10-Round Feistel is Indifferentiable from an Ideal Cipher

Dana Dachman-Soled* Jonathan Katz† Aishwarya Thiruvengadam†

September 8, 2015

Abstract

We revisit the question of constructing an ideal cipher from a random oracle. Coron et al. (Journal of Cryptology, 2014) proved that a 14-round Feistel network using random, independent, keyed round functions is indifferentiable from an ideal cipher, thus demonstrating the feasibility of such a construction. Left unresolved is the best possible efficiency of the transformation. We improve upon the result of Coron et al. and show that 10 rounds suffice.

1 Introduction

The security of practical block ciphers—i.e., families of pseudorandom permutations—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 construction has been developed. An important area of research is to understand the provable security guarantees offered by these classical paradigms.

One of the well-known approaches for building practical block ciphers is to use a *Feistel network* [Fei73], 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 $\{0,1\}^{2n}$. 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 [LR88] 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 is indistinguishable from a random permutation under chosen plaintext/ciphertext attacks.

In the Luby-Rackoff result, the round functions are secretly keyed and thus the adversary does not have direct access to them; the security notion considered—namely, *indistinguishability*—is one in which the key of the overall Feistel network is also unknown to the adversary. A stronger

*Dept. of Electrical and Computer Engineering, University of Maryland. **Email:** danadach@ece.umd.edu. Work supported in part by NSF CAREER award #1453045. This work was done in part while the author was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #1523467.

†Dept. of Computer Science, University of Maryland. **Email:** {jkatz,aish}@cs.umd.edu. Work supported in part by NSF award #1223623.

notion of security, called *indifferentiability* [MRH04], applies even when the round functions are *public*, and aims to show that a block cipher behaves like an *ideal cipher*, i.e., an oracle where each key defines an independent, random permutation. Proving indifferentiability is more complex than proving indistinguishability: to prove indifferentiability of a block-cipher construction $\mathbf{BC}$ that relies on an ideal primitive $\mathcal{O}$ from an ideal cipher $\mathbf{IC}$, one must exhibit a *simulator* $\mathbf{S}$ such that that the view of any distinguisher interacting with $(\mathbf{BC}^{\mathcal{O}}, \mathcal{O})$ is indistinguishable from its view when interacting with $(\mathbf{IC}, \mathbf{S}^{\mathbf{IC}})$. In the context of Feistel networks, it is known (see [CHK+14]) that one can simplify the problem 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.

In a landmark result building on [CPS08, Seu09, HKT11], Coron et al. [CHK+14] proved that when using independent, random round functions, a 14-round Feistel network is indifferentiable from a public random permutation. 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 [CHK+14] that 5 rounds are not sufficient, while (as we have just noted) 14 rounds are. In this work, we narrow this gap and show that a 10-round Feistel network is indifferentiable from a random permutation.

We provide an overview of our proof, and the differences from that of Coron et al., in Section 2.

Concurrent work. Concurrent with our own work, Dai and Steinberger [DS15] have also claimed a proof of indifferentiability of a 10-round Feistel network from an ideal cipher. We have been in communication with them in order to coordinate the release of our results, but as of this writing neither team has seen the technical details of the other's work.

1.1 Other Related Work

Ramzan and Reyzin [RR00] 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 [GR04] showed that a 4-round Feistel network can be used to instantiate the random permutation in the Even-Mansour cipher [EM93] 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 [DP07] 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. [CDMP05] adapted the notion of indifferentiability to the framework of interactive Turing machines. Various relaxations of indifferentiability, such as *public indifferentiability* [DP07, YMO08], or *honest-but-curious indifferentiability* [DP06], have also been considered. Dodis and Puniya [DP06] 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. [MPS12] proved that the 6-round Feistel network is publicly indifferentiable from an ideal cipher.

1.2 Organization of the Paper

In Section 2 we provide a high-level overview of our proof, and how it differs from the prior work of Coron et al. After some brief background in Section 3, we jump into the technical details of our paper, describing our simulator in Section 4 and giving the proof of indifferentiability in Section 5.

2 Overview of Our Proof

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

2.1 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 $\mathbf{F}_1, \ldots, \mathbf{F}_r$, by always returning a uniformly random value. Unfortunately, there is a simple distinguisher who can distinguish oracle access to $(\mathsf{Feistel}_r^{\mathbf{F}}, \mathbf{F})$ from oracle access to $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$: 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 $\mathbf{P}(x_0, x_1) = (x_r, x_{r+1})$ before the distinguisher queries the entire chain, then the simulator can assign values for the remaining queries $\mathbf{F}_i(x_i)$, conditioned on the restriction $\mathsf{Feistel}_r^{\mathbf{F}}(x_0, x_1) = (x_r, x_{r+1})$. 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 $\mathbf{F}_i(x_i)$, $\mathbf{F}_{i+1}(x_{i+1})$ to be consistent with $\mathbf{P}(x_0, x_1) = (x_r, x_{r+1})$. When the simulator adapts the assignment of $\mathbf{F}_i(x_i)$ to be consistent with a constraint $\mathbf{P}(x_0, x_1) = (x_r, x_{r+1})$, we say that this value of $\mathbf{F}_i(x_i)$ has been assigned via a *ForceVal* assignment. We next discuss further details of the Coron et al. [CHK$^+$14] construction.

