definition 17. *We say that* EM *is* indifferentiable from an ideal cipher *if there exists a simulator* $\mathcal{S}$ *and a polynomial* t *such that for all distinguishers* $\mathcal{D}$ *making at most* $q = \mathsf{poly}(n)$ *queries,* $\mathcal{S}$ *runs in time* $t(q)$ *and*

$$|\Pr[\mathcal{D}^{\mathsf{EM}^{\mathbf{P}},\mathbf{P}}(1^n) = 1] - \Pr[\mathcal{D}^{\mathsf{IC},\mathcal{S}^{\mathsf{IC}}}(1^n) = 1]|$$

is negligible, where $\mathbf{P}$ *consists of random, independent permutations over* $\{0,1\}^n$ *and* IC *is an ideal cipher with key space* $\{0,1\}^n$ *and message space* $\{0,1\}^n$*.*

Recall that Definition 1 and hence Definition 17 allows the simulator $\mathcal{S}$ to depend on the number of queries q. In fact, the simulator that we show in the pseudocode (cf. Figs. 4.1 and 4.2) does not depend on q, but this simulator is efficient only with high probability, as will become clear in the proof. In Theorem 4.44, we discuss an optimized implementation of this simulator that, among other things, uses knowledge of q to abort whenever its runtime exceeds the limit of a "good" execution, thus ensuring that the simulator is efficient with probability 1.

4.3 Our Simulator

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

4.3.1 Informal Description

We start with a high-level overview of how the simulator $\mathcal{S}$ works, deferring the formal description in pseudocode to Section 4.3.2. For each $i \in \{1, \ldots, 5\}$, the simulator maintains a pair of tables P_i and P_i^{-1} with 2^n entries containing either an n-bit value or a special symbol $\perp$, allowing the simulator to keep track 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 public interface Query(i, δ, z) allowing the distinguisher to request the value $P_i(z)$ when $\delta = +$ or $P_i^{-1}(z)$ when $\delta = -$ for input $z \in \{0,1\}^n$. Upon reception of a query (i, δ, z), the simulator checks whether $P_i^{\delta}(z)$ has already been defined, and returns the corresponding value if this is the case. Otherwise, it marks query (i, δ, z) as "pending" and starts a "chain detection/completion" mechanism, called a *query cycle* in the following, in order to maintain consistency between its answers and the IC as we now explain. (We stress that some of the wording introduced here is informal and that all notions will be made rigorous in the next sections.)

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

"k-path of length $j+i-1$" (hence, paths might "wrap around" the IC).

The very simple idea at the heart of the simulator is that, before answering any distinguisher's query to some simulated permutation, it ensures that any path of length three (or more) has been preemptively extended to a "complete" path of length five $((1, x_1, y_1), \ldots, (5, x_5, y_5))$ compatible with the ideal cipher (i.e., such that $\mathsf{IC}(k, x_1) = y_5$). For this, assume that at the moment the distinguisher makes a permutation query (i, δ, z) which is not table-defined yet (otherwise the simulator just returns the answer that was preemptively set), any path of length three is complete. This means that any existing incomplete path has length at most two. These length-2 paths will be called (table-defined[2]) *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 $\mathsf{IC}(k, x_1) = y_5$, so that possible types are $(1,2)$, $(2,3)$, $(3,4)$, $(4,5)$, and $(5,1)$). Let us also call the direct input to permutation P_{i+2} and the inverse input to permutation P_{i-1} when extending the 2chain in the natural way the resp. right and left *endpoint* of the 2chain.[3]

The "pending" permutation query (i, δ, z) asked by the distinguisher, once

[2] 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.

[3] 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.

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

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

Table 4.1: The five types of $(i, i+1)$-query cycles of the simulator.

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

new $(i, i+1)$-2chain to detect and trigger. Note that the simulator can completely determine which 2chains should be triggered *before* starting the completion process itself.

Once all 2chains that must eventually be completed have been detected and "triggered", the simulator starts the completion process. First, it randomly samples all "pending" queries (which are necessarily of the form $(i+2, +, \cdot)$ or $(i-1, -, \cdot)$ for a query cycle of type $(i, i+1)$). This "all-at-the-same-time" randomness sampling is reminiscent of the simulator in [28]. Finally, for all triggered 2chains, it adapts the corresponding path by computing the corresponding input x_{i+3} and output y_{i+3} at position $i+3$ and "forcing" $P_{i+3}(x_{i+3}) = y_{i+3}$. In case some collision occurs when trying to assign a value for some permutation, the simulator aborts. All important characteristics of an $(i, i+1)$-query cycle are summarized in Table 4.1.

4.3.2 Formal Description

We now give the full pseudocode for the simulator. The simulator has access to the ideal cipher, that we formally capture with two interfaces $\mathrm{Enc}(k, x)$ and $\mathrm{Dec}(k, y)$ for encryption and decryption respectively. 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 $P_1, \ldots, P_5$ when needed is also made explicit in the pseudocode through uniformly random permutations tapes $\mathbf{p} = (p_1, p_1^{-1}, \ldots, p_5, p_5^{-1})$. By random permutation tapes, we mean that $p_i : \{0,1\}^n \to \{0,1\}^n$ is a uniformly random permutation and p_i^{-1} is its inverse.

In the pseudocode and more generally throughout this chapter, the result of arithmetics on positions is automatically wrapped into the range $\{1, 2, 3, 4, 5\}$. For any table or tape $\mathcal{T}$ and $\delta \in \{+, -\}$, we let $\mathcal{T}^\delta$ be $\mathcal{T}$ if $\delta = +$ and be $\mathcal{T}^{-1}$ if $\delta = -$. Given a list L, $L \hookleftarrow x$ means that x is appended to L. If the simulator aborts (line 58), we assume it returns a special symbol $\perp$ in response to the distinguisher's query being processed.

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 58.

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

**public procedure** SimQuery$(i, \delta, z)$:
  **if** $P_i^\delta(z) = \bot$ **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
```

Figure 4.1: Pseudocode of the simulator.

```
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 $\text{Dec}(k, y_i)$
  else return $y_i \oplus k$

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

private procedure Check($k, x_1, y_5$):
  return $\text{Enc}(k, x_1) = y_5$ \\ $\mathsf{G}_1$
  return $T(k, x_1) = y_5$ \\ $\mathsf{G}_2, \mathsf{G}_3, \mathsf{G}_4$
```

Figure 4.2: Pseudocode of the simulator (contd.)

4.4 Proof of Indifferentiability

Our main result is the following theorem.

Theorem 4.1. *The probability that a distinguisher $\mathcal{D}$ making at most q queries outputs 1 in an interaction with $(\mathsf{IC}, \mathcal{S}^{\mathsf{IC}})$ and the probability that it outputs 1 in an interaction with $(\mathsf{EM}^{\mathbf{P}}, \mathbf{P})$ differ by at most $O(q^{38}/2^n)$. Moreover, $\mathcal{S}$ runs in time polynomial in $O(q^5)$.*

Furthermore, the bounds hold even if the distinguisher is allowed to make q permutation queries in each position (i.e., it can call 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 proof of the theorem is given in the rest of the chapter, with the indifferentiability simulator being the one described in Section 4.3.

4.4.1 Proof Overview

Our proof uses a sequence of games G_1, G_2, G_3 and G_4 as described in Figure 4.3, with G_1 being the simulated world and G_4 being the real world. We fully describe the intermediate worlds that will be used in the indifferentiability proof. The distinguisher $\mathcal{D}$ has access to the public interface $\text{Query}(i, \delta, z)$, which in the ideal world is answered by the simulator, and to the ideal cipher/IEM construction interface, that we (as mentioned earlier) formally capture with two interfaces $\text{Enc}(k, x)$ and $\text{Dec}(k, y)$ for encryption and decryption respectively. We will refer to queries to any of these two interfaces as *cipher queries*, by opposition to *permutation queries* made to interface $\text{Query}(\cdot, \cdot, \cdot)$. In the ideal world, cipher queries are answered by an ideal cipher IC. As mentioned previously, we make the randomness of IC explicit through two random tapes ic, $\mathsf{ic}^{-1} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ such that for any $k \in \{0,1\}^n$, $\mathsf{ic}(k, \cdot)$ is a uniformly random permutation and $\mathsf{ic}^{-1}(k, \cdot)$ is its inverse. Hence, in the ideal world, a query $\text{Enc}(k, x)$, resp. $\text{Dec}(k, y)$, is simply answered with $\mathsf{ic}(k, x)$, resp. $\mathsf{ic}^{-1}(k, y)$. The randomness used by the simulator for lazily sampling permutations $P_1, \ldots, P_5$ when needed has also been made explicit in the pseudocode through uniformly random permutations tapes $\mathbf{p} = (p_1, p_1^{-1}, \ldots, p_5, p_5^{-1})$. 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 by G_4. In this world, queries to $\text{Query}(i, \delta, z)$ are simply answered with the corresponding value stored in the random permutation tapes $\mathbf{p}$, while queries to Enc or Dec are answered by the IEM construction based on random permutations $\mathbf{p}$. Randomness in G_4 is fully captured by $\mathbf{p}$.

Intermediate Games. The indifferentiability proof relies on two intermediate games G_2 and G_3. In game G_2, the Check procedure used by the simulator (cf. Fig. 4.2) 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 a 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 $\mathsf{EM}^{\mathbf{p}}$ when transitioning from G_2 to G_3).

The pseudocode for the public (i.e., accessible by the distinguisher) procedures Query, Enc, and Dec is given in Fig. 4.3, together with helper procedures that capture the changes from games G_1 to G_4. Lines commented with "\\G_i" apply only to game G_i. In the pseudocode, the result of arithmetics on positions is automatically wrapped into the range $\{1, 2, 3, 4, 5\}$. For any table or tape $\mathcal{T}$ and $\delta \in \{+, -\}$, we let $\mathcal{T}^\delta$ be $\mathcal{T}$ if $\delta = +$ and be $\mathcal{T}^{-1}$ if $\delta = -$.

Tables T and T^{-1} are used to record the cipher queries that have been issued (by the distinguisher *or* the simulator).

Throughout the proof we will fix an arbitrary information-theoretic distinguisher $\mathcal{D}$ with q queries, meaning that $\mathcal{D}$ has unlimited computation power and

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

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

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

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

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

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

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

private procedure Check$(k, x_1, y_5)$:
  return Enc$(k, x_1) = y_5$ \\ $\mathsf{G}_1$
  return $T(k, x_1) = y_5$ \\ $\mathsf{G}_2, \mathsf{G}_3, \mathsf{G}_4$
```

Figure 4.3: 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.

can issue a limited number of queries. Using a trick in [23], we allow the distinguisher $\mathcal{D}$ to make q cipher queries and q permutation queries in *each* position, as described in Theorem 4.1. This trick allows us to obtain a better security bound, even though the distinguisher can make more queries than usual.[4] We can assume without loss of generality that $\mathcal{D}$ is *deterministic*, as any distinguisher can be derandomized using the "optimal" random tape and achieve at least the same advantage.

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

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

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

Our proof starts with discussions about the game G_2, which will be the starting point of the transitions. As usual, there are *bad events* that might cause the simulator to fail. We will prove that bad events are unlikely, and show properties of *good executions* in which bad events don't occur. The proof of efficiency of the simulator (in good executions of G_2) is the most interesting and technical part of this paper, which is given in Section 4.4.3. During the proof of efficiency we also obtain upper bounds on the sizes of the tables and on the number of calls to each procedure, which will be a crucial component for the transitions.

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

For the G_1-G_2 transition, note that the only difference between the two games is in Check. If the simulator is efficient, the probability that the two executions diverge in a call to Check is negligible. Therefore, if an execution of G_2 is good, it is identical to the G_1-execution with the same random tapes except with negligible probability (cf. Lemma 4.42). In particular, this implies that an execution of G_1 is efficient with high probability.

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

We will show that if a G_3-execution has a preimage, then the answers of the permutation queries output by the simulator must be compatible with the random permutation tapes (cf. Lemma 4.51). 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 $\mathrm{Query}(5,+,x_5)$, or else received the value x_5 as an answer to a query of the form $\mathrm{Query}(5,-,y_5)$, 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 $\mathrm{Query}(5,\cdot,\cdot)$; moreover, there are at most $2q$ possibilities for y_3, as already noted; and we have the relation

$$y_3 \oplus x_5 = x_4 \oplus y_4 \tag{4.1}$$

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 $x_4 \neq x_4'$.[5] Hence, with high probability (4.1) has at most a unique solution x_4, y_4 for each y_3,

[5] 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.

x_5, and scenario (a) accounts for at most $2q^2$ path completions (one for each possible left-hand side of (4.1)) 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

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

(both sides are equal to x_5) implies

$$y_3 \oplus y'_3 = x_4 \oplus y_4 \oplus x'_4 \oplus y'_4 \tag{4.2}$$

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)[6]) at most one solution to the RHS of (4.2) for each LHS; hence, scenario (b) accounts for at most $4q^2$ path completions[7] 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 Query$(2, \cdot, \cdot)$ and "wraparound" path completions

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

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

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.

4.4.2 Properties of the Second Experiment

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

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

Queries and 2chains. The central notion for reasoning about the simulator is the notion of 2chain, that we develop below.

Definition 1. 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 2. 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 3. A permutation query (i, δ, z) is *table-defined* if $P_i^\delta(z) \neq \bot$, and *table-undefined* otherwise. Similarly, a cipher query (δ, k, z) is *table-defined* if $T^\delta(k, z) \neq \bot$, and *table-undefined* otherwise.

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

Note that if $(i, +, x_i)$ is table-defined and $P_i(x_i) = y_i$, then necessarily $(i, -, y_i)$ is also table-defined and $P_i^{-1}(y_i) = x_i$. Indeed, tables P_i and P_i^{-1} are only modified in procedure Assign, where existing entries are never overwritten due to the check at line 58. Thus the two tables always encode a partial permutation and its inverse, i.e., $P_i(x_i) = y_i$ if and only if $P_i^{-1}(y_i) = x_i$. Hence, we often say that a triple (i, x_i, y_i) is table-defined as a shorthand to mean that both $(i, +, x_i)$ and $(i, -, y_i)$ are table-defined with $P_i(x_i) = y_i$ and $P_i^{-1}(y_i) = x_i$.

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

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

Remark 4.2. *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 4.3 below.*

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

Remark 4.3. *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 6 (endpoints)**.** Let $C = (i, i+1, y_i, x_{i+1}, k)$ be a table-defined 2chain.

The *right endpoint* of C, denoted $r(C)$ is defined as

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

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

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

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

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

Definition 7. A *complete path* with key k is a 5-tuple of table-defined permutation queries $((1, x_1, y_1), \ldots, (5, x_5, y_5))$ such that

$$y_i \oplus x_{i+1} = k \text{ for } i = 1, 2, 3, 4 \text{ and } T(k, x_1) = y_5. \tag{4.3}$$

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

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

Lemma 4.4. *In any execution of* G_2*, any 2chain belongs to at most one complete path.*

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

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

Definition 8 (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 4.5 (a) and Table 4.1).