Partial chain detection and preemptive completion. To allow the simulator to preemptively discover $\mathbf{P}(x_0, x_1) = (x_r, x_{r+1})$, 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 $\mathbf{F}_i(x_i)$, it also checks whether there exists a tuple of the form $(x_1, x_2, x_{13}, x_{14})$ such that (1) $\mathbf{F}_1(x_1)$, $\mathbf{F}_2(x_2)$, $\mathbf{F}_{13}(x_{13})$, $\mathbf{F}_{14}(x_{14})$ have all been assigned and (2) $\mathbf{P}(\mathbf{F}_1(x_1) \oplus x_2, x_1) = (x_{14}, \mathbf{F}_{13}(x_{13}) \oplus x_{14})$; or whether there exists a tuple of the form (x_7, x_8) such that $\mathbf{F}_7(x_7)$ and $\mathbf{F}_8(x_8)$ have both been assigned. We call such a sequence of consecutive assignments a "partial chain," and when a new partial chain is detected it is "enqueued for completion" and will later be dequeued and preemptively completed. When a partial chain 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 $\mathbf{F}_i(x_i)$ can be *simultaneously* satisfied. In particular the "bad case" will be when some assignment $\mathbf{F}_i(x_i)$ must be adapted via a ForceVal assignment, but an assignment to $\mathbf{F}_i(x_i)$ 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, [CHK$^+$14] introduce the notion of a 4-round buffer zone. The simulator of [CHK$^+$14] 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 positions*. [CHK$^+$14] 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 uniformly random values to $\mathbf{F}_3(x_3)$ and $\mathbf{F}_6(x_6)$ (respectively $\mathbf{F}_9(x_9)$ and $\mathbf{F}_{12}(x_{12})$) immediately before assigning values to $\mathbf{F}_4(x_4)$ and $\mathbf{F}_5(x_5)$ (respectively $\mathbf{F}_{10}(x_{10})$ and $\mathbf{F}_{11}(x_{11})$) using ForceVal. This will ensure that ForceVal never overwrites since $x_4 = x_2 \oplus \mathbf{F}_3(x_3)$ is only determined at the moment $\mathbf{F}_3(x_3)$ is assigned and so the probability that $\mathbf{F}_4(x_4)$ has already been assigned is negligible (a similar argument holds for the other adapt positions).

Rigid structure. The rigid structure of [CHK+14] is used in their proof in two ways: First, since all assignments, across all completed chains are uniformly random except in the fixed adapt positions $\{4, 5\}$ and $\{10, 11\}$, it is easier to argue about "bad events" occurring. In particular, since the adapt zone of one chain ($\{4, 5\}$ or $\{10, 11\}$) cannot overlap with the detect zone of another chain ($\{1, 2, 13, 14\}$ or $\{7, 8\}$), they can argue that if a "bad event" occurs in a detect zone of a chain C, the bad event must have been caused by a uniform random setting of $\mathbf{F}_i(x_i)$ either during a direct distinguisher query to x_i, or during completion of some other chain D.

Bounding the simulator's runtime. The approach of [CHK+14] (originally introduced in [CPS08]) 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 $\mathbf{P}(\mathbf{F}_1(x_1) \oplus x_2, x_1) = (x_{14}, \mathbf{F}_{13}(x_{13}) \oplus x_{14})$ and so, intuitively, the distinguisher had to query either $\mathbf{P}$ or $\mathbf{P}^{-1}$ 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 have 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).

2.2 Our Techniques

We next briefly discuss how our techniques differ from the techniques of [CHK+14] in the four main areas discussed above.

Separating detection from completion for wrap-around chains. When the distinguisher makes a query $\mathbf{F}_i(x_i)$ 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 with the property that one of the set uniform positions has already been assigned. (Although this contradicts the strong set uniform property of [CHK+14], 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 reduce the complexity of 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 5.35.

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 [CHK$^+$14]) 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. In order to reason about bad events occurring due to the fact that positions do not have dedicated roles and due to the fact that partial chains in the middle detect zone are detected during the completion of other chains, we introduce the new bad events BadlyHitFV and BadlyCollideFV. Furthermore, in order to prove that a wraparound chain that has not been detected does not get created during the completion of other chains we introduce the new bad event BadlyCollideP. (See Section 5.3.1 for a list of the bad events).

Balancing detection with the simulator's runtime. In these proofs, there is a clear trade-off between the achieved security bound and the running time of the simulator. If the simulator is too "aggressive" and detects too many chains too early, then we may perhaps achieve better security at the cost of extremely high simulator complexity. In comparison to the construction of [CHK$^+$14], our construction has more detect zones and, moreover, for wraparound chains, we detect on partial chains consisting of three consecutive queries instead of four consecutive queries. Nevertheless, at a high-level, our proof that the simulator runtime is bounded follows very similarly to [CHK$^+$14]. As in [CHK$^+$14], we can first bound the number of completed partial chains of the form (x_1, x_2, x_{10}) and (x_1, x_9, x_{10}) (such chains are wraparound chains since they contain both x_1 and x_{10}). Once we have done this, we again have only a single non-wraparound detect zone and so we can follow the argument of [CHK$^+$14] 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 Background

We use the definition of indifferentiability used by Coron et al. [CHK$^+$14], based on the definition of Maurer, Renner, and Holenstein [MRH04].