The portion of the query cycle consisting of calls to FindNewPaths at line 8 is called the *detection phase* of the query cycle; the portion of the query cycle consisting of calls to ReadTape at line 9 and to AdaptPath at line 10 is called the *completion phase* of the query cycle.

During a query cycle, we say that a permutation query (i, δ, z) is *pending* (or simply that z is pending when i and δ are clear from the context) if it is appended by the simulator to list $\mathsf{Pending}$ at line 7, 30, or 48. 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 $\mathsf{Triggered}$ at line 27 or 45.

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

Lemma 4.5. *During an $(i, i+1)$-query cycle whose initiating query was (i_0, δ_0, z_0), the following properties always hold:*

(a) Only 2chains of type $(i, i+1)$ are triggered.

(b) Only permutations queries of type $(i-1, -)$ or $(i+2, +)$ are pending.

(c) Any 2chain that is triggered was table-defined at the beginning of the query cycle.

(d) At the end of the detection phase, any pending query is either the initiating query, or the endpoint of a triggered 2chain.

(e) If a 2chain C is triggered during the query cycle, and the simulator does not abort, then C is complete at the end of the query cycle.

Proof. The proof of (a) and (b) proceeds by inspection of the pseudocode: note that calls to FindNewPaths$(i-1,-,\cdot)$ can only add 2chains of type $(i,i+1)$ to $\mathsf{Triggered}$ and permutations queries of type $(i+2,+)$ to $\mathsf{Pending}$, whereas calls to FindNewPaths$(i+2,+,\cdot)$ can only add 2chains of type $(i,i+1)$ to $\mathsf{Triggered}$ and permutations queries of type $(i-1,-)$ to $\mathsf{Pending}$. 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 $\mathsf{Triggered}$, and only permutation queries of type $(i-1,-)$ or $(i+2,+)$ will ever be added to $\mathsf{Pending}$. 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 28 or line 46 as the endpoint of a triggered 2chain. Finally, the proof of (e) follows from the fact that, assuming the simulator does not abort, all values computed during the call to AdaptPath(C) form a complete path to which C obviously belongs. □

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

Lemma 4.6. *In any execution of G_2, the following properties hold:*

(a) During a $(1,2)$-query cycle, tables T/T^{-1} are only modified during the detection phase by calls to Enc$(\cdot,\cdot)$ *resulting from calls to* Prev$(1,\cdot,\cdot)$ *at line 28.*

(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 55.*

(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 53.*

(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 46.*

(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 a moment of thinking should make it clear that they are always called with arguments (x_1,k), resp. (y_5,k) which were previously used during the detection phase, so that this cannot modify tables T/T^{-1} any more. □

We also introduce the following helper lemma.

Lemma 4.7. *Consider any execution of G_2. Assume that two table-defined $(i,i+1)$-2chains $C=(i,i+1,y_i,x_{i+1},k)$ and $C'=(i,i+1,y'_i,x'_{i+1},k')$ have the same key and a common non-dummy endpoint, i.e., are such that $k=k'$ and $r(C)=r(C')\neq\perp$ or $\ell(C)=\ell(C')\neq\perp$. Then $C=C'$.*

Proof. We show the result for the case where $k=k'$ and $r(C)=r(C')$, the case where $\ell(C)=\ell(C')$ is similar. Consider first the case where $i\in\{1,2,3,5\}$. By definition of the right endpoint, this implies that $P_{i+1}(x_{i+1})=P_{i+1}(x'_{i+1})$ and hence

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

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

Definition 9. Consider a permutation query (i_0, δ_0, z_0) or a cipher query (δ_0, k_0, z_0) made by the distinguisher. Let $\mathcal{H}'$ be the multiset of all n-bit strings appearing in the list of table-defined permutation or cipher queries and of all keys and non-dummy endpoints of any table-defined 2chain when the query occurs. That is, write the list of all table-defined permutation queries (i, x_i, y_i), all table-defined cipher queries (δ, k, z), all keys and non-dummy endpoints of all table-defined 2chains, and count each n-bit string as many times as it appears in this list. Then we define the "history" with respect to a permutation query (i_0, δ_0, z_0) made by the distinguisher as the multiset

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

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

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

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

Remark 4.8. *The sets in the above definition are time-dependent and do not include the values added to the tables* during *the query cycle or due to the distinguisher's cipher query (except z_0 for a permutation query and k_0 and z_0 for a cipher query). Note also that keys and non-dummy endpoints of table-defined 2chains can* always *be expressed as the xor of at most three n-bit values appearing in the list of table-defined permutation of cipher queries, so that strictly speaking the set $\mathcal{H}'$ could consist of these values only. However, the current definition of the history simplifies the discussion along the proof.*

Definition 10. Given a query cycle, we denote $\mathcal{P}$ the multiset of random values read by ReadTape on tapes $(p_1, p_1^{-1}, \ldots, p_5, p_5^{-1})$ in the current query cycle. Given a query cycle or a distinguisher's cipher query, we denote $\mathcal{C}$ the multiset of random values read by Enc and Dec on tapes ic or ic^{-1}.[8]

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

Definition 11. Let $\mathcal{H}^{\oplus i}$ be the set of values equal to the exclusive-or of exactly i *distinct* elements in $\mathcal{H}$, and let $\mathcal{H}^{\oplus 0} := \{0\}$. The sets $\mathcal{P}^{\oplus i}$ and $\mathcal{C}^{\oplus i}$ are defined similarly.[9]

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

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

Definition 12. $\mathsf{BadPerm}$ is the event that the exclusive-or of i distinct elements of $\mathcal{P}$ equals the exclusive-or of j distinct elements of $\mathcal{H}$, i.e. $\mathcal{P}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$, with $i \geq 1$, $j \geq 0$, and $i + j \leq 8$.

Definition 13. BadIC is the event that the exclusive-or of i distinct elements of $\mathcal{C}$ equals the exclusive-or of j distinct elements of $\mathcal{H}$, i.e. $\mathcal{C}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$, with $i \geq 1$, $j \geq 0$, and $i + j \leq 2$.

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

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

Definition 14 (Good Executions)**.** An execution of G_2 is said to be *good* if neither $\mathsf{BadPerm}$ nor BadIC occurs in the execution.

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 15 (safe query cycle)**.** A query cycle is said to be *safe* if for any 2chain C triggered during the query cycle, both endpoints of C were dummy or table-undefined[10] at the beginning of the query cycle.

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

As just explained, our first step will be to prove that the simulator does not abort during a safe query cycle. The simulator can only abort in procedure Assign which is only called during the completion phase. Moreover, this completion phase can be split into two sub-phases: first, the simulator calls ReadTape(i, δ, z) for each pending query (i, δ, z), and then it calls AdaptPath(C) for each triggered 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 4.5 (c), C must be table-defined at the beginning of the query cycle, hence, by definition of the history $\mathcal{H}$, any endpoint of C which was non-dummy at the beginning of the query cycle is in $\mathcal{H}$. The following lemma clarifies the situation in case a triggered 2chain has a dummy endpoint at the beginning of the query cycle.

Lemma 4.9. *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.*

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

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 $\mathrm{FindNewPaths}(3, -, \cdot)$ or to $\mathrm{FindNewPaths}(1, +, \cdot)$ in a $(4,5)$-query cycle. We first consider the case where it is triggered during a call to $\mathrm{FindNewPaths}(3, -, \cdot)$. Inspection of the pseudocode then shows that right after C has been triggered, a call to $\mathrm{Dec}(k, y_5)$ resulting from a call to $\mathrm{Next}(5, y_5, k)$ at line 46 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 $\mathrm{FindNewPaths}(1, +, \cdot)$. Note that during a call to $\mathrm{FindNewPaths}(1, +, x_1)$, the simulator triggers a table-defined 2chain $C = (4, 5, y_4, x_5, k)$ only if $\mathrm{Check}(k, x_1, y_5)$, where $y_5 = P_5(x_5)$, is true, which can never be if $r(C) = \perp$. This implies that C had a non-dummy right endpoint at the beginning of the query cycle. This is because by Lemma 4.6, we know that in a $(4,5)$-query cycle the entries in tables T/T^{-1} are modified only during the detection phase by calls to $\mathrm{Dec}(\cdot, \cdot)$ resulting from calls to $\mathrm{Next}(5, \cdot, \cdot)$ at line 46. 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 4.7 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 20 in the pseudocode. □

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

Lemma 4.10. *Consider a safe query cycle in a good execution of* G_2*. Then the simulator does not abort during the calls to* ReadTape *occurring during the query cycle.*

Proof. Let τ_0 denote the beginning of the query cycle. Assume towards a contradiction that the simulator aborts in a call to ReadTape during a safe $(i, i+1)$-query cycle. By Lemma 4.5 (b), ReadTape can only be called for permutation queries of type $(i-1, -)$ or $(i+2, +)$. Assume that the simulator aborts in a call to ReadTape$(i+2, +, x_{i+2})$ (the case of a call to ReadTape$(i-1, -, y_{i-1})$ is similar). This means that we have either $x_{i+2} \in P_{i+2}$ or $p_{i+2}(x_{i+2}) \in P_{i+2}^{-1}$ (where p_{i+2} is the random permutation tape) when the call occurs. Assume first that $x_{i+2} \in P_{i+2}$ when the call to ReadTape occurs. We first show that x_{i+2} is table-undefined (i.e., $x_{i+2} \notin P_{i+2}$) just before the completion phase starts. Procedure ReadTape is only called on pending queries, hence, by Lemma 4.5 (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 4.9, $x_{i+2} = r(C)$ is non-dummy when the completion phase starts and is in $\mathcal{C}$. If x_{i+2} is table-defined when the completion phase starts, then it was already table-defined at τ_0, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens, contradicting the assumption that the execution is good. In all cases, we see that x_{i+2} is table-undefined just before the completion phase starts. Hence, if $x_{i+2} \in P_{i+2}$ when the call to ReadTape occurs, this can only be due to another call to $\text{ReadTape}(i+2, +, x_{i+2})$ 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 29 and 47. Assume now that $p_{i+2}(x_{i+2}) \in P_{i+2}^{-1}$ when the call to ReadTape occurs. If $p_{i+2}(x_{i+2}) \in P_{i+2}^{-1}$ at the beginning of the query cycle, then $p_{i+2}(x_{i+2}) \in \mathcal{P} \cap \mathcal{H}$ and $\mathsf{BadPerm}$ occurs. Otherwise, this can only happen due to another call to $\text{ReadTape}(i+2, +, x'_{i+2})$ 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. □

Now that we proved that during a safe query cycle, the simulator does not abort during calls to ReadTape, we know that it will try to "adapt" each triggered 2chain C by calling $\text{AdaptPath}(C)$. 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 4.11. *Consider a safe $(i, i+1)$-query cycle in a good execution of G_2. Let $C = (i, i+1, y_i, x_{i+1}, k)$ be a triggered 2chain, and assume that* $\text{AdaptPath}(C)$ *is* *called during the completion phase.*[11] *Consider the resulting call to* Assign$(i+3, x_{i+3}, y_{i+3})$[12] *at line* 56*. Then*

- *if* $i \neq 3$*,* $x_{i+3} = p_{i+2}(r(C)) \oplus k$ *where* $p_{i+2}(r(C)) \in \mathcal{P}$ *and* $k \in \mathcal{H}$*;*
- *if* $i = 3$*,* $x_{i+3} = x_1 = \mathsf{ic}^{-1}(k, p_5(r(C))) \in \mathcal{C}$*;*
- *if* $i \neq 2$*,* $y_{i+3} = p_{i-1}^{-1}(\ell(C)) \oplus k$ *where* $p_{i-1}^{-1}(\ell(C)) \in \mathcal{P}$ *and* $k \in \mathcal{H}$*;*
- *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 4.10) and inspection of the pseudocode. Note that k is the key of C which is table-defined at the beginning of the query cycle (since it is triggered), hence by definition $k \in \mathcal{H}$. Consider now the case $i = 3$, i.e., the completion of a $(3,4)$-2chain $C = (3, 4, y_3, x_4, k)$ during a $(3,4)$-query cycle. By Lemma 4.10, the simulator does not abort during the call to Assign$(5, x_5, y_5)$ resulting from the call to ReadTape$(5, +, x_5)$, where $x_5 = P_4(x_4) \oplus k$ and $y_5 = p_5(x_5) \in \mathcal{P}$. Hence, when the call to AdaptPath(C) occurs, by inspection of the pseudocode, a call to Next$(5, y_5, k)$ occurs at line 53, resulting in a call to Dec(k, y_5). We argue that the cipher query $(-, k, y_5)$ is table-undefined when this call occurs. If it is table-defined at the beginning of the query cycle, then by definition $y_5 \in \mathcal{H}$, so that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs, contradicting the assumption

[11] 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.

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

that the execution is good. If y_5 is table-undefined when the query cycle begins, but table-defined when the call to AdaptPath(C) occurs, then, since by Lemma 4.6, tables T/T^{-1} are not modified during the detection phase in a $(3,4)$-query cycle, this can only be due to a call to AdaptPath(C') for another triggered 2chain C' which caused a call to Dec(k, y_5). But this would mean that C and C' have the same key and $r(C) = r(C') = y_5 \neq \perp$, which by Lemma 4.7 implies $C = C'$, which is impossible since a 2chain is triggered at most once in a query cycle because of the checks at lines 20 and 38. Hence, $(-, k, y_5)$ is table-undefined when the call to Dec(k, y_5) occurs and hence $x_1 = \mathsf{ic}^{-1}(k, y_5) \in \mathcal{C}$. □

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

Lemma 4.12. *Consider a safe query cycle in a good execution of* G_2. *Then the simulator does not abort during the calls to* AdaptPath *occurring during the query cycle.*

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

Consider first the case $i \neq 3$. Then, by Lemma 4.11, we have $x_{i+3} = p_{i+2}(x_{i+2}) \oplus k$, where $x_{i+2} = r(C)$, $p_{i+2}(x_{i+2}) \in \mathcal{P}$, and $k \in \mathcal{H}$. If $x_{i+3} \in P_{i+3}$ at the beginning of the query cycle, then by definition $x_{i+3} \in \mathcal{H}$, so that $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$, which means $\mathsf{BadPerm}$ 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 4.11, 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

$$p_{i+2}(x_{i+2}) \oplus p_{i+2}(x'_{i+2}) = k \oplus k'. \tag{4.4}$$

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

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

which by Lemma 4.7 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 20 and 38. Hence, $x_{i+2} \neq x'_{i+2}$ and Eq. (4.4) implies that $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$, 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 4.11, 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 4.11, 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

$$\mathsf{ic}^{-1}(k, y_5) = \mathsf{ic}^{-1}(k', y'_5). \tag{4.5}$$

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 4.7 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. (4.5) implies that $\mathcal{C}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$, i.e., BadIC occurs.