Definition 1. Let $\mathbf{C}$ be a construction that, for any n, accesses functions $\mathbf{F} = (\mathbf{F}_1, \ldots, \mathbf{F}_r)$ over $\{0,1\}^n$ and implements an invertible permutation over $\{0,1\}^{2n}$. (We stress that $\mathbf{C}$ allows evaluation of both the forward and inverse direction of the permutation.) We say that $\mathbf{C}$ is *indifferentiable from a random permutation* if there exists a simulator $\mathbf{S}$ and a polynomial t such that for all distinguishers $\mathbf{D}$ making at most $q = \mathsf{poly}(n)$ queries, $\mathbf{S}$ runs in time $t(q)$ and

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

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

The r-round Feistel construction, given access to $\mathbf{F} = (\mathbf{F}_1, \ldots, \mathbf{F}_r)$, 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

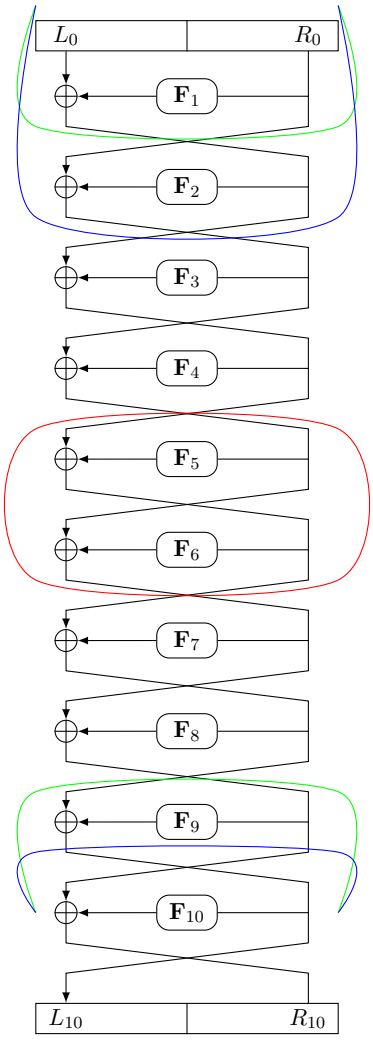

Figure 1: The 10-round Feistel construction with detect zones labeled.

output (L_i, R_i) of the i-th round of the construction is given by $L_i := R_{i-1}$ and $R_i := L_{i-1} \oplus \mathbf{F}_i(R_{i-1})$. 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) as illustrated in Figure 1.

4 Our Simulator

4.1 Informal Description of the Simulator

The queries to $\mathbf{F}_1, \ldots, \mathbf{F}_{10}$ are answered by the simulator through the procedure $\mathbf{S}.\mathrm{F}(i, x)$ for $i = 1, \ldots, 10$. When the distinguisher asks a query $\mathrm{F}(i, x)$, the simulator checks to see if 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 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. 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 $G_{i'}(x_{i'})$). Say the evaluation stopped with $x_{i'} \notin G_{i'}$. Then, the simulator adds $x_{i'}$ to $A^j_{i'}$ and checks if $i' \in \{1, 2, 5, 6, 9, 10\}$ and if so, detects any additional partial chains that form with $(x_{i'}, i')$ and enqueues them for completion if necessary and repeats the process again 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 $C = (x_k, x_{k+1}, k, \ell, g, b)$ is a chain that is enqueued to be adapted at position ℓ i.e. the "adapt" positions for C are at $\ell, \ell+1$ and the "set uniform" positions are at $\ell-1, \ell+2$ with the "set uniform" position that is adjacent to the query that caused C to be enqueued being at "good" set uniform position g and the other "set uniform" position at b. If, at the time of enqueueing, the chain 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 $x_b \notin G_b$, then C is enqueued in Q_b. (Note that there are chains that exhibit this property but are not enqueued for completion in Q_b and are not even dequeued for completion. These are the chains that belong to the set MidEquivChains. The reason for this 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 belong to CompletedChains at the end of the simulator's run while answering $\mathbf{D}$'s j^{th} query.)

The completion of enqueued chains starts with the completion of the chains enqueued in Q_b for $b \in \{1, 5, 6, 10\}$. A chain C is dequeued from Q_b and if $C \notin$ CompletedChains and $C \notin$ MidEquivChains, the simulator "completes" the chain. This process proceeds similarly to the completion process in [CHK$^+$14]. The simulator evaluates the chain forward/backward upto the 4-round buffer setting $G_i(x_i)$ values uniformly at random for any $x_i \notin G_i$ that comes up in the evaluation forward/backward. 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 at random (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 CompletedChains along with the chains that are got 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 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 the 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 $\mathrm{F}(i, x)$.