This concludes the proof. □

Lemma 4.13. *In a good execution of* G_2*, the simulator does not abort during a safe query cycle.*

Proof. This follows directly from Lemmas 4.10 and 4.12 since the simulator can only abort during calls to ReadTape and AdaptPath. □

It remains now to show that all query cycles in a good execution are safe. The key observation for this is that the simulator ensures that, at the end of the completion phase, any table-defined 2chain with a table-defined endpoint (one can think of it as a "3chain") necessarily belongs to a complete path as per Definition 7. We show that this property is preserved by any distinguisher's cipher query (a direct consequence of Lemma 4.14) and by any query cycle (Lemmas 4.15, 4.16 and 4.17), and deduce that this holds at the beginning of any query cycle (Lemma 4.19). We also show that a 2chain which is complete at the beginning of a query cycle cannot be triggered (Lemma 4.21). 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 4.14. *Consider a cipher query made by the distinguisher in a good execution of G_2 that modifies tables T/T^{-1}. Let C be a 2chain. Then the following two properties hold:*

(a) If C is table-undefined before the query, then C is still table-undefined after the query has been answered.

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

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

We then prove (b). We prove it in the case of the right endpoint of a $(4,5)$-2chain (the case of the left endpoint of a $(1,2)$-2chain follows by symmetry). Let

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

Lemmas 4.15, 4.16, and 4.17 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. In the three lemmas, we say that "a complete path has been triggered" if any of the five 2chains belonging to the complete path has been triggered. Note also that the three lemmas assume a safe query cycle, which by Lemma 4.13 implies that the simulator does not abort, which by Lemma 4.5 (e) implies that any triggered 2chain is complete at the end of the query cycle.

Lemma 4.15. *Consider a safe query cycle in a good execution of G_2. Let C be a 2chain which is table-defined at the beginning of the query cycle and such that one of 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 4.6, the simulator only calls Enc during such query cycles. This means that $y_5 = \mathsf{ic}(k, x_1)$ must have been read during the query cycle, and since $y_5 \in \mathcal{H}$, $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC occurs, contradicting the assumption that the execution is good.

- *Case of a* $(3,4)$*-query cycle.* By Lemma 4.6, the simulator only calls Dec during the completion phase in this case. Moreover, it is easy to check from the pseudocode that the call $\mathrm{Dec}(k, y_5)$ can only occur during the call to $\mathrm{AdaptPath}(D)$, where $D = (3, 4, y_3, x_4, k)$ with $x_4 = P_4^{-1}(y_4)$ and $y_3 = x_4 \oplus k$. By Lemma 4.5 (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 4.6, the simulator only calls Dec during the detection phase in this case. Moreover, it is easy to check from the pseudocode that the call to $\mathrm{Dec}(k, y_5)$ can only occur just after C has been triggered. Hence, by Lemma 4.5 (e), C is necessarily complete at τ_1.

- *Case of a* $(5,1)$*-query cycle.* By Lemma 4.6, 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 4.16. *Consider a safe query cycle in a good execution of* G_2*. Let* C *be a 2chain which is table-defined at the beginning of the query cycle and such that one of the endpoints of* C *is table-undefined at the beginning of the query cycle and table-defined at the end of the query cycle. Then, at the end of the query cycle,* C *belongs to a complete path which was triggered during the query cycle.*

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

Case of an $(i, i+1)$-2chain $C = (i, i+1, y_i, x_{i+1}, k)$. We consider the case where this is the right endpoint of C which goes from non-dummy and table-undefined to table-defined during the query cycle (the case of the left endpoint follows by symmetry). Since $r(C)$ is non-dummy and table-undefined at τ_0, necessarily $(i+2, +, r(C))$ became pending during the query cycle (otherwise it would still be table-undefined at τ_1). Since C is table-defined at τ_0, C was necessarily triggered (either during the call to FindNewPaths$(i+2, +, r(C))$, or during the call the FindNewPaths$(i-1, -, \ell(C))$ which then made $r(C)$ pending). Hence, by Lemma 4.5 (e), C is complete at τ_1.

Case of an $(i+1, i+2)$-2chain $C = (i+1, i+2, y_{i+1}, x_{i+2}, k)$. First, note that since P_i/P_i^{-1} are not modified during the query cycle, the left endpoint of C cannot go from non-dummy and table-undefined to table-defined during the query cycle, hence this is necessarily the right endpoint of C which does. Since $r(C)$ is non-dummy at τ_0, then by definition $x_{i+3} \coloneqq r(C)$ is in $\mathcal{H}$. The only way x_{i+3} can become table-defined during the query cycle is because of a call to 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 4.11, if $i \neq 3$ we have $x'_{i+3} = p_{i+2}(r(C')) \oplus k'$ where $k' \in \mathcal{H}$ and $p_{i+2}(r(C'))$ is read during the query cycle, so that $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ happens, whereas if $i = 3$ then $x'_{i+3} = x'_1 = \mathsf{ic}^{-1}(k', p_5(r(C')))$ is read during the query cycle, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens. In all cases this contradicts the assumption that the execution is good.

Case of an $(i+2, i+3)$-2chain $C = (i+2, i+3, y_{i+2}, x_{i+3}, k)$. First, note that since P_{i+1}/P^{-1}_{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. Note that since $x_{i+4} = x_{i-1} \coloneqq r(C)$ is non-dummy at τ_0, by definition $x_{i-1} \in \mathcal{H}$. The only way x_{i-1} can become table-defined during the $(i, i+1)$-query cycle is because of a call to Assign$(i-1, p^{-1}_{i-1}(y_{i-1}), y_{i-1})$ with $x_{i-1} = p^{-1}_{i-1}(y_{i-1})$ resulting from a call to ReadTape$(i-1, -, y_{i-1})$. This implies that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and hence $\mathsf{BadPerm}$ occurs, contradicting the assumption that the execution is good.

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

Lemma 4.17. *Consider a safe query cycle in a good execution of* G_2. *Let* C *be a 2chain such that* *(i) at the beginning of the query cycle, C is table-undefined;*

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

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

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

Case of an $(i, i+1)$-2chain $C = (i, i+1, y_i, x_{i+1}, k)$. Since P_i/P_i^{-1} and P_{i+1}/P_{i+1}^{-1} are not modified during an $(i, i+1)$-query cycle, and moreover tables T/T^{-1} are not modified during a $(5, 1)$-query cycle by Lemma 4.6, 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 Next(k, y_5) at line 46. Yet this would mean that $x_1 = \mathsf{ic}^{-1}(k, y_5)$, so that

$\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC occurs, contradicting the assumption that the execution is good. In all cases, we see that x_{i+2} was necessarily table-undefined at τ_0.

We now distinguish two cases depending on which endpoint of C is table-defined at τ_1. Assume first that this is the left endpoint, and let y_i be the value of $\ell(C)$ at τ_1. Since tables P_i/P_i^{-1} are not modified during the query cycle, y_i was already table-defined at τ_0. Let $x_{i+1} = P_{i+1}^{-1}(y_{i+1})$ and $D = (i, i+1, y_i, x_{i+1}, k)$. Then D was table-defined at τ_0, its right endpoint was either dummy or equal to x_{i+2} and hence table-undefined at τ_0, and at τ_1 its right endpoint is table-defined since C is table-defined. Hence, by Lemmas 4.15 and 4.16, 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 this is the right endpoint of C which is table-defined at τ_1 and let x_{i+3} denote the value of $r(C)$ at τ_1. Since x_{i+2} was table-undefined at τ_0 and we are considering an $(i, i+1)$-query cycle, x_{i+2} necessarily became pending during the query cycle and a call to Assign$(i+2, x_{i+2}, p_{i+2}(x_{i+2}))$ occurred. By Lemma 4.5 (d), x_{i+2} is either the initiating query, in which case it is in $\mathcal{H}$ by definition, or the endpoint of some triggered $(i, i+1)$-2chain D, in which case it is in $\mathcal{H}$ by definition if $r(D)$ is non-dummy at τ_0, or in $\mathcal{C}$ if $r(D)$ is dummy at τ_0 by Lemma 4.9, which might only happen when $i = 4$. We now distinguish three sub-cases depending on i:

- Case $i \in \{1, 2, 5\}$. Then C is neither a $(4,5)$- nor a $(5,1)$-2chain and its right endpoint is given by $x_{i+3} = p_{i+2}(x_{i+2}) \oplus k = p_{i+2}(x_{i+2}) \oplus y_{i+1} \oplus x_{i+2}$. Assume first that x_{i+3} was table-defined already at τ_0, so that $x_{i+3} \in \mathcal{H}$.

Then $p_{i+2}(x_{i+2}) = x_{i+3} \oplus k = x_{i+3} \oplus y_{i+1} \oplus x_{i+2} \in \mathcal{H}^{\oplus 3}$ (recall $x_{i+2} \in \mathcal{H}$ if $i \neq 4$), so $\mathcal{P} \cap \mathcal{H}^{\oplus 3} \neq \emptyset$ and $\mathsf{BadPerm}$ happens, contradicting the assumption that the execution is good. Assume now that x_{i+3} was table-undefined at τ_0. Then it can only become table-defined during the query cycle because of a call to 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 4.11, we have $x'_{i+3} = p_{i+2}(x'_{i+2}) \oplus k'$ where $x'_{i+2} = r(C')$, $p_{i+2}(x'_{i+2}) \in \mathcal{P}$ and $k' \in \mathcal{H}$. Hence, $x'_{i+3} = x_{i+3}$ implies that

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

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

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

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

- Case $i = 3$. Then we are considering a $(3, 4)$-query cycle, $C = (4, 5, y_4, x_5, k)$ is a $(4, 5)$-2chain, and its right endpoint at τ_1 is given by $x_{i+3} = x_1 = T^{-1}(k, p_5(x_5))$. If $(-, k, p_5(x_5))$ was already table-defined at τ_0, then $p_5(x_5) \in \mathcal{H}$, so that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs. Hence, $(-, k, p_5(x_5))$ became table-defined during the query cycle, which implies that at τ_1, the 2chain $(5, 1, p_5(x_5), x_1, k)$ belongs to a complete path that was triggered during the query cycle. It is easy to see that C belongs to the same complete path at τ_1.

- Case $i = 4$. Then we are considering a $(4, 5)$-query cycle, $C = (5, 1, y_5, x_1, k)$ is a $(5,1)$-2chain, and its right endpoint at τ_1 is given by $x_{i+3} = x_2 = p_1(x_1) \oplus k$. If $(-, k, y_5)$ was table-undefined at τ_0, then it became table-defined during the query cycle, so that C belongs to a complete path which was triggered during the query cycle. Assume now that $T^{-1}(k, y_5) = x_1$ already at τ_0, so that $k \in \mathcal{H}$. First, if x_2 was table-defined already at τ_0, then since $p_1(x_1) = x_2 \oplus k$ one has $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ happens. If x_2 was table-undefined at τ_0, then it can only become table-defined during the query cycle because of a call to $\text{Assign}(2, x_2', y_2')$ with $x_2' = x_2$ resulting from a call to $\text{AdaptPath}(C')$, where $C' = (4, 5, y_4', x_5', k')$ has been triggered during the query cycle. By Lemma 4.11, we have $x_2' = p_1(x_1') \oplus k'$ where $x_1' = r(C')$, $p_1(x_1') \in \mathcal{P}$, and $k' \in \mathcal{H}$. Hence, $x_2' = x_2$ implies that $p_1(x_1) \oplus p_1(x_1') = k \oplus k'$. Hence, if $x_1 \neq x_1'$, then $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and $\mathsf{BadPerm}$ occurs, whereas if $x_1 = x_1'$, then

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

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

Case of an $(i+2, i+3)$-2chain $C = (i+2, i+3, y_{i+2}, x_{i+3}, k)$. Assume first that the left endpoint of C is table-defined at τ_1 and denote y_{i+1} the value of $\ell(C)$ at τ_1. Since tables P_{i+1}/P_{i+1}^{-1} are not modified during an $(i, i+1)$-query cycle, y_{i+1} is already table-defined at τ_0. Let $D = (i+1, i+2, y_{i+1}, x_{i+2}, k)$, where x_{i+2} is the value of $P_{i+2}^{-1}(y_{i+2})$ at τ_1. Then either D is table-undefined at τ_0 and table-defined with a table-defined right endpoint at τ_1, in which case we can apply the conclusion of the analysis for the case of a $(i+1, i+2)$-2chain, or D is table-defined at τ_0 and its right endpoint becomes table-defined during the query cycle, in which case we can apply Lemmas 4.15 and 4.16. 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

$$y_{i+2} \oplus x_{i+3} \oplus y_{i+3} \oplus x_{i-1} = 0. \tag{4.6}$$

  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 4.11, we have $x_{i+3} = p_{i+2}(x'_{i+2}) \oplus k'$ where $x'_{i+2} = r(C')$, $p_{i+2}(x'_{i+2}) \in \mathcal{P}$, and $k' \in \mathcal{H}$, and $y_{i+3} = p^{-1}_{i-1}(y'_{i-1}) \oplus k'$ where $y'_{i-1} = \ell(C')$ and $p^{-1}_{i-1}(y'_{i-1}) \in \mathcal{P}$.

- if x_{i-1} was table-undefined at τ_0, then it became table-defined because of a call to ReadTape$(i-1, -, y''_{i-1})$ and $x_{i-1} = p^{-1}_{i-1}(y''_{i-1}) \in \mathcal{P}$.

Finally, note that for the case where x_{i+3} was table-undefined at τ_0, we can assume that $x_{i+2} \neq x'_{i+2}$ 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 (4.6) with at least y_{i+2} or x_{i+3} table-undefined at τ_0 (and hence involving an element of $\mathcal{P}$), we always end up with an equation implying that $\mathcal{P}^{\oplus i} \cap \mathcal{H}^{\oplus j} \neq \emptyset$ for some $1 \leq i \leq 4$ and some $0 \leq j \leq 4$. (For example, if we assume y_{i+2}, x_{i+3}, y_{i+3}, and x_{i-1} were all table-undefined at τ_0, then Eq. (4.6) 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 $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ depending on whether $y'_{i-1} = y''_{i-1}$.)

- Case $i = 2$. Then we are considering a $(2,3)$-query cycle, $C = (4, 5, y_4, x_5, k)$ is a $(4,5)$-2chain, and its right endpoint at τ_1 is given by $x_{i+4} = x_1 = T^{-1}(k, y_5)$ where $y_5 = P_5(x_5)$. If the cipher query $(-, k, y_5)$ was table-undefined at τ_0, then it became table-defined during the query cycle, which implies that at τ_1, the 2chain $(5, 1, y_5, x_1, k)$ belongs to a complete path which was triggered during the query cycle, and C belongs to the same complete path. Assume now that the cipher query (k, x_1, y_5) was table-defined at τ_0, so that $k, x_1, y_5 \in \mathcal{H}$. If x_1 was table-undefined at τ_0, then it became table-defined because of a call ReadTape$(1, -, y_1)$ and hence $x_1 = p_1^{-1}(y_1) \in \mathcal{P}$, so that $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and BadPerm happens. Since C is table-undefined at τ_0, either y_4 or x_5 is table-undefined at τ_0. Assume first that x_5 was table-undefined at τ_0. Then it could only become table-defined because of a call to Assign$(5, x_5, y_5)$ resulting from a call to AdaptPath(C') where C' was triggered during the query cycle. By Lemma 4.11, 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 $x_{i+4} = x_2 = y_1 \oplus k$ where $y_1 = P_1(x_1)$. If the cipher query (k, x_1, y_5) was table-undefined at τ_0, then it became table-defined during the query cycle, which implies that at τ_1, C belongs to a complete path which was triggered during the query cycle. Assume now that the cipher query (k, x_1, y_5) was table-defined at τ_0, so that $k, x_1, y_5 \in \mathcal{H}$. Assume that y_5 was table-undefined at τ_0. Then y_5 became table-defined because of a call ReadTape$(5, +, x_5)$, so that $y_5 = p_5(x_5) \in \mathcal{P}$. Hence, $\mathcal{P} \cap \mathcal{H} \neq \emptyset$ and $\mathsf{BadPerm}$ 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 4.11, we have $x_1 \in \mathcal{C}$, so that $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens. Finally, assume that y_1 was table-defined and x_2 was table-undefined at τ_0 (so that $y_1 \in \mathcal{H}$). Then x_2 became table-defined because of a call ReadTape$(2, -, y_2)$ and hence $x_2 = p_2^{-1}(y_2) \in \mathcal{P}$. Since $x_2 = y_1 \oplus k$ where $k \in \mathcal{H}$, one has $\mathcal{P} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ and hence $\mathsf{BadPerm}$ occurs.