4.2 Formal Description of the Simulator

The simulator **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; these sets store values that would 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 remember 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 $A_i^j := \emptyset$ 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 procedure $\mathrm{F}(i, x)$ provides the interface to a distinguisher.

```
1  procedure F(i, x):
2      j := j + 1
3      for i ∈ {1, ..., 10} do
4          A_i^j := ∅
5      F^ENQ(i, x)
6      while ¬Q_enq.EMPTY() do
7          (x_k, x_{k+1}, k, ℓ, g, b) := Q_enq.DEQUEUE()
8          if (x_k, x_{k+1}, k) ∉ CompletedChains then
9              (x_r, x_{r+1}, r) := EVALFWDENQ(x_k, x_{k+1}, k, ℓ − 2)
10             if r + 1 = b ∧ x_{r+1} ∉ G_{r+1} then
11                 Q_b.ENQUEUE(x_k, x_{k+1}, k, ℓ, g, b)
12             (x_r, x_{r+1}, r) := EVALBWDENQ(x_k, x_{k+1}, k, ℓ + 2)
13             if r = b ∧ x_r ∉ G_r then
14                 Q_b.ENQUEUE(x_k, x_{k+1}, k, ℓ, g, b)
15     for each Q ∈ ⟨Q_1, Q_5, Q_6, Q_10, Q_all, Q_mid⟩ do        ▷ processed in that order
16         while ¬Q.EMPTY() do
17             (x_k, x_{k+1}, k, ℓ, g, b) := Q.DEQUEUE()
18             if (x_k, x_{k+1}, k) ∉ CompletedChains then
19                 (x_{ℓ−2}, x_{ℓ−1}) := EVALFWDCOMP(Q, x_k, x_{k+1}, k, ℓ − 2)
20                 (x_{ℓ+2}, x_{ℓ+3}) := EVALBWDCOMP(Q, x_k, x_{k+1}, k, ℓ + 2)
21                 ADAPT(Q, x_{ℓ−2}, x_{ℓ−1}, x_{ℓ+2}, x_{ℓ+3}, ℓ, g, b)
22                 (x_1, x_2) := EVALBWDCOMP(⊥, x_k, x_{k+1}, k, 1)
23                 (x_5, x_6) := EVALFWDCOMP(⊥, x_1, x_2, 1, 5)
24                 (x_9, x_10) := EVALFWDCOMP(⊥, x_1, x_2, 1, 9)
25                 CompletedChains := CompletedChains ∪ {(x_1, x_2, 1), (x_5, x_6, 5), (x_9, x_10, 9)}
```

```
    F^COMP(i, x)
    return G_i(x)

procedure EvalFwdEnq(x_k, x_{k+1}, k, m):
    if k = 5 then
        flagForMid:= 1
    while (k ≠ m) ∧ ((k = 10) ∨ (F^ENQ(k + 1, x_{k+1}) ≠⊥)) do
        if k = 10 then
            (x_0, x_1) := P^{-1}(x_10, x_11)
            k := 0
        else
            if k = 9 ∧ flagForMid = 1 then
                MidEquivChains := MidEquivChains ∪ {(x_k, x_{k+1}, k)}
            x_{k+2} := x_k ⊕ 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 ≠ m) ∧ ((k = 0) ∨ (F^ENQ(k, x_k) ≠⊥)) do
        if k = 0 then
            (x_10, x_11) := P(x_0, x_1)
            k := 10
        else
            if k = 1 ∧ flagForMid = 1 then
                MidEquivChains := MidEquivChains ∪ {(x_k, x_{k+1}, k)}
            x_{k−1} := x_{k+1} ⊕ G(k, x_k)
            k := k − 1
    flagForMid:= 0
    return (x_k, x_{k+1}, k)

procedure F^ENQ(i, x):
    if x ∈ G_i then
        return G_i(x)
    else if x ∈ A_i^j then
        return ⊥
    else
        A_i^j := {x} ∪ A_i^j
        if i ∈ {1, 2, 5, 6, 9, 10} then
            EnqNewChains(i, x)
        return ⊥
```