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

We collect Lemmas 4.15, 4.16, and 4.17 under a simpler to grasp form in the following lemma. To state it more concisely, we introduce the following definition.

Definition 16. We say a 2chain C *induces a 3chain* if C is table-defined and at least one of its endpoints is table-defined.

Lemma 4.18. *Consider a safe query cycle in a good execution of* G_2*. Let C be a 2chain which does not induce a 3chain at the beginning of the query cycle, but induces a 3chain at the end of the query cycle. Then, at the end of the query cycle, C belongs to a complete path which was triggered during the query cycle.*

Proof. Note that the opposite of "C induces a 3chain" is "C is table-defined and its two endpoints are dummy or table-undefined, or C is table-undefined". Hence, Lemmas 4.15, 4.16, and 4.17 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 4.19. *Consider a point τ_0 in a good execution of* G_2 *which is either the beginning of a query cycle, or the end of the execution, and assume that all query cycles until τ_0 were safe. Let C be a 2chain. Assume that at τ_0, C is table-defined and at least one of the two endpoints of C is table-defined. Then C is complete at τ_0.*

Proof. Let C be a 2chain which induces a 3chain at time τ_0. Recall that the opposite of "C induces a 3chain" is "C is table-defined and its two endpoints are dummy or table-undefined, or C is table-undefined". Consider the last assignment to a table before C induces a 3chain. This cannot have been an assignment to T/T^{-1} due to a cipher query made by the distinguisher; indeed, by Lemma 4.14, such an assignment cannot switch the state of a 2chain from table-undefined to table-defined, nor switch the endpoint of a table-defined 2chain from dummy to table-defined (and besides it is clear that it cannot switch an endpoint from non-dummy and table-undefined to table-defined). Hence this can only have happened during a query cycle that occurred before τ_0 (which was necessarily safe by the assumption that all query cycles before τ_0 were safe). But according to Lemma 4.18, at the end of this previous query cycle, C was complete, and this still holds at τ_0, hence the result. □

Lemma 4.20. *Consider an $(i, i+1)$-query cycle with initiating query (i_0, δ_0, z_0) in a good execution of G_2. Then, if an $(i, i+1)$-2chain C is triggered during the query cycle, there exists a sequence of $(i, i+1)$-2chains $(C_0, \ldots, C_t)$ triggered in this order such that $C_t = C$, and,* at the beginning of the query cycle, *either $z_0 = r(C_0) \neq \perp$, $\ell(C_1) = \ell(C_0) \neq \perp$, $r(C_2) = r(C_1) \neq \perp$, 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 $(C_0, \ldots, C_t)$ follows easily from inspection of the pseudocode, the only non-trivial point to prove is that all endpoints are already non-dummy at the beginning of the query cycle. But this follows easily from the fact that a $(4, 5)$-2chain with a right dummy endpoint cannot be triggered by a call to FindNewPaths$(1, +, \cdot)$, and that if a $(4, 5)$-2chain C with a right dummy endpoint is triggered by a call to FindNewPaths$(3, -, \cdot)$, then by Lemma 4.9 its right endpoint $r(C)$ is in $\mathcal{C}$ after getting non-dummy, and hence the call to FindNewPaths$(1, +, r(C))$ cannot trigger any 2chain unless $\mathcal{C} \cap \mathcal{H} \neq \emptyset$ and BadIC happens (and from the symmetric observations for a $(1, 2)$-2chain with a dummy left endpoint). □

Lemma 4.21. *Consider an $(i, i+1)$-query cycle in a good execution of G_2, and assume that all previous query cycles were safe. Then, if an $(i, i+1)$-2chain is complete at the beginning of the query cycle, it cannot be triggered during this query cycle.*

Proof. Let τ_0 denote the beginning of the query cycle. Assume towards a contradiction that there exists an $(i, i+1)$-2chain C which is complete at τ_0 and is triggered during the query cycle. Denote (i_0, δ_0, z_0) the initiating query of the query cycle. By Lemma 4.20, there exists a sequence $(C_0, \ldots, C_t)$ of triggered $(i, i+1)$-2chains such that $C_t = C$, and, at τ_0, either $r(C_0) = z_0$, $\ell(C_1) = \ell(C_0) \neq \perp$, $r(C_2) = r(C_1) \neq \perp$, 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. Hence, since C and C_{t-1} have a common endpoint, C_{t-1} is table-defined and has one of its endpoints non-dummy and table-defined at τ_0, which by Lemma 4.19 and the assumption that all previous query cycles were safe implies that C_{t-1} is complete at τ_0. By recursion, it follows that C_0 is complete at τ_0, which implies that the initiating query was table-defined at the beginning of the query cycle, a contradiction. □

Lemma 4.22. *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 4.5 (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 4.19, since all previous query cycles were safe, this implies that C was complete at the beginning of the query cycle, which by Lemma 4.21 (and again the fact that all previous query cycles were safe) implies that C cannot be triggered during the query cycle, a contradiction. □

The main result of this section now follows easily.

Lemma 4.23. *The simulator does not abort in a good execution of* G_2*.*

Proof. This is a direct consequence of Lemmas 4.13 and 4.22. □

4.4.3 Efficiency of the Simulator

Now that we have established that the simulator does not abort in good executions of G_2, we prove that it also runs in polynomial time. For this, we first prove some additional properties of good executions of G_2 below.

2chain-Cycles. In this section, we establish some additional properties of good executions in G_2 which would be used to prove the efficiency of the simulator in

Section 4.4.3. From Lemma 4.22, 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 4.23, 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 4.22 or 4.23).

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

Definition 17 (2cycle)**.** We say that 2 table-defined 2chains

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

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

1. $C_1 \neq C_2$;

2. both endpoints of 2chains C_1 and C_2 are non-dummy and table-undefined;

3. $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_1)$.

Definition 18 (4cycle)**.** We say that 4 table-defined 2chains

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

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

1. $C_1 \neq C_2$, $C_2 \neq C_3$, $C_3 \neq C_4$, $C_4 \neq C_1$;

2. both endpoints of all 2chains are non-dummy and table-undefined;

3. $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_3)$, $\ell(C_3) = \ell(C_4)$, $r(C_4) = r(C_1)$.

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

Lemma 4.24. *In a good execution of* G_2*, if* (k, x_1, y_5) *was table-defined through a cipher query made by the simulator in a query cycle, then the 2chain* $(5, 1, y_5, x_1, k)$ *is table-defined at the end of the query cycle.*

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

(1, 2)-QUERY CYCLE. By Lemma 4.6, a query (k, x_1, y_5) can get table-defined only through a call $\mathrm{Prev}(1, x_1, k)$ at line 28. Then, by inspection of the pseudocode $(5, -, y_5)$ is a pending query (where $\mathrm{Prev}(1, x_1, k)$ returned y_5 by assumption) and $(1, x_1, y_1)$ is table-defined. Since the simulator does not abort in a good execution of G_2 by Lemma 4.23, the pending query is table-defined at the end of the query cycle. Hence, the 2chain $(5, 1, y_5, x_1, k)$ is table-defined at the end of the query cycle.

(2, 3)-QUERY CYCLE. By Lemma 4.6, a query (k, x_1, y_5) can get table-defined only through a call $\mathrm{Prev}(1, x_1, k)$ at line 55. Since the simulator does not abort in a good execution of G_2 by Lemma 4.23, the call to $\mathrm{Assign}(5, \cdot, y_5)$ leads to $(5, \cdot, y_5)$ being table-defined. One can also verify from the pseudocode that $\mathrm{Prev}(1, x_1, k)$ is called at line 55 in a (2, 3)-query cycle only if $(1, x_1, y_1)$ is table-defined through $\mathrm{ReadTape}(1, -, y_1)$. Hence, the 2chain $(5, 1, y_5, x_1, k)$ is table-defined at the end of the query cycle.

The cases of a (3, 4) and (4, 5)-query cycles are analogous to those of (2, 3) and (1, 2)-query cycles respectively. In a (5, 1)-query cycle, by Lemma 4.6, no new cipher queries get table-defined. This completes the proof of the lemma. □

Lemma 4.25. *In a good execution of* G_2*, a 4cycle never appears at* $(4,5)$ *or at* $(1,2)$ *due to a cipher query by the distinguisher.*

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

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

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

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

Lemma 4.26. *If a 4cycle does not exist at* $(4, 5)$ *or at* $(1, 2)$ *at the beginning of a query cycle in a good execution of* G_2*, then a 4cycle does not exist at* $(4, 5)$ *or at* $(1, 2)$ *at the end of that query cycle.*

Proof. Assume towards a contradiction that at the end of a query cycle in a good execution of G_2 table-defined 2chains (C_1, C_2, C_3, C_4) form a 4cycle where $C_j = (4, 5, y_4^{(j)}, x_5^{(j)}, k^{(j)})$. Let $r(C_j) = T^{-1}(k^{(j)}, y_5^{(j)}) = x_1^{(j)} \neq \perp$ since the endpoints of the 2chains are not dummy by Definition 18. By the same definition, the endpoint queries of the 2chains are table-undefined.

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

Let us consider case (i) first. Here, we assume that the 4cycle formed immediately after a cipher query $(k^{(j)}, x_1^{(j)}, y_5^{(j)})$ that was table-defined in the query cycle for some $j \in \{1, \ldots, 4\}$. (Note that in a query cycle, cipher queries are made only by the simulator.) By Lemma 4.24, this means that the 2chain $(5, 1, x_1^{(j)}, y_5^{(j)}, k^{(j)})$ is table-defined. In particular, this implies that the right endpoint of C_j $r(C_j) = x_1^{(j)}$ is table-defined which contradicts our assumption.

Next we consider case (ii) i.e. the 4cycle formed immediately after a permutation query $(4, x_4^{(j)}, y_4^{(j)})$ or $(5, x_5^{(j)}, y_5^{(j)})$ was table-defined. Then, for $j = 1, 2, 3, 4$ the cipher queries $T(k^{(j)}, x_1^{(j)}) = y_5^{(j)}$ are table-defined at the beginning of the query cycle; this is because if they are table-defined during the query cycle, by Lemma 4.24, the right endpoint $r(C_j)$ gets table-defined contradicting our assumption that a 4cycle is formed in the query cycle. Hence, we have $k^{(j)} \in \mathcal{H}$, $x_1^{(j)} \in \mathcal{H}$, $y_5^{(j)} \in \mathcal{H}$ for $j \in \{1, \ldots, 4\}$.

$(1,2)$-QUERY CYCLE. Let the query cycle be a $(1,2)$-query cycle. From Table 4.1, we can see that in a $(1,2)$-query cycle, calls to $\mathrm{ReadTape}(5, -, \cdot)$ occur and adaptation occurs at position 4.

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

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

If $\ell(C') = y_5^{(j)}$, then at the end of the query cycle C_j will be such that its right endpoint $r(C_j) = x'_1$ which is table-defined. Hence, this will not form a 4cycle. If $\ell(C') \neq y_5^{(j)}$, let $\ell(C') = y'_5$. Then, Assign$(5, x'_5, y'_5)$ occurs in a call to ReadTape$(5, -, y'_5)$ leading to $x'_5 \in \mathcal{P}$. Now, we have

$$x'_5 \oplus k' = y_4^{(j)} = x_5^{(j)} \oplus k^{(j)}.$$

In the equation above, $k', k^{(j)} \in \mathcal{H}$, $x'_5 \in \mathcal{P}$. Also, $x_5^{(j)} \in \mathcal{P} \cup \mathcal{H}$ since $(5, x_5^{(j)}, y_5^{(j)})$ could either be table-defined as a left endpoint of a 2chain $\tilde{C} \neq C'$ in the current query cycle or could have been table-defined at the beginning of the query cycle itself. Then, either $\mathcal{H}^{\oplus 3} \cap \mathcal{P} \neq \emptyset$ or $\mathcal{H}^{\oplus 2} \cap \mathcal{P}^{\oplus 2} \neq \emptyset$ and BadPerm occurs.

$(2,3)$-QUERY CYCLE. Let the query cycle be a $(2,3)$-query cycle. From Table 4.1, we can see that in a $(2,3)$-query cycle, calls to ReadTape$(4, +, \cdot)$ occur and adaptation occurs at position 5.

Again, let us consider the case where all permutation queries $(4, x_4^{(j)}, y_4^{(j)})$ were table-defined at the beginning of the query cycle. Then, for the 4cycle to form at least one permutation query $(5, x_5^{(j)}, y_5^{(j)})$ should get table-defined. In a $(2,3)$-query cycle, this occurs in a call to Assign$(5, x_5^{(j)}, y_5^{(j)})$ called during AdaptPath(C') for some triggered chain C' in the query cycle. Since all query cycles in a good execution are safe by Lemma 4.22, using Lemma 4.11, we have $y_5^{(j)} \in \mathcal{C}$. However, since all cipher queries $(k^{(j)}, x_1^{(j)}, y_5^{(j)})$ are table-defined at the beginning of the query cycle, we have $y_5^{(j)} \in \mathcal{H}$. Hence, $\mathcal{H} \cap \mathcal{C} \neq \emptyset$ and BadIC occurs. So for a 4cycle to form in a $(2,3)$-query cycle, all permutation queries $(5, x_5^{(j)}, y_5^{(j)})$ need to be table-defined at the beginning of the query cycle.

Now, let us consider the case where at least one permutation query $(4, x_4^{(j)}, y_4^{(j)})$

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

$(3,4)$-QUERY CYCLE. Let the query cycle be a $(3,4)$-query cycle. From Table 4.1, we can see that in a $(3,4)$-query cycle, calls to ReadTape$(5,+,\cdot)$ occur and no new permutation query of the form $(4,\cdot,\cdot)$ gets table-defined. So for a 4cycle to form in a $(3,4)$-query cycle, all permutation queries $(4, x_4^{(j)}, y_4^{(j)})$ need to be table-defined at the beginning of the query cycle. Hence, $x_4^{(j)}, y_4^{(j)} \in \mathcal{H}$.

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

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

$(5,1)$-QUERY CYCLE. Let the query cycle be a $(5,1)$-query cycle. From Table 4.1, we can see that in a $(5,1)$-query cycle, calls to ReadTape$(4,-,\cdot)$ occur and no new permutation query of the form $(5,\cdot,\cdot)$ gets table-defined.

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

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

Next, we consider the case where $(4, x_4^{(2)}, y_4^{(2)})$ is table-defined during the query cycle. Then, $x_4^{(2)} \in \mathcal{P}$ since we are considering a $(5,1)$-query cycle. By definition of a 4cycle, $\ell(C_1) = \ell(C_2)$. This also means that $x_4^{(1)} \neq x_4^{(2)}$ as otherwise we have $x_5^{(1)} = x_5^{(2)}$ and $C_1 = C_2$ which violates the definition of a 4cycle (Definition 18). Then, $\ell(C_1) = \ell(C_2)$ implies that

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

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

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

□

Lemma 4.27. *At the end of a query cycle in a good execution of* G_2*, a 4cycle does not exist at* $(4,5)$ *or at* $(1,2)$*.*

Proof. This follows immediately from Lemmas 4.25 and 4.26. □

Corollary 4.28. *At the end of a query cycle in a good execution of* G_2*, there does not exist a 2cycle at* $(4,5)$ *or at* $(1,2)$*.*

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

Then there exists a 4cycle at $(4,5)$ consisting of chains C_j for $j = 1,\ldots,4$, where $C_j = C$ if $j = 1,3$ and $C_j = C'$ if $j = 2,4$. This is because we have $C_j \neq C_{j+1}$ (where $C_{j+1} = C_1$ when $j = 4$) for all j since $C \neq C'$. We know that both endpoints of C and C' are non-dummy and table-undefined. Finally, $\ell(C_j) = \ell(C_{j+1})$ if $j = 1,3$ since $\ell(C) = \ell(C')$ and $r(C_j) = r(C_{j+1})$ if $j = 2,4$ since $r(C') = r(C)$. This contradicts Lemma 4.27 which says that no 4cycle appears at $(4,5)$ in a good execution of G_2. □

Lemma 4.29. *In a good execution of* G_2*, for* $i \in \{2,4\}$*, there do not exist two distinct table-defined queries* (i,x_i,y_i) *and* (i,x'_i,y'_i) *such that* $x_i \oplus y_i = x'_i \oplus y'_i$*.*

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

For case (i), let us assume without loss of generality that $(2,x'_2,y'_2)$ was table-defined in an earlier query cycle. We consider the query cycle in which $(2, x_2, y_2)$ was table-defined. Then, $x'_2, y'_2 \in \mathcal{H}$. In the query cycle, the permutation query $(2, x_2, y_2)$ can get table-defined only in one of the following ways: (a) in a call to $\mathrm{ReadTape}(2, +, x_2)$ or $\mathrm{ReadTape}(2, -, y_2)$ (b) in a call to $\mathrm{Assign}(2, x_2, y_2)$ that occurs in $\mathrm{AdaptPath}(C)$ for a 2chain C triggered in the query cycle.

If $(2, x_2, y_2)$ was table-defined in a call to $\mathrm{ReadTape}(2, +, x_2)$, then we have $y_2 \in \mathcal{P}$ and $x_2 \in \mathcal{H}$. The former is due to the fact that in a good execution of G_2 all query cycles are safe by Lemma 4.22. The latter is because by inspection of the pseudocode, we can see that $\mathrm{ReadTape}(2, +, x_2)$ is called only when $(2, +, x_2)$ is a pending query and by Lemma 4.5(d), $(2, +, x_2)$ is pending only if it were the initiating query or the endpoint of a triggered 2chain. Note also that an endpoint at position 2 cannot be dummy by definition. Hence, by definition, history $\mathcal{H}$ contains all non-dummy endpoints and the initiating query. Thus, $x_2 \in \mathcal{H}$. By a similar analysis, we can show that when $(2, x_2, y_2)$ is table-defined in a call to $\mathrm{ReadTape}(2, -, y_2)$, we have $x_2 \in \mathcal{P}$ and $y_2 \in \mathcal{H}$. Thus, if $(2, x_2, y_2)$ was table-defined in a call to $\mathrm{ReadTape}(2, \cdot, \cdot)$ and $x_2 \oplus y_2 = x'_2 \oplus y'_2$, we have $\mathcal{H}^{\oplus 3} \cap \mathcal{P} \neq \emptyset$. This implies that $\mathsf{BadPerm}$ occurs which contradicts our assumption that this is a good execution of G_2.

Next, we consider the sub-case where $(2, x_2, y_2)$ is table-defined in a call to $\mathrm{Assign}(2, x_2, y_2)$ that occurs in $\mathrm{AdaptPath}(C)$ for a 2chain $C = (i, i+1, y_i, x_{i+1}, k)$ triggered in the query cycle. From Table 4.1, we know that 2 is an adapt position only for a $(4, 5)$-query cycle. Hence, from Lemma 4.11, we can deduce that $x_2 \oplus y_2 \in \mathcal{P}^{\oplus 2}$. This is because $x_2 \oplus y_2 = (p_1(r(C)) \oplus k) \oplus (p_3^{-1}(\ell(C)) \oplus k) \in \mathcal{P}^{\oplus 2}$. Then, if

$x_2 \oplus y_2 = x'_2 \oplus y'_2$, we have $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$. This implies that $\mathsf{BadPerm}$ occurs which contradicts our assumption that this is a good execution of G_2. This completes the analysis of case (i) where we assumed that the two distinct queries were table-defined in different query cycles.

In case (ii), the queries $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ are both table-defined in the same query cycle. As observed before, a permutation query gets table-defined either in a call to $\text{ReadPath}(2, \cdot, \cdot)$ or in a call to $\text{AdaptPath}(C)$ for a triggered 2chain C. Since both queries are at position 2 and are table-defined in the same query cycle, they are both table-defined through the same procedures i.e. either both through calls to ReadPath or both through calls to AdaptPath. Hence, we consider the following two sub-cases: (a) $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ are table-defined through calls to $\text{ReadPath}(2, +, x_2)$ and $\text{ReadPath}(2, +, x'_2)$ respectively or through calls to $\text{ReadPath}(2, -, y_2)$ and $\text{ReadPath}(2, -, y'_2)$ respectively. (b) $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ are table-defined through calls to $\text{Assign}(2, x_2, y_2)$ and $\text{Assign}(2, x'_2, y'_2)$ that occurred in calls to $\text{AdaptPath}(C)$ and $\text{AdaptPath}(C')$ respectively.

We analyse sub-case (a) first. As argued earlier, if $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ were table-defined through calls to $\text{ReadTape}(2, +, x_2)$ and $\text{ReadTape}(2, +, x'_2)$ respectively, then $x_2, x'_2 \in \mathcal{H}$ and $y_2, y'_2 \in \mathcal{P}$. Note that $y_2 \neq y'_2$ as the two queries are distinct and the permutation tables always encode a partial permutation. So $x_2 \oplus y_2 = x'_2 \oplus y'_2$ implies $\mathcal{H}^{\oplus 2} \cap \mathcal{P}^{\oplus 2} \neq \emptyset$. Hence $\mathsf{BadPerm}$ occurs which contradicts our assumption that this is a good execution of G_2. The analogous analysis works for the case where calls to ReadTape was of the form $\text{ReadTape}(2, -, y_2)$ and $\text{ReadTape}(2, -, y'_2)$.

Next, we analyse the sub-case where $(2, x_2, y_2)$ and $(2, x'_2, y'_2)$ were adapted in the same query cycle. Say they were adapted in AdaptPath(C) and AdaptPath(C') respectively where $C = (4, 5, y_4, x_5, k)$ and $C' = (4, 5, y'_4, x'_5, k')$ and $C \neq C'$ since the two queries are distinct. Again, by consulting Table 4.1, we can see that 2 is an adapt position only in a $(4, 5)$-query cycle. By Lemma 4.11, $x_2 = p_1(r(C)) \oplus k$ and $y_2 = p_3^{-1}(\ell(C)) \oplus k$ and similarly $x'_2 = p_1(r(C')) \oplus k'$ and $y'_2 = p_3^{-1}(\ell(C')) \oplus k'$. Since we are in a safe query cycle by Lemma 4.22, $p_1(r(C))$, $p_1(r(C'))$, $p_3^{-1}(\ell(C))$, $p_3^{-1}(\ell(C')) \in \mathcal{P}$. So, if $x_2 \oplus y_2 = x'_2 \oplus y'_2$, we have

$$p_1(r(C)) \oplus p_3^{-1}(\ell(C)) = p_1(r(C')) \oplus p_3^{-1}(\ell(C')). \tag{4.7}$$

We consider the following sub-cases here: (a) $r(C) \neq r(C')$, (b) $\ell(C) \neq \ell(C')$ and (c) $r(C) = r(C')$ and $\ell(C) = \ell(C')$. Consider case (a) first, if $r(C) \neq r(C')$, then in Equation (4.7), we have either $0 \in \mathcal{P}^{\oplus 4}$ or $0 \in \mathcal{P}^{\oplus 2}$ (where the latter occurs if $\ell(C) = \ell(C')$). Then, either $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ and in either case, BadPerm occurs. Considering case (b) next, if $\ell(C) \neq \ell(C')$, by a similar argument as just presented in case (a), we have either $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 0} \neq \emptyset$ and in either case, BadPerm occurs.

Finally, considering case (c), if $r(C) = r(C')$ and $\ell(C) = \ell(C')$, we distinguish between two cases here: (1) the cipher queries $T^{-1}(k, x_5)$ and $T^{-1}(k', x'_5)$ are table-defined at the beginning of the query cycle and (2) at least one of the cipher queries is not table-defined at the beginning of the query cycle. Considering case (1), if both cipher queries are table-defined at the beginning of the query cycle, then a 2cycle appears at (4, 5). This is because $C \neq C'$ by assumption that the two adapted queries $(2, x_2, y_2)$ and $(2, x_2', y_2')$ are distinct, the endpoints of the 2chains are non-dummy by assumption that the cipher queries are table-defined at the beginning of the query cycle, the endpoints are table-undefined since query cycles in a good execution of G_2 are safe by Lemma 4.22 and finally, $r(C) = r(C')$ and $\ell(C) = \ell(C')$ by assumption. This contradicts Corollary 4.28 which says that a 2cycle does not appear at (4, 5) in a good execution of G_2.

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

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

□

Lemma 4.30. *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*

$$\sum_{j=1}^{4}(x_i^{(j)} \oplus y_i^{(j)}) = 0. \tag{4.8}$$

Proof. Assume towards a contradiction that in a good execution of G_2, there exist four distinct queries $(i, x_i^{(j)}, y_i^{(j)})$ such that Eq. (4.8) holds. We analyse the case when $i = 2$. The proof for the case where $i = 4$ proceeds similarly. Let us consider the last assignment to the tables before the equation held. The assignment should have table-defined $(2, x_2^{(j)}, y_2^{(j)})$ for some $j \in \{1, \ldots, 4\}$. Without loss of generality, let us assume that $(2, x_2^{(1)}, y_2^{(1)})$ (i.e. $j = 1$) was the last query to be table-defined. Let us consider the various query cycles in which the query could have been table-defined.

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

(3, 4)-query cycle. By referring to Table 4.1, we can see that in a $(3, 4)$-query cycle, a query at position 2 gets table-defined only through calls to $\text{ReadTape}(2, -, \cdot)$. For queries $(2, x_2^{(j)}, y_2^{(j)})$ where $j = 2, \ldots, 4$, either $(2, x_2^{(j)}, y_2^{(j)})$ was already table-defined at the beginning of the query cycle or it gets table-defined in the same query cycle (but before $(2, x_2^{(1)}, y_2^{(1)})$ by assumption). Hence, we have $y_2^{(j)} \in \mathcal{H}$ for all $j = 1, 2, 3, 4$. This is because either $(2, x_2^{(j)}, y_2^{(j)})$ is table-defined at the beginning of the query cycle or $(2, -, y_2^{(j)})$ is a pending query and hence gets table-defined in the current query cycle. By definition of history $\mathcal{H}$, the initiating query, all (non-dummy) endpoints of table-defined 2chains and table-defined permutation queries exist in $\mathcal{H}$. Combining this fact with Lemma 4.5 (d), we have $y_2^{(j)} \in \mathcal{H}$ for all

$j = 1, 2, 3, 4$.

For the case of $x_2^{(j)}$ for $j = 1, \ldots, 4$, we can immediately deduce that $x_2^{(1)} \in \mathcal{P}$ by assumption that $(2, x_2^{(1)}, y_2^{(1)})$ is the last query to get table-defined (and by assumption that we are in a good execution of G_2 and all query cycles are safe by Lemma 4.22). For $x_2^{(j)}$ for $j \neq 1$, similar to the reasoning above, $(2, x_2^{(j)}, y_2^{(j)})$ was already table-defined at the beginning of the query cycle or it gets table-defined in the same query cycle (but before $(2, x_2^{(1)}, y_2^{(1)})$ by assumption). Then either $x_2^{(j)} \in \mathcal{H}$ if $(2, x_2^{(j)}, y_2^{(j)})$ was already table-defined at the beginning of the query cycle or $x_2^{(j)} \in \mathcal{P}$ if $(2, x_2^{(j)}, y_2^{(j)})$ gets table-defined in the current query cycle through $\text{ReadTape}(2, -, y_2^{(j)})$. (The reasoning for this is similar to prior reasoning for $x_2^{(1)}$.)

Then, if Eq. (4.8) holds, let t be the number of queries $(j, x_2^{(j)}, y_2^{(j)})$ that get table-defined in the current query cycle (where $1 \leq t \leq 4$). Recall that the queries $(j, x_2^{(j)}, y_2^{(j)})$ are distinct (and that $\mathcal{H}$ is a multiset). Then (a) if $t = 1$, we have $\mathcal{H}^{\oplus 7} \cap \mathcal{P} \neq \emptyset$, (b) if $t = 2$, we have $\mathcal{H}^{\oplus 6} \cap \mathcal{P}^{\oplus 2} \neq \emptyset$, (c) if $t = 3$, we have $\mathcal{H}^{\oplus 5} \cap \mathcal{P}^{\oplus 3} \neq \emptyset$ and (d) if $t = 4$, we have $\mathcal{H}^{\oplus 4} \cap \mathcal{P}^{\oplus 4} \neq \emptyset$. This implies that $\mathsf{BadPerm}$ occurs which contradicts our assumption that we are in a good execution of G_2.

$(4,5)$-QUERY CYCLE. Again, by referring to Table 4.1, we can see that in a $(4,5)$-query cycle, a query at position 2 gets table-defined only through calls to $\text{Assign}(2, \cdot, \cdot)$ that occur in a procedure $\text{AdaptPath}(C)$ for some triggered 2chain C. By assumption, $(2, x_2^{(1)}, y_2^{(1)})$ was (the last of the 4 queries to be) table-defined in this query cycle. Say this was due to call to $\text{Assign}(2, x_2^{(1)}, y_2^{(1)})$ that occurred in $\text{AdaptPath}(C_1)$ for a 2chain $C_1 = (4, 5, y_4^{(1)}, x_5^{(1)}, k^{(1)})$ triggered in the current query cycle. By Lemma 4.11, we have $x_2^{(1)} = p_1(r(C_1)) \oplus k^{(1)}$ and $y_2^{(1)} = p_3^{-1}(\ell(C_1)) \oplus k^{(1)}$. Here, we have $p_1(r(C_1), p_3^{-1}(\ell(C_1)) \in \mathcal{P}$ since all query cycles are safe in a good execution of G_2.