66 **procedure** EnqNewChains(i, x):
67 **if** $i = 1$ **then**
68 **for all** $(x_9, x_{10}, x_1) \in (G_9 \cup A_9^j) \times G_{10} \times \{x\}$ **do**
69 **if** CheckBwd$(x_{10}, G_{10}(x_{10}) \oplus x_9, x_1)$ **then**
70 **if** $(x_9, x_{10}, 9) \notin$ MidEquivChains **then**
71 Q_{enq}.Enqueue$(x_9, x_{10}, 9, 3, 2, 5)$
72 Q_{all}.Enqueue$(x_9, x_{10}, 9, 3, 2, 5)$
73 **if** $i = 2$ **then**
74 **for all** $(x_{10}, x_1, x_2) \in (G_{10} \cup A_{10}^j) \times G_1 \times \{x\}$ **do**
75 **if** CheckFwd$(x_2 \oplus G_1(x_1), x_1, x_{10})$ **then**
76 **if** $(x_1, x_2, 1) \notin$ MidEquivChains **then**
77 Q_{enq}.Enqueue$(x_1, x_2, 1, 4, 3, 6)$
78 Q_{all}.Enqueue$(x_1, x_2, 1, 4, 3, 6)$
79 **if** $i = 5$ **then**
80 **for all** $(x_5, x_6) \in \{x\} \times (G_6 \cup A_6^j)$ **do**
81 Q_{enq}.Enqueue$(x_5, x_6, 5, 2, 4, 1)$
82 Q_{all}.Enqueue$(x_5, x_6, 5, 2, 4, 1)$
83 **if** $i = 6$ **then**
84 **for all** $(x_5, x_6) \in (G_5 \cup A_5^j) \times \{x\}$ **do**
85 Q_{enq}.Enqueue$(x_5, x_6, 5, 8, 7, 10)$
86 Q_{all}.Enqueue$(x_5, x_6, 5, 8, 7, 10)$
87 **if** $i = 9$ **then**
88 **for all** $(x_9, x_{10}, x_1) \in \{x\} \times G_{10} \times (G_1 \cup A_1^j)$ **do**
89 **if** CheckBwd$(x_{10}, G_{10}(x_{10}) \oplus x_9, x_1)$ **then**
90 **if** $(x_9, x_{10}, 9) \notin$ MidEquivChains **then**
91 Q_{enq}.Enqueue$(x_9, x_{10}, 9, 6, 8, 5)$
92 Q_{all}.Enqueue$(x_9, x_{10}, 9, 6, 8, 5)$
93 **if** $i = 10$ **then**
94 **for all** $(x_{10}, x_1, x_2) \in \{x\} \times G_1 \times (G_2 \cup A_2^j)$ **do**
95 **if** CheckFwd$(x_2 \oplus G_1(x_1), x_1, x_{10})$ **then**
96 **if** $(x_1, x_2, 1) \notin$ MidEquivChains **then**
97 Q_{enq}.Enqueue$(x_1, x_2, 1, 7, 9, 6)$
98 Q_{all}.Enqueue$(x_1, x_2, 1, 7, 9, 6)$

99 **procedure** CheckFwd(x_0, x_1, x_{10}):
100 $(x'_{10}, x'_{11}) := \mathbf{P}(x_0, x_1)$
101 **return** $x'_{10} \stackrel{?}{=} x_{10}$

102 **procedure** CheckBwd(x_{10}, x_{11}, x_1):
103 $(x'_0, x'_1) := \mathbf{P}^{-1}(x_{10}, x_{11})$
104 **return** $x'_1 \stackrel{?}{=} x_1$

105 **procedure** EvalFwdComp(Q, x_k, x_{k+1}, k, m):

```
    while k ≠ m do
        if k = 10 then
            (x_0, x_1) := P^{-1}(x_10, x_11)
            k := 0
        else
            x_{k+2} := x_k ⊕ F^COMP(Q, k + 1, x_{k+1})
            k := k + 1
    return (x_m, x_{m+1})

procedure EvalBwdComp(Q, x_k, x_{k+1}, k, m):
    while k ≠ m do
        if k = 0 then
            (x_10, x_11) := P(x_0, x_1)
            k := 10
        else
            x_{k-1} := x_{k+1} ⊕ F^COMP(Q, k, x_k)
            k := k − 1
    return (x_m, x_{m+1})

procedure F^COMP(Q, i, x):
    if x ∉ G_i then
        G_i(x) ← {0,1}^n
        if Q ≠⊥ ∧Q = Q_all ∧ i ∈ {5, 6} then
            EnqNewMidChains(i, x)
    return G_i(x)

procedure EnqNewMidChains(i, x):
    if i = 5 then
        for all (x_5, x_6) ∈ {x} × G_6 do
            Q_mid.Enqueue(x_5, x_6, 5, 2, 4, 1)
    if i = 6 then
        for all (x_5, x_6) ∈ G_5 × {x} do
            Q_mid.Enqueue(x_5, x_6, 5, 8, 7, 10)

procedure Adapt(Q, x_{ℓ−2}, x_{ℓ−1}, x_{ℓ+2}, x_{ℓ+3}, ℓ, g, b):
    flagMidAdapt0 := 0
    flagMidAdapt1 := 0
    F^COMP(Q, ℓ − 1, x_{ℓ−1})
    x_ℓ := x_{ℓ−2} ⊕ G_{ℓ−1}(x_{ℓ−1})
    if (Q = Q_all) ∧ (ℓ = 5 ∨ ℓ = 6) ∧ (x_ℓ ∉ G_ℓ) then
        flagMidAdapt0 := 1
    F^COMP(Q, ℓ + 2, x_{ℓ+2})
    x_{ℓ+1} := x_{ℓ+3} ⊕ G_{ℓ+2}(x_{ℓ+2})
    if (Q = Q_all) ∧ (ℓ + 1 = 5 ∨ ℓ + 1 = 6) ∧ (x_{ℓ+1} ∉ G_{ℓ+1}) then
```

```
146         flagMidAdapt1 := 1
147     ForceVal(x_ℓ, x_{ℓ+1} ⊕ x_{ℓ−1}, ℓ)
148     if flagMidAdapt0 = 1 then
149         EnqNewMidChains(ℓ, x_ℓ)
150     ForceVal(x_{ℓ+1}, x_ℓ ⊕ x_{ℓ+2}, ℓ + 1)
151     if flagMidAdapt1 = 1 then
152         EnqNewMidChains(ℓ + 1, x_{ℓ+1})

153 procedure ForceVal(x, y, ℓ):
154     G_ℓ(x) := y
```

5 Proof of Indifferentiability

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

Theorem 5.1. *The probability that a distinguisher* $\mathbf{D}$ *making at most* q *queries outputs 1 in an interaction with* $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ *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,* $\mathbf{S}$ *runs in time polynomial in* q*.*

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

5.1 Proof Overview

Our proof structure utilizes four hybrid experiments $H_1, \ldots, H_4$ as in the proof of Coron et al. [CHK+14]. Hybrid H_1 denotes the scenario in which $\mathbf{D}$ interacts with $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$, and H_4 denotes the scenario in which $\mathbf{D}$ interacts with $(\mathsf{Feistel}^{\mathbf{F}}, \mathbf{F})$. To prove indifferentiability, we show that the difference between the probability $\mathbf{D}$ outputs 1 in H_1 and the probability $\mathbf{D}$ outputs 1 in H_4 is at most $\text{poly}(q)/2^n$.

In H_2, we replace the random permutation $\mathbf{P}$ in H_1 with a two-sided random function $\mathbf{R}$ that also implements $\mathbf{R}$.CheckFwd and $\mathbf{R}$.CheckBwd. Following [CHK+14], 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 then argue that the *simulator*'s views are indistinguishable in H_1 and H_2. However, in H_1 the simulator itself implements CheckFwd and CheckBwd, while in H_2 the procedures $\mathbf{R}$.CheckFwd and $\mathbf{R}$.CheckBwd are provided by the two-sided random function $\mathbf{R}$. Thus, we introduce an additional hybrid $H_{1.5}$ in which $\mathbf{D}$ interacts with $(\mathbf{P}, \hat{\mathbf{S}}^{\mathbf{P}^+})$, where $\mathbf{P}^+$ denotes a random permutation equipped with additional procedures CheckFwd and CheckBwd that are implemented as in the simulator $\mathbf{S}$. The simulator $\hat{\mathbf{S}}$ in $H_{1.5}$ implements the procedures CheckFwd and CheckBwd by simply calling $\mathbf{P}^+$.CheckFwd and $\mathbf{P}^+$.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 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 $\mathbf{R}$ from H_2 with the 10-round Feistel construction $\mathsf{Feistel}$. Then, the distinguisher $\mathbf{D}$ interacts with $(\mathsf{Feistel}, \hat{\mathbf{S}}^{\mathsf{Feistel}^+})$ where $\mathsf{Feistel}^+$ (similar to $\mathbf{P}^+$) 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 $\mathbf{D}$ is exactly the probability with which the "bad events" occur by explicitly mapping the randomness used in H_2 and H_3. The mapping and the proof follows exactly along the lines of the proof in [CHK+14].

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

5.2 Indistinguishability of the First and Second Experiments

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

Lemma 5.2. *The probability that* $\mathbf{D}$ *outputs 1 in* H_1 *is equal to the probability it outputs 1 in* $H_{1.5}$*. Moreover, the number of queries* $\mathbf{S}$ *makes to* $\mathbf{P}$ *plus the number of times* $\mathbf{S}$ *internally runs* CheckFwd *and* CheckBwd *in* H_1 *is equal to the number of queries* $\hat{\mathbf{S}}$ *makes to* $\mathbf{P}^+$ *in* $H_{1.5}$*.*

Experiment H_2 differs from $H_{1.5}$ in that we replace the random permutation $\mathbf{P}^+$ with a random two-sided function $\mathbf{R}$. This two-sided function maintains a hashtable P containing elements of the form $(\downarrow, x_0, x_1)$ and $(\uparrow, x_{10}, x_{11})$. Whenever the procedure $\mathbf{R}.\mathrm{P}(x_0, x_1)$ is queried, $\mathbf{R}$ checks if $(\downarrow, x_0, x_1) \in P$ and if so, answers accordingly. 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.

In addition to procedures P and P^{-1}, $\mathbf{R}$ contains procedures CheckFwd(x_0, x_1, x_{10}) and CheckBwd(x_{10}, x_{11}, x_1).[1] 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 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 otherwise.

[1] This is similar to the CHECK procedure in [CHK+14].

The pseudocode for the two-sided random function $\mathbf{R}$, using hashtable P, is as follows:

```
1  procedure P(x_0, x_1):
2      if (↓, x_0, x_1) ∉ P then
3          (x_10, x_11) ←$ {0,1}^{2n}
4          P(↓, x_0, x_1) := (x_10, x_11)
5          P(↑, x_10, x_11) := (x_0, x_1) //May overwrite an entry
6      return P(↓, x_0, x_1)

7  procedure P^{-1}(x_10, x_11):
8      if (↑, x_10, x_11) ∉ P then
9          (x_0, x_1) ←$ {0,1}^{2n}
10         P(↑, x_10, x_11) := (x_0, x_1)
11         P(↓, x_0, x_1) := (x_10, x_11) //May overwrite an entry
12     return P(↑, x_10, x_11)

13 procedure CheckFwd(x_0, x_1, x_10):
14     if (↓, x_0, x_1) ∈ P then
15         (x'_10, x'_11) := P(↓, x_0, x_1)
16         return x'_10 ?= x_10
17     return false

18 procedure CheckBwd(x_10, x_11, x_1):
19     if (↑, x_10, x_11) ∈ P then
20         (x'_0, x'_1) := P(↑, x_10, x_11)
21         return x'_1 ?= x_1
22     return false
```

Figure 2: Random Two-sided Function $\mathbf{R}$

In the following section, we show that H_2 and $H_{1.5}$ are indistinguishable.