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

Let t be the number of queries $(j, x_2^{(j)}, y_2^{(j)})$ that get table-defined in the current query cycle (where $1 \leq t \leq 4$). If $t = 1$, then $(2, x_2^{(1)}, y_2^{(1)})$ is the only query to be table-defined in the query cycle. Hence, we have $x_2^{(1)} \oplus y_2^{(1)} = p_1(r(C_1) \oplus p_3^{-1}(\ell(C_1)) \in \mathcal{P}^{\oplus 2}$ and $\sum_{j=2}^{4} x_2^{(j)} \oplus y_2^{(j)} \in \mathcal{H}^{\oplus 6}$. Thus, if Eq. (4.8) holds, $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 6} \neq \emptyset$ and hence $\mathsf{BadPerm}$ occurs.

If $t = 2$, assume without loss of generality that $(2, x_2^{(j)}, y_2^{(j)})$ are the queries to get table-defined in the query cycle where $j = 1, 2$. Then, if Eq. (4.8) holds, we have

$$\sum_{j=1}^{2} x_2^{(j)} \oplus y_2^{(j)} = \sum_{j=3}^{4} x_2^{(j)} \oplus y_2^{(j)} \tag{4.9}$$

where the LHS is equal to $\sum_{j=1}^{2} p_1(r(C_j) \oplus p_3^{-1}(\ell(C_j))$ and RHS is in $\mathcal{H}^{\oplus 4}$ by assumption. Since $p_1(r(C_j), p_3^{-1}(\ell(C_j)) \in \mathcal{P}$ for $j = 1, 2$, $\mathsf{BadPerm}$ occurs unless $r(C_1) = r(C_2)$ and $\ell(C_1) = \ell(C_2)$. If $r(C_1) = r(C_2)$ and $\ell(C_1) = \ell(C_2)$, then the

LHS of Eq. (4.9) is 0. However, by Lemma 4.29, in a good execution of G_2, we cannot have two distinct queries $(2, x_2^{(1)}, y_2^{(1)})$ and $(2, x_2^{(2)}, y_2^{(2)})$ such that $\sum_{j=1}^{2} x_2^{(j)} \oplus y_2^{(j)} = 0$. Hence, if Eq. (4.9) holds, $\mathsf{BadPerm}$ occurs since we have either $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 4} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 4} \neq \emptyset$.

If $t = 3$, we assume without loss of generality that $(2, x_2^{(j)}, y_2^{(j)})$ are the queries to get table-defined in the query cycle where $j = 1, 2, 3$. Then, if Eq. (4.8) holds, we have

$$\sum_{j=1}^{3} x_2^{(j)} \oplus y_2^{(j)} = x_2^{(4)} \oplus y_2^{(4)} \tag{4.10}$$

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

$$\sum_{j=1}^{3} x_2^{(j)} \oplus y_2^{(j)} = \sum_{j=1}^{3} p_1(r(C_j) \oplus p_3^{-1}(\ell(C_j)). \tag{4.11}$$

Since $p_1(r(C_j), p_3^{-1}(\ell(C_j)) \in \mathcal{P}$ for $j = 1, 2, 3$, $\mathsf{BadPerm}$ occurs (since we have either $\mathcal{P}^{\oplus 6} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ or $\mathcal{P}^{\oplus 4} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$ or $\mathcal{P}^{\oplus 2} \cap \mathcal{H}^{\oplus 2} \neq \emptyset$). This is because even if some of the right (resp. left) endpoints of the triggered 2chains C_j are equal, the terms $p_1(r(C_j))$ (resp. $p_3^{-1}(\ell(C_j)))$ can cancel each other out in the sum on the RHS of Eq. (4.11) only if there are even number of right (resp. left) endpoints.

If $t = 4$, all queries $(2, x_2^{(j)}, y_2^{(j)})$ for $j = 1, \ldots, 4$ are table-defined in the current query cycle. Then, if Eq. (4.8) holds, we have

$$\sum_{j=1}^{4} x_2^{(j)} \oplus y_2^{(j)} = \sum_{j=1}^{4} p_1(r(C_j) \oplus p_3^{-1}(\ell(C_j)) = 0 \tag{4.12}$$

where $p_1(r(C_j), p_3^{-1}(\ell(C_j)) \in \mathcal{P}$ for $j = 1, 2, 3, 4$. Then, $\mathsf{BadPerm}$ occurs (since $\mathcal{H}^{\oplus 0} = 0$) unless the $p_1(\cdot)$ and $p_3^{-1}(\cdot)$ terms in Eq. (4.12) cancel with each other. For the terms in the sum in Eq. (4.12) to cancel out, we need 4 pairs (r_1, r_2), (r_3, r_4), (ℓ_1, ℓ_2) and (ℓ_3, ℓ_4) such that $r(C_{r_1}) = r(C_{r_2})$, $r(C_{r_3}) = r(C_{r_4})$, $\ell(C_{\ell_1}) = \ell(C_{\ell_2})$ and $\ell(C_{\ell_3}) = \ell(C_{\ell_4})$ where $r_i, \ell_i \in \{1, 2, 3, 4\}$ and $r_i \neq r_{i'}$ for $i \neq i'$, $\ell_i \neq \ell_{i'}$ for $i \neq i'$.

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

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

evaluates to 0. However, this implies that in a good execution of G_2 there exist two distinct queries $(2, x_2^{(r_i)}, y_2^{(r_i)})$ and $(2, x_2^{(r_{i'})}, y_2^{(r_{i'})})$ such that $x_2^{(r_i)} \oplus y_2^{(r_i)} \oplus x_2^{(r_{i'})} \oplus y_2^{(r_{i'})} = 0$ which contradicts Lemma 4.29.

Then, without loss of generality, we can assume that the 4 pairs (r_1, r_2), (r_3, r_4), (ℓ_1, ℓ_2) and (ℓ_3, ℓ_4) are respectively $(2, 3)$, $(4, 1)$, $(1, 2)$ and $(3, 4)$ i.e. $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_3)$, $\ell(C_3) = \ell(C_4)$ and $r(C_4) = r(C_1)$. Recall that C_j are the triggered 2chains such that $(2, x_2^{(j)}, y_2^{(j)})$ was table-defined in Assign$(2, x_2^{(j)}, y_2^{(j)})$ made in the procedure AdaptPath(C_j). To analyse when Eq. (4.12) can hold, we distinguish between three cases: (i) all cipher queries $T^{-1}(k^{(j)}, y_5^{(j)})$ are table-defined at the beginning of the query cycle, (ii) exactly one cipher query (say) $T^{-1}(k^{(1)}, y_5^{(1)})$ is table-defined during the query cycle and (iii) more than one cipher query $T^{-1}(k^{(j)}, y_5^{(j)})$ gets table-defined in the current query cycle.

In case (i), if all cipher queries $T^{-1}(k^{(j)}, y_5^{(j)})$ are table-defined at the beginning of the query cycle, then we observe that there exists a 4cycle at $(4, 5)$. This is because (a) all four triggered 2chains C_j are distinct by assumption that the queries $(2, x_2^{(j)}, y_2^{(j)})$ are distinct, (b) the endpoints of all triggered 2chains C_j are non-dummy and table-undefined since the cipher queries are table-defined at the beginning of the query cycle and since all the query cycles are safe in a good execution by Lemma 4.22 and (c) we have $\ell(C_1) = \ell(C_2)$, $r(C_2) = r(C_3)$, $\ell(C_3) = \ell(C_4)$ and $r(C_4) = r(C_1)$ since (r_1, r_2), (r_3, r_4), (ℓ_1, ℓ_2) and (ℓ_3, ℓ_4) are respectively $(2, 3)$, $(4, 1)$, $(1, 2)$ and $(3, 4)$. However, by Lemma 4.27, in a good execution of G_2, there does not appear a 4cycle at $(4, 5)$. Hence, this contradicts our assumption that we are in a good execution of G_2.

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

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

(5, 1)-QUERY CYCLE. By referring to Table 4.1, we can see that in a $(5, 1)$-query cycle, a query at position 2 gets table-defined only through calls to $\mathrm{ReadTape}(2, +, \cdot)$. For queries $(2, x_2^{(j)}, y_2^{(j)})$ where $j = 2, \ldots, 4$, either $(2, x_2^{(j)}, y_2^{(j)})$ was already table-defined at the beginning of the query cycle or it gets table-defined in the same query cycle. Hence, we have $x_2^{(j)} \in \mathcal{H}$ for all $j = 1, 2, 3, 4$. This is because either $(2, x_2^{(j)}, y_2^{(j)})$ is table-defined at the beginning of the query cycle or $(2, +, x_2^{(j)})$ is a pending query and hence gets table-defined in the current query cycle. By definition of history $\mathcal{H}$, the initiating query, all (non-dummy) endpoints of table-defined 2chains and table-defined permutation queries exist in $\mathcal{H}$. Combining this fact with Lemma 4.5 (d), we have $x_2^{(j)} \in \mathcal{H}$ for all $j = 1, 2, 3, 4$. The rest of the analysis for the case of a $(5, 1)$-query cycle proceeds analogous to that of a $(3, 4)$-query cycle.

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

The core of the termination argument is presented below.

Termination Proof

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

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

Definition 19. During a query cycle, we say that a 2chain C is *triggered by query* (i, δ, z) if it is added to $\mathsf{Triggered}$ during a call to FindNewPaths(i, δ, z). We say C is an (i, δ)-triggered 2chain if it is triggered by a query of type (i, δ).

By Lemma 4.5 (b), a triggered $(i, i+1)$-2chain is either $(i-1, -)$- or $(i+2, +)$-triggered. For brevity, we group four special types of triggered 2chains under a common name.

Definition 20. 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 21. 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 29 and 47.

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

Lemma 4.31. *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 4.5 (e), at the end of the query cycle where C is triggered, C belongs to a complete path which must be Π by Lemma 4.4. By Lemma 4.21, this implies that neither C nor any other 2chain belonging to Π will ever be triggered in any subsequent query cycle. □

Lemma 4.32. *For* $i \in \{1, \ldots, 5\}$*, the number of table-defined permutation queries* (i, x_i, y_i) *during an execution of* G_2 *can never exceed the sum of*

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

Proof. Entries are added to P_i/P_i^{-1} either by a call to ReadTape during an $(i+1, i+2)$- or an $(i-2, i-1)$-query cycle or by a call to AdaptPath during an $(i+2, i+3)$-query cycle (see Table 4.1).

Consider first entries that were added by a call to ReadTape during an $(i+1, i+2)$- or an $(i-2, i-1)$-query cycle. Clearly, the number of such table-defined queries cannot exceed the sum of the total number $N_{i,+}$ of queries of type $(i, +)$ that became pending during an $(i-2, i-1)$-query cycle and the total number $N_{i,-}$ of queries of type $(i, -)$ that became pending during an $(i+1, i+2)$-query cycle. Now, $N_{i,+}$ cannot exceed the sum of the total number of initiating and non-initiating pending queries of type $(i, +)$ over all $(i-2, i-1)$-query cycles. The total number of initiating queries of type $(i, +)$ is clearly at most the number of distinguisher's calls to Query$(i, +, \cdot)$, while the total number of non-initiating pending queries of type $(i, +)$ over all $(i-2, i-1)$-query cycles cannot exceed the total number of $(i-3, -)$-triggered 2chains (since clearly a non-initiating pending query of type $(i, +)$ cannot be associated with an $(i, +)$-triggered $(i-2, i-1)$-2chain). Similarly, $N_{i,-}$ cannot exceed the sum of the total number of distinguisher's call to 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. Clearly, the total number of such triples cannot exceed the total number of $(i+2, i+3)$-2chains that are triggered over all $(i+2, i+3)$-query cycles (irrespectively of whether they are $(i+1, -)$- or $(i+4, +)$-triggered). The result follows. □

The following lemma contains the standard "bootstrapping" argument introduced in [18] that has been used to prove termination of all simulators for the Feistel and the IEM constructions since then.

Lemma 4.33. *In a good execution of* G_2*, at most* q *wrapping 2chains get triggered in total.*

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

By inspection of the pseudocode, we see that for each triggered wrapping 2chain C, there is a triple (k, x_1, y_5) such that 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 4.21.

Hence, each triggered wrapping 2chain can be mapped to a distinct table-defined cipher query (k, x_1, y_5). This cipher query became table-defined because of call to Enc/Dec made either by the distinguisher or the simulator. It remains to show that the corresponding call cannot have been made by the simulator. But this follows easily from Lemma 4.6 since when the simulator makes a cipher query which modifies T/T^{-1}, the corresponding 2chain has already been triggered. Hence it will be complete at the end of the query cycle and cannot be triggered again by Lemma 4.31. □

Lemma 4.34. *In a good execution of* G_2*, one always has* $|P_3| \leq 2q$*.*

Proof. By Lemma 4.32, the number of table-defined permutation queries $(3, x_3, y_3)$ (and hence the size of P_3) cannot exceed the sum of

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

The number of entries of the first type is at most q by the assumption that the distinguisher makes at most q oracle queries to each permutation. On the other hand, note that any 2chain mentioned for the three other types are wrapping 2chains.

Hence, by Lemma 4.33, there are at most q such entries in total, so that $|P_3| \leq 2q$. □

Lemma 4.35. *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 4.5 (e), C belongs to a complete path $((1,x_1,y_1),\ldots,(5,x_5,y_5))$ at the end of the execution (since the simulator does not abort), and $C = (3,4,y_3,x_4,k)$ if it was $(5,+)$-triggered, whereas $C = (4,5,y_4,x_5,k)$ if it was $(3,-)$-triggered.

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

We now distinguish two cases depending on how $(5,+,x_5)$ became table-defined or pending. Assume first that this was because of a distinguisher's call to Query$(5,\cdot,\cdot)$. There are at most q such calls, hence there are at most q possibilities for x_5. There are at most $2q$ possibilities for y_3 by Lemma 4.34. 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 4.29 (note that one must have $x_4 \oplus y_4 = y_3 \oplus x_5$). Hence there are at most $2q^2$ possibilities in that case.