5.2.1 Indistinguishability of H_2 and $H_{1.5}$

We first bound the effect of replacing $\mathbf{P}^+$ with $\mathbf{R}$ as a function of the total number of queries q' made to this oracle. Note, however, that both the simulator and $\mathbf{D}$ make queries to this oracle. Thus, 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 $\mathbf{D}$.

Theorem 5.3. *Let $\mathbf{S}'$ be an algorithm that makes at most q' queries to an oracle. Then the difference between the probability that $\mathbf{S}'$ outputs 1 when interacting with $\mathbf{P}^+$, and the probability that $\mathbf{S}'$ outputs 1 when interacting with $\mathbf{R}$ is at most $12q'^2/2^n$.*

Proof. Recall that both the simulator and $\mathbf{D}$ make queries to the oracle. So, $\mathbf{S}'$ consists of both the simulator and $\mathbf{D}$ and the number of queries to $\mathbf{S}'$ corresponds to the queries to $\mathrm{P}/\mathrm{P}^{-1}$ and

CheckFwd/CheckBwd made by both the simulator and the distinguisher. In order to prove this theorem, we consider experiments $E_0, E_1, \ldots, E_3$, where E_0 corresponds to $\mathbf{S}'$ interacting with $\mathbf{P}^+$, and E_3 corresponds to $\mathbf{S}'$ interacting with $\mathbf{R}$.

Experiment E_0: $\mathbf{S}'$ interacts with $\mathbf{P}^+$.

Experiment E_1: $\mathbf{S}'$ interacts with $\mathbf{P}_1$ where procedure $\mathbf{P}_1$.P is defined as follows:

procedure $\mathbf{P}_1.\mathrm{P}(x_0, x_1)$
  if $(\downarrow, x_0, x_1) \notin P$ **then**
    $(x_{10}, x_{11}) \xleftarrow{\$} \{0,1\}^{2n}$
    if $(\uparrow, x_{10}, x_{11}) \in P$ **then**
      $(x_{10}, x_{11}) \xleftarrow{\$} \{0,1\}^{2n} \setminus \{(x'_{10}, x'_{11}) | (\uparrow, x'_{10}, x'_{11}) \in P\}$
    $P(\downarrow, x_0, x_1) := (x_{10}, x_{11})$
    $P(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$
  return $P(\downarrow, x_0, x_1)$

Procedure $\mathbf{P}_1.\mathrm{P}^{-1}$ is defined analogously. Procedures $\mathbf{P}_1$.CheckFwd and $\mathbf{P}_1$.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 $x'_{10} \stackrel{?}{=} x_{10}$
  $(x'_{10}, x'_{11}) := \mathrm{P}(x_0, x_1)$ //Note that procedure $\mathbf{P}_1$.P is called
  return $x'_{10} \stackrel{?}{=} x_{10}$

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 $x'_1 \stackrel{?}{=} x_1$
  $(x'_0, x'_1) := \mathrm{P}^{-1}(x_{10}, x_{11})$ //Note that procedure $\mathbf{P}_1.\mathrm{P}^{-1}$ is called
  return $x'_1 \stackrel{?}{=} x_1$

Claim 5.4. *The probabilities that $\mathbf{S}'$ outputs 1 in E_0 and E_1 are identical.*

Proof. The values assigned through procedures P and P^{-1} are chosen uniformly at random 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, the experiments E_0 and E_1 behave identically. □

Experiment E_2: $\mathbf{S}'$ interacts with $\mathbf{P}_2$ where the procedures $\mathbf{P}_2$.P and $\mathbf{P}_2.\mathrm{P}^{-1}$ are defined exactly as $\mathbf{R}$.P and $\mathbf{R}.\mathrm{P}^{-1}$ respectively while procedures $\mathbf{P}_2$.CheckFwd and $\mathbf{P}_2$.CheckBwd are defined as in $\mathbf{P}_1$.CheckFwd and $\mathbf{P}_1$.CheckBwd respectively.

Claim 5.5. *The probability that* $\mathbf{S}'$ *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 [CHK+14, Lemma 3.8]. □

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

Claim 5.6. *The probability that* $\mathbf{S}'$ *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 BadCheckForward occurs if $\mathbf{P}'$.CheckFwd returns true in the last line in E_2. Similarly, the event BadCheckBackward occurs if $\mathbf{P}'$.CheckBwd returns true in the last line in E_2. The probability of the events BadCheckForward and BadCheckBackward is at most $q'/2^n$.

The event BadOverwrite occurs if either in 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 BadOverwrite occurs in E_2 and E_3 is at most $2(q')^2/2^{2n}$.

The event 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 CheckFwd query, and $P(\downarrow, x_0, x_1)$ is set to (x_{10}^*, x_{11}^*).

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

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

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

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

(ii) After (i), the query $\mathrm{P}(x_0^*, x_1^*)$, or CheckFwd(x_0^*, x_1^*, x_{10}) or 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 event (BadBackwardQuery $\wedge \neg$BadCheckForward $\wedge \neg$BadCheckBackward) occurs. To analyze the probability of this event, let us consider the first case by which BadBackwardQuery can occur. Consider any pair $(x_0, x_1, x_{10}^*, x_{11}^*)$ such that (i) of event (1) holds. Since BadCheckForward does not occur, we have that the CheckFwd query returns false. Now, as long as the queries $\mathrm{P}(x_0, x_1)$, $\mathrm{P}^{-1}(x_{10}^*, x_{11}^*)$, CheckFwd$(x_0, x_1, x_{10}^*)$ or 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 CheckFwd(x_0, x_1, x_{10}') and CheckBwd(x_1, x_{10}', x_{11}') was 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)

CheckFwd(x_0, x_1, x^*_{10}) is at most $\frac{q'}{2^n - q'}$ or c) 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, probability of (BadBackwardQuery$\wedge\neg$BadCheckForward$\wedge\neg$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 scenarios E_2 and E_3 behave identically. □

Theorem 5.3 follows from Claims 5.4–5.6. □

Theorem 5.3 bounds the effect of replacing $\mathbf{P}^+$ with $\mathbf{R}$ in terms of the total number of queries q' made to the $\mathbf{P}^+/\mathbf{R}$ oracle. In the following sequence of claims, we show that $q' = \mathsf{poly}(q)$ (where, recall, q is the total number of queries made by the original distinguisher $\mathbf{D}$).

Claim 5.7. *In H_2, the simulator dequeues a partial chain of the form $(x_1, x_2, 1, \ell, g, b)$ such that $(x_1, x_2, 1) \notin$ CompletedChains at most q times in total from the queue Q_{enq}.*

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 CheckFwd$(x_2 \oplus G_1(x_1), x_1, x_{10}) =$ 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 CheckFwd$(x_2 \oplus G_1(x_1), x_1, x_{10}) =$ 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 at random in line 3 of $\mathbf{R}$ or to $\mathrm{P}^{-1}(x_{10}, x_{11})$ such that (x_0, x_1) was chosen uniformly at random in line 9 of $\mathbf{R}$ for some $x_{11} \in \{0,1\}^n$. This call to P or P^{-1} was made by the distinguisher or the simulator. Now if $(x_1, x_2, 1) \notin$ 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 EvalFwdEnq or EvalBwdEnq
2. in a call to EvalFwdComp or EvalBwdComp

Consider the case where the simulator queried $\mathrm{P}/\mathrm{P}^{-1}$ during a call to EvalFwdEnq(x_k, x_{k+1}, k, m) or EvalBwdEnq(x_k, x_{k+1}, k, m). If $m \in \{1, 9\}$, 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 $m = 5$, 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 in any of the queues 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, it 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 CheckFwd$(x'_2 \oplus G_1(x'_1), x'_1, x'_{10}) =$ true due to the same distinguisher query. If $\mathbf{D}$'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 $\mathbf{D}$'s query was $(\uparrow, x_{10}, x_{11})$ that returned (x_{10}, x_{11}), we have the same result again. So, there is a unique distinguisher query corresponding to the chain $(x_1, x_2, 1)$. Since $\mathbf{D}$ makes at most q queries, we have that there are most q such partial chains dequeued. □

The following can be proved similarly to Claim 5.7.

Claim 5.8. *In H_2, the simulator dequeues a partial chain of the form $(x_9, x_{10}, 9, \ell, g, b)$ such that $(x_9, x_{10}, 9) \notin \mathsf{CompletedChains}$ at most q times in total from the queue Q_{enq}.*

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

Claim 5.9. *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) 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 (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 5.7, 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 5.8, 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 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: (i) the distinguisher queried $\mathrm{F}(5, x_5)$ (ii) the simulator made a call to EvalFwdEnq or EvalBwdEnq during the completion of a $(x_9, x_{10}, 9)$ chain where $(x_9, x_{10}, 9) \notin \mathsf{CompletedChains}$ (iii) 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 5.7 and 5.8. 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 bound by $\sum_{j=1}^{q} |A_5^j| . |G_6 \cup A_6^j| + \sum_{j=1}^{q} |G_5 \cup A_5^j| . |A_6^j|$ and hence can be bound by $54q^2$. And, the number of partial chains of the form $(x_5, x_6, 5)$ partial chain that are enqueued in Q_{mid} can be bound by $|G_5| \cdot |G_6|$ and hence can be bound by $36q^2$. □

Claim 5.10. *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) 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 (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 5.7 and 5.8. There are at most $90q^2$ completions of a $(x_5, x_6, 5)$ partial chain by Claim 5.9. So, $|G_i| \leq 90q^2 + 3q \leq 93q^2$. □

Claim 5.11. *In H_2, the simulator makes at most $644q^2$ queries to $\mathbf{R}.\mathrm{P}$ and $\mathbf{R}.\mathrm{P}^{-1}$.*

Proof. The