Assume now that this $(5, +, x_5)$ was a non-initiating pending query in the same query cycle where C was triggered, or became table-defined during a previous query cycle than the one where C was triggered and for which $(5, +, x_5)$ was neither the initiating query nor became table-defined during the ReadTape call for the initiating query. In all cases there exists a table-defined $(3, 4)$-2chain $C' = (3, 4, y'_3, x'_4, k')$ distinct from $(3, 4, y_3, x_4, k)$ such that $x_5 = r(C') = y'_4 \oplus x'_4 \oplus y'_3$. Since we also have $x_5 = y_4 \oplus x_4 \oplus y_3$, we obtain $x_4 \oplus y_4 \oplus x'_4 \oplus y'_4 = y_3 \oplus y'_3$. If $y_3 = y'_3$, by Lemma 4.29 we have $x_4 = x'_4$ and $C' = (3, 4, y_3, x_4, k)$, which cannot be. On the other hand, for a fixed (orderless) pair of $y_3 \neq y'_3$, the (orderless) pair of $(4, x_4, y_4)$ and $(4, x'_4, y'_4)$ is unique by Lemmas 4.29 and 4.30 (otherwise, one of the lemmas must be violated by the two pairs). There are at most $\binom{2q}{2} = q(2q-1)$ choices of y_3 and y'_3; for each pair there is at most one (orderless) pair of $(4, x_4, y_4)$ and $(4, x'_4, y'_4)$, so there are 2 ways to combine the queries to form two 2chains. Moreover, C' must either have been completed during a previous query cycle than the one where C is triggered, or must have been triggered *before* C in the same query cycle and have made x_5 pending (in which case C was triggered by $(5, +, x_5)$). Thus each way to combine y_3, y'_3, $(4, x_4, y_4)$ and $(4, x'_4, y'_4)$ to form two 2chains corresponds to at most one $(3, +)$- or $(5, -)$-triggered 2chain, so at most $4q^2 - 2q$ such 2chains are triggered this way.

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

Lemma 4.36. *In a good execution of* G_2, *one always has* $|P_2| \leq 6q^2$ *and* $|P_4| \leq 6q^2$.

Proof. By Lemma 4.32, 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 $\text{Query}(2, \cdot, \cdot)$,
- the number of $(3, 4)$-2chains that were $(5, +)$-triggered,
- the number of $(5, 1)$-2chains that were $(4, -)$-triggered,
- the number of $(4, 5)$-2chains that were either $(3, -)$- or $(1, +)$-triggered.

There are at most q entries of the first type by the assumption that the distinguisher makes at most q oracle queries. Note that any 2chain mentioned for the other cases are either wrapping, $(3, -)$-triggered, or $(5, +)$-triggered 2chains. Hence, by Lemmas 4.33 and 4.35, there are at most $q + 6q^2 - 2q$ entries of the three other types in total. Thus, we have $|P_2| \le q + q + 6q^2 - 2q = 6q^2$. Symmetrically, $|P_4| \le 6q^2$. □

Lemma 4.37. *In a good execution of* G_2*, at most* $12q^3$ *2chains are triggered in total.*

Proof. Since the simulator does not abort in good executions by Lemma 4.23, any triggered 2chain belongs to a complete path at the end of the execution. Moreover, by Lemma 4.31, 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 4.4. 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 4.34 and 4.36 is at most $2q \cdot 6q^2 = 12q^3$. □

Lemma 4.38. *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 4.39. *In a good execution of* G_2*, one always has* $|P_1| \leq 12q^3 + q$ *and* $|P_5| \leq 12q^3 + q$*.*

Proof. By Lemma 4.32, 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 4.37. Therefore, the size of $|P_1|$ is at most $12q^3 + q$. The same reasoning applies to $|P_5|$. □

Lemma 4.40. *In good executions of* G_2*, the simulator runs in time* $O(q^8)$ *and uses* $O(q^3)$ *space.*

Proof. By Lemmas 4.34, 4.36 and 4.39, the total number of queries that are defined during an execution is $O(q^3)$. In a non-aborting execution, every time Assign is called an undefined query becomes defined. Therefore, Assign is called $O(q^3)$ times, which implies ReadTape and AdaptPath are called $O(q^3)$ times. The aforementioned procedures run in constant time, so the total running time of them is $O(q^3)$.

The procedure FindNewPaths is called once for each pending query. In a non-aborting execution, pending queries become distinct defined queries at the end of the execution, which implies the total number of pending queries in position i is upper bounded by $|P_i|$ at the end of the proof. In a call to 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 $|P_{i+1}| \cdot |P_{i+2}|$[13]. 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 4.34, 4.36 and 4.39.

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

Theorem 4.41. *An execution of* G_2 *is good with probability at least* $1 - 10^{22}q^{38}/2^n$.

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

By Lemmas 4.34, 4.36, 4.39 and 4.38 and by Definition 5, the number of table-defined $(5, 1)$-2chains is upper bounded by $12q^2 + q \leq 13q^3$ (since each cipher query in T uniquely determines a $(5, 1)$-2chain), and the total number of table-defined $(i, i+1)$-2chains for $i = 1, 2, 3, 4$ is at most $2((12q^3 + q) \cdot 6q^2 + 6q^2 \cdot 2q) \leq 180q^5$ (since each pair of table-defined queries uniquely determine an $(i, i + 1)$-2chain). There are at most $40q^3$ table-defined permutation queries and $13q^3$ table-defined cipher

[13] 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 4.34, 4.36 and 4.39.

queries. By Definition 9, the size of $\mathcal{H}$ can be upper bounded by

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

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

- (cf. Definition 12) an entry read from p_i/p_i^{-1} is in $\mathcal{P}^{\oplus j} \oplus \mathcal{H}^{\oplus \ell}$ for some $j, \ell \geq 0$ such that $j + \ell \leq 7$, where $\mathcal{P}$ refers to the random values that *has already been sampled* from permutation tapes in the current query cycle;
- (cf. Definition 13) an entry read from $\mathsf{ic}/\mathsf{ic}^{-1}$ is in $\mathcal{H}$ or in $\mathcal{C}$, where $\mathcal{C}$ refers to the random values that *has already been sampled* from the cipher tape in the current query cycle.

We note that the bounds in Lemmas 4.34, 4.36, 4.39 and 4.38 hold as long as no bad event has occurred in the execution (indeed, the only property of good executions used in the proof is that "the bad events never occur"). Therefore, unless a bad event occurs, the execution terminates before the numbers of entries read from p_i/p_i^{-1} (resp. $\mathsf{ic}/\mathsf{ic}^{-1}$) exceeds $40q^3$ (resp. $13q^3$), and the bounds on $\mathcal{H}$, $\mathcal{P}$ and $\mathcal{C}$ for good executions also hold.

Assume no bad event has occurred in the execution, and consider a call to ReadTape in which $p_i(x_i)$ is read (the case of p_i^{-1} is symmetric). At this point, less than $13q^3$ entries of p_i/p_i^{-1} has been read, so the value of $p_i(x_i)$ is uniformly distributed among at least $2^n - 13q^3 \geq 2^{n-1}$ values, where the inequality is due to the assumption that $13q^3 \leq 2^{n-1}$. The total size of the sets $\mathcal{P}^{\oplus j} \oplus \mathcal{H}^{\oplus \ell}$ for $j + \ell \leq 7$ is at most $(700q^5)^7$,[14] so the probability that $\mathsf{BadPerm}$ occurs is $(700q^5)^7/2^{n-1}$.

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

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

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

4.4.4 Indistinguishability of the First and Second Experiments

Recall that the only difference between G_1 and G_2 is in procedure Check (line 68): 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 4.42. *For randomly chosen tapes $\mathcal{T} = (p_1, \ldots, p_5, \mathsf{ic})$, the probability that the execution of G_2 with $\mathcal{T}$ is good and that the executions of G_1 and G_2 with the same tapes $\mathcal{T}$ are identical is at least $1 - (10^{22}q^{38}/2^n + 2704q^8/2^n)$.*

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

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

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 4.38, 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. 4.1 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 4.36 and 4.39, there are

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

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

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

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

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

Lemma 4.43. *We have*

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

Proof. Since $\mathcal{D}$ outputs the same value in identical executions of G_1 and G_2, this is a direct corollary of Lemma 4.42. □

Theorem 4.44. *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 4.40, 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 4.42, the simulator runs in time $O(q^8)$ with high probability in G_1.

Using a similar trick as [23], 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.[15] 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 4.40, 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 $\text{Enc}(y_1 \oplus x_2, x_1) = y_5$.

[15] 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.

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 4.36 and 4.39, 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 4.37), so handling the triggered 2chains uses $O(q^3)$ running time. There are $O(q^2)$ calls at positions 2, 3 or 4, and each call uses $O(q^3)$ time to traverse a table.

Recall in the proof of Lemma 4.40 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 4.42) and since the simulator knows the value of q (cf. Definition 17), we can let the simulator abort when the running time or the number of queries exceeds the corresponding bound. Then the simulator is efficient with probability 1 and the change affects an execution only with negligible probability. □

4.4.5 Indistinguishability of the Second and Fourth Experiments

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 [23].

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

Definition 22. 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 $\text{Dec}(k, y'_5) = x_1$.[16]

Note that for an arbitrary distinguisher $\mathcal{D}$, it is easy to construct an equivalent distinguisher $\mathcal{D}'$ that completes all paths and that always outputs the same value as $\mathcal{D}$: Let $\mathcal{D}'$ run $\mathcal{D}$ until $\mathcal{D}$ outputs a value b. For each cipher query that has been made by $\mathcal{D}$, $\mathcal{D}'$ issues the permutation queries $\text{Query}(i, +, x_i)$ to "complete the path" as in Definition 22. Finally $\mathcal{D}'$ outputs b, ignoring the answers of the extra

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

queries. The distinguisher $\mathcal{D}'$ issues at most q extra queries in each position.

In the rest of this section (except for Lemma 4.53), we will assume without loss of generality that the distinguisher $\mathcal{D}$ completes all paths (the assumption is without loss of generality with the cost of a multiplicative factor in the final bound). The definitions and lemmas that only hold under this assumption will be marked with (*).

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

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

Definition 23. A *partial random permutation tape* is a pair of tables $\tilde{p}, \tilde{p}^{-1} : \{0,1\}^n \to \{0,1\}^n \cup \{\perp\}$, such that $\tilde{p}^{-1}(\tilde{p}(x)) = x$ for all x satisfying $\tilde{p}(x) \neq \perp$, and such that $\tilde{p}(\tilde{p}^{-1}(y)) = y$ for all y satisfying $\tilde{p}^{-1}(y) \neq \perp$.

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

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

Definition 24. We say a random permutation tape p is *compatible* with a partial random permutation tape $\tilde{p}$ if $p(x) = \tilde{p}(x)$ for all x such that $\tilde{p}(x) \neq \perp$. Similarly, an ideal cipher tape ic is *compatible* with a partial ideal cipher tape pic if $\mathsf{ic}(k, u) = \mathsf{pic}(k, u)$ for all k, u such that $\mathsf{pic}(k, u) \neq \perp$.

Definition 25. Consider an execution of G_2 with random tapes $p_1, p_2, \ldots, p_5, \mathsf{ic}$. The *footprint* of the execution is the set of partial random tapes $\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \mathsf{pic}$ consisting of entries of the corresponding tapes that are accessed[17] 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 actually obtainable in some execution of $\mathcal{D}$. We let FP_2 and FP_3 denote the set of obtainable footprints in G_2 and G_3 respectively.

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

Lemma 4.45. *For $i = 2, 3$ and for an obtainable footprint $\omega \in \mathsf{FP}_i$, an execution of G_i has footprint ω if and only if the random tapes are compatible with ω.*

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

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

need to prove the "if" direction.

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

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

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

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

For an obtainable footprint $\omega \in \mathsf{FP}_i$, $\Pr_{\mathsf{G}_i}[\omega]$ equals the probability that the random tapes are compatible with ω by Lemma 4.45. Let $|\tilde{p}_i|$ (resp. $|\mathsf{pic}|$) denote the number of non-$\perp$ entries in $\tilde{p}_i$ (resp. pic), and let $|\mathsf{pic}(k)|$ denote the number of non-$\perp$ entries of the form $(k,*)$ in pic. For $\omega = (\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \mathsf{pic}) \in \mathsf{FP}_2$, we have

$$\Pr_{\mathsf{G}_2}[\omega] = \Big(\prod_{i=1}^{5}\prod_{\ell=0}^{|\tilde{p}_i|-1}\frac{1}{2^n-\ell}\Big)\Big(\prod_{k\in\{0,1\}^n}\prod_{\ell=0}^{|\mathsf{pic}(k)|-1}\frac{1}{2^n-\ell}\Big), \tag{4.13}$$

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

$$\Pr_{\mathsf{G}_3}[\omega] = \prod_{i=1}^{5}\prod_{\ell=0}^{|\tilde{t}_i|-1}\frac{1}{2^n-\ell}. \tag{4.14}$$

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

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

Definition 26. (*) We define the injection $\zeta : \mathsf{FP}_2^* \to \mathsf{FP}_3^*$ as follows: for $\omega = (\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_5, \mathsf{pic}) \in \mathsf{FP}_2^*$, let $\zeta(\omega) = (\tilde{t}_1, \tilde{t}_2, \ldots, \tilde{t}_5)$ where

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

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

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

Lemma 4.46. (*) *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 4.5 (e), since good executions don't abort, a triggered 2chain must belong to a complete path. Therefore, we only need to prove that for each $T(k, x_1) = y_5$, $(5, 1, y_5, x_1, k)$ is equivalent to a triggered 2chain.

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

If $\text{Enc}(k, x_1)$ is called by the distinguisher: since the distinguisher completes all paths (cf. Definition 22), 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 4.18 and since query cycles in good executions must be safe (cf. Lemma 4.22), C belongs to a complete path which was triggered during the query cycle. The lemma follows by the fact that C is equivalent to $(5, 1, y_5, x_1, k)$.

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

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

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

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

Lemma 4.47. (*) *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 $\omega = (\tilde{p}_1, \dots, \tilde{p}_5, \mathsf{pic})$ and $\zeta(\omega) = (\tilde{t}_1, \dots, \tilde{t}_5)$. Let $\mathcal{T} = (t_1, \dots, t_5)$ 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 4.46, the 2chain $(5, 1, y_5, x_1, k)$ is complete, i.e., P contains queries (i, x_i, y_i) for $i = 1, 2, 3, 4, 5$ satisfying (4.3). 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 $\mathsf{EM}(k, x_1)$ or $\mathsf{EM}^{-1}(k, y_5)$, so in G_3 we also have $T(k, x_1) = y_5$ and the two executions don't diverge on cipher queries. Hence, the two executions never diverge and are identical.

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

If $P_i(x_i) = y_i$ is assigned in ReadTape, then the simulator should have read $t_i(x_i)$ or $t_i^{-1}(y_i)$ in the same procedure call. If $P_i(x_i) = y_i$ is assigned in AdaptPath, then there exist defined queries (j, x_j, y_j) for $j \in \{1, 2, 3, 4, 5\} \setminus \{i\}$ and a cipher query $T(k, x_1) = y_5$ such that $x_{j+1} = y_j \oplus k$ for $j = 1, 2, 3, 4$. When $T(k, x_1) = y_5$ is assigned, the entries $t_j(x_j) = y_j$ for $j = 1, 2, 3, 4, 5$ are read in the call to $\mathsf{EM}(k, x_1)$ or $\mathsf{EM}^{-1}(k, y_5)$, which includes the entry $t_i(x_i) = y_i$. In both cases, $t(x_i) = y_i$ is read and the footprint should contain $\tilde{t}(x_i) = y_i$. □

Lemma 4.48. (*) *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 4.23), Lemma 4.47 implies that the execution of G_3 with footprint $\zeta(\omega)$ is non-aborting and hence $\zeta(\omega) \in \mathsf{FP}_3^*$.

Lemma 4.47 also implies that the ζ is injective: For a fixed $\zeta(\omega)$, the G_2-execution with footprint ω is identical to the G_3-execution with footprint $\zeta(\omega)$, which can be recovered from $\zeta(\omega)$. Let $\omega = (\tilde{p}_1, \ldots, \tilde{p}_5, \mathsf{pic})$, and it can be uniquely reconstructed as follows: the partial permutation tapes $\tilde{p}_i$ contain entries that are read by the simulator during the execution; the partial tape pic contains cipher queries that are issued by the distinguisher or the simulator. □

Lemma 4.49. (*) *For* $\omega = (\tilde{p}_1, \ldots, \tilde{p}_5, \mathsf{pic}) \in \mathsf{FP}_2^*$ *and* $\zeta(\omega) = (\tilde{t}_1, \ldots, \tilde{t}_5)$*, we have*

$$\sum_{i=1}^{5} |\tilde{t}_i| = \sum_{i=1}^{5} |\tilde{p}_i| + |\mathsf{pic}|. \tag{4.15}$$

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 26). On the other hand, $\tilde{p}_i$ contains an entry $\tilde{p}_i(x_i) = y_i$ if and only if $P_i(x_i) = y_i$ is assigned in a call to ReadTape. Thus the difference between the sizes of $\tilde{t}_i$ and $\tilde{p}_i$ equals the number of adapted queries assigned in AdaptPath. From the pseudocode, we observe that AdaptPath is called for each triggered 2chain, in which exactly one query is assigned.

We only need to prove the number of triggered 2chains equals the size of pic. Note that pic contains the same queries as the table T at the end of the execution. By Lemma 4.46, 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 4.31 the triggered 2chains are not equivalent. Thus the triggered 2chains belong to distinct complete paths, each containing a distinct table-defined $(5,1)$-2chain that corresponds to a distinct query in T. Thus there is a one-one correspondence between triggered 2chains and entries in T, implying $|T| = |\mathsf{pic}|$ equals the number of triggered 2chains. □

Lemma 4.50. (*) *For $\omega \in \mathsf{FP}_2^*$, we have*

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

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

Let $\omega = (\tilde{p}_1, \ldots, \tilde{p}_5, \mathsf{pic})$ and $\zeta(\omega) = (\tilde{t}_1, \ldots, \tilde{t}_5)$. By Lemma 4.38, we have $|T| \leq 12q^3 + q \leq 13q^3$. Since pic contains the same entries as T at the end of the execution and $\mathsf{pic}(k)$ is a subset of pic, for any key $k \in \{0,1\}^n$ we have

$$|\mathsf{pic}(k)| \leq |\mathsf{pic}| \leq 13q^3. \tag{4.16}$$

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 (4.13) and (4.14), we have

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

where the first and the second-to-last inequalities use (4.16), and where the second equality uses (4.15) and the fact that $\sum_k |\mathsf{pic}(k)| = |\mathsf{pic}|$ with the sum taken over all possible keys. □

Lemma 4.51. (*) *If the footprint of a* G_3*-execution is* $\zeta(\omega)$ *for some* $\omega \in \mathsf{FP}_2^*$*, then the view of the distinguisher in this execution is identical to its view in the* G_4*-execution with the same random tapes. In particular,* $\mathcal{D}$ *outputs the same value in the two executions.*

Proof. Let $\zeta(\omega) = (\tilde{t}_1, \ldots, \tilde{t}_5)$, and consider a G_3-execution with tapes $\mathcal{T} = (t_1, \ldots, t_5)$ that has footprint $\zeta(\omega)$. By Lemma 4.47, the G_3-execution with footprint $\zeta(\omega)$ is identical to the G_2-execution with footprint ω; in particular, the tables P_i/P_i^{-1} in the two executions are identical. Thus at the end of the G_3-execution, the table P_i is the same as $\tilde{t}_i$ by Definition 26. The tape t_i is compatible with $\tilde{t}_i$, so if $P_i(x_i) = y_i$ we have $t_i(x_i) = y_i$.

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

Lemma 4.52. (*) *If the distinguisher* $\mathcal{D}$ *completes all paths, we have*

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

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 4.47, for $\omega \in \mathsf{FP}_2^*(1)$ we have $\zeta(\omega) \in \mathsf{FP}_3^*(1)$. Moreover, by Lemma 4.51, if a G_3-execution has footprint $\zeta(\omega) \in \mathsf{FP}_3^*(1)$, $\mathcal{D}$ outputs 1 in the G_4-execution with the same random tapes. Since ζ is injective, the probability that $\mathcal{D}$ outputs 1 in G_4 is at least

$$\begin{aligned}\sum_{\omega \in \mathsf{FP}_2^*(1)} \Pr_{\mathsf{G}_3}[\zeta(\omega)] &\geq \sum_{\omega \in \mathsf{FP}_2^*(1)} \Pr_{\mathsf{G}_2}[\omega] \cdot \Big(1 - \frac{169q^6}{2^n}\Big) \\ &\geq \Pr_{\mathsf{G}_2}[\mathsf{FP}_2^*(1)] - \frac{169q^6}{2^n} \qquad (4.17)\end{aligned}$$

where the first inequality uses Lemma 4.50 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 (4.17)

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

where the last inequality uses Theorem 4.41. □

Lemma 4.53. *For an arbitrary distinguisher $\mathcal{D}$, we have*

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

Proof. As discussed after Definition 22, an arbitrary distinguisher $\mathcal{D}$ with q queries can be converted to an equivalent distinguisher $\mathcal{D}'$ that completes all paths and that makes at most q extra queries in each position. $\mathcal{D}'$ makes at most $2q$ queries in each position and is subject to Lemma 4.52, so we have

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

□

Theorem 4.54. *Any distinguisher with q queries cannot distinguish G_1 from G_4 with advantage more than $10^{34}q^{38}/2^n$.*

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

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

where the first inequality uses Lemmas 4.43 and 4.53. □

Chapter 5: Conclusion

In this thesis, we present two significant advancements to the theory underlying designs of modern block ciphers. We study two classical paradigms for block cipher designs: the Feistel network and the iterated Even-Mansour construction. In Chapter 3, we showed that ten rounds of the Feistel network suffice to produce an ideal cipher, an ideal version of a block cipher, assuming that the underlying functions of the Feistel network are independent, random functions. In Chapter 4, we showed that five rounds of the iterated Even-Mansour construction suffice to produce an ideal cipher assuming that the round keys are identical and the underlying permutations of the iterated Even-Mansour construction are independent, random permutations.

The main open questions that remain from this work are establishing the exact round complexity required in the contexts of the Feistel network and the iterated Even-Mansour construction to build an ideal cipher. There has been subsequent progress since the results presented in Chapters 3 and 4. For Feistel networks, Dai and Steinberger [23] show that eight rounds suffice to produce an ideal cipher when the keyed round functions are constructed from a random oracle. This result leaves only the question of whether six or seven rounds are sufficient. (Coron et al. [17]

have shown that at least six rounds are necessary.) We believe that the results in Chapter 4 combined with the techniques of Dai and Steinberger [23] can be used to prove that seven rounds of the Feistel network produce an ideal cipher when the underlying functions are independent, random functions. However, the case of the six-round Feistel network seems much more complicated, requiring completely different proof techniques. (See Coron et al. [17] for a discussion of why this is the case.)

In the context of the exact round complexity of the iterated Even-Mansour construction, recent results have established this answer. In a recent work accepted for publication that I was involved jointly with Dai, Steinberger, and Seurin [21], we show that five rounds of the iterated Even-Mansour construction is *necessary* to produce an ideal cipher when the round keys are identical and the underlying permutations are independent, random permutations. Thus, combining this with the result in Chapter 4 which proves that five rounds are sufficient, we obtain an exact answer for the round complexity of the iterated Even-Mansour construction with the trivial key-schedule, i.e., when the round keys are identical. In the case where the round keys of the iterated Even-Mansour construction are derived in an idealized manner, i.e., the attacker holding a "master" key can derive the round keys only by querying an ideal oracle, a recent pre-print [36] claims that three rounds of the iterated Even-Mansour construction suffice to produce an ideal cipher when assuming that the idealized oracle for key derivation is a random function and the underlying permutations of the iterated Even-Mansour construction are independent, random functions.

In conclusion, we believe that the results of this thesis significantly advances the state-of-the-art in our understanding of the classical paradigms for design of modern block ciphers.

Bibliography

[1] Technical Specification Group Services 3rd Generation Partnership Project and 3G Security System Aspects. Specification of the 3gpp confidentiality and integrity algorithms; document 2: Kasumi specification, v.3.1.1, 2001.

[2] 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, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 531–550. Springer, Heidelberg, August 2013.

[3] Elena Andreeva, Andrey Bogdanov, and Bart Mennink. Towards understanding the known-key security of block ciphers. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 348–366. Springer, Heidelberg, March 2014.

[4] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, October 1997.

[5] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 491–506. Springer, Heidelberg, May 2003.

[6] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.

[7] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006.

[8] Eli Biham. New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, 7(4):229–246, 1994.

[9] Eli Biham, Ross J. Anderson, and Lars R. Knudsen. Serpent: A new block cipher proposal. In Serge Vaudenay, editor, *FSE'98*, volume 1372 of *LNCS*, pages 222–238. Springer, Heidelberg, March 1998.

[10] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and related-key attack on the full AES-256. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 231–249. Springer, Heidelberg, August 2009.

[11] John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In Matthew J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *LNCS*, pages 328–340. Springer, Heidelberg, March 2006.

[12] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 320–335. Springer, Heidelberg, August 2002.

[13] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Heidelberg, September 2007.

[14] Benoit 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, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 584–613. Springer, Heidelberg, April 2015.

[15] Benoît Cogliati and Yannick Seurin. Strengthening the known-key security notion for block ciphers. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 494–513. Springer, Heidelberg, March 2016.

[16] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, Heidelberg, August 2005.

[17] 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. *Journal of Cryptology*, 29(1):61–114, January 2016.

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

[19] Dana Dachman-Soled, Jonathan Katz, and Aishwarya Thiruvengadam. 10-round feistel is indifferentiable from an ideal cipher. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 649–678. Springer, Heidelberg, May 2016.

[20] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag, 2002.

[21] Yuanxi Dai, Yannick Seurin, John Steinberger, and Aishwarya Thiruvengadam. Indifferentiability of iterated even-mansour ciphers with non-idealized key-schedules: Five rounds are necessary and sufficient. Cryptology ePrint Archive, Report 2017/042, 2017. http://eprint.iacr.org/2017/042.

[22] Yuanxi Dai and John P. Steinberger. Feistel networks: Indifferentiability at 10 rounds, 2015. Manuscript.

[23] Yuanxi Dai and John P. Steinberger. Indifferentiability of 8-round feistel networks. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 95–120. Springer, Heidelberg, August 2016.

[24] Gregory Demay, Peter Gaži, Martin Hirt, and Ueli Maurer. Resource-restricted indifferentiability. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 664–683. Springer, Heidelberg, May 2013.

[25] Anand Desai. The security of all-or-nothing encryption: Protecting against exhaustive key search. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 359–375. Springer, Heidelberg, August 2000.

[26] Yevgeniy Dodis and Prashant Puniya. On the relation between the ideal cipher and the random oracle models. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 184–206. Springer, Heidelberg, March 2006.

[27] Yevgeniy Dodis and Prashant Puniya. Feistel networks made public, and applications. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 534–554. Springer, Heidelberg, May 2007.

[28] Yevgeniy Dodis, Martijn Stam, John P. Steinberger, and Tianren Liu. Indifferentiability of confusion-diffusion networks. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 679–704. Springer, Heidelberg, May 2016.

[29] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 210–224. Springer, Heidelberg, November 1993.

[30] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. *Journal of Cryptology*, 10(3):151–162, 1997.

[31] Pooya Farshim and Gordon Procter. The related-key security of iterated Even-Mansour ciphers. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 342–363. Springer, Heidelberg, March 2015.

[32] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, 1973.

[33] Craig Gentry and Zulfikar Ramzan. Eliminating random permutation oracles in the Even-Mansour cipher. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 32–47. Springer, Heidelberg, December 2004.

[34] Louis Granboulan. Short signatures in the random oracle model. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 364–378. Springer, Heidelberg, December 2002.

[35] 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.

[36] Chun Guo and Dongdai Lin. Indifferentiability of 3-round even-mansour with random oracle key derivation. Cryptology ePrint Archive, Report 2016/894, 2016. http://eprint.iacr.org/2016/894.

[37] 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, *43rd ACM STOC*, pages 89–98. ACM Press, June 2011.

[38] Tetsu Iwata and Tadayoshi Kohno. New security proofs for the 3GPP confidentiality and integrity algorithms. In Bimal K. Roy and Willi Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 427–445. Springer, Heidelberg, February 2004.

[39] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[40] Joe Kilian and Phillip Rogaway. How to protect DES against exhaustive key search. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 252–267. Springer, Heidelberg, August 1996.

[41] Lars R. Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 315–324. Springer, Heidelberg, December 2007.

[42] Xuejia Lai and James L. Massey. Hash function based on block ciphers. In Rainer A. Rueppel, editor, *EUROCRYPT'92*, volume 658 of *LNCS*, pages 55–70. Springer, Heidelberg, May 1993.

[43] 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, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 444–463. Springer, Heidelberg, December 2013.

[44] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2), 1988.

[45] Avradip Mandal, Jacques Patarin, and Yannick Seurin. On the public indifferentiability and correlation intractability of the 6-round Feistel construction. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 285–302. Springer, Heidelberg, March 2012.

[46] Mitsuru Matsui. New block encryption algorithm MISTY. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 54–68. Springer, Heidelberg, January 1997.

[47] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

[48] Ralph C. Merkle. One way hash functions and DES. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 428–446. Springer, Heidelberg, August 1990.

[49] National Bureau of Standards. Data encryption standard. U.S. Department of Commerce, FIPS pub. 46, January 1977.

[50] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 368–378. Springer, Heidelberg, August 1994.

[51] Zulfikar Ramzan and Leonid Reyzin. On the round security of symmetric-key cryptographic primitives. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 376–393. Springer, Heidelberg, August 2000.

[52] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with composition: Limitations of the indifferentiability framework. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, Heidelberg, May 2011.

[53] Yannick Seurin. *Primitives et Protocoles Cryptographiques à Sécurité Prouvée.* PhD thesis, Versailles University, 2009.

[54] Yannick Seurin. A note on the indifferentiability of the 10-round feistel construction. http://eprint.iacr.org/2015/903, 2011.

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

[56] Akihiro Shimizu and Shoji Miyaguchi. Fast data encipherment algorithm FEAL. In David Chaum and Wyn L. Price, editors, *EUROCRYPT'87*, volume 304 of *LNCS*, pages 267–278. Springer, Heidelberg, April 1988.

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

[58] Kazuki Yoneyama, Satoshi Miyagawa, and Kazuo Ohta. Leaky random oracle (extended abstract). In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *ProvSec 2008*, volume 5324 of *LNCS*, pages 226–240. Springer, Heidelberg, October / November 2008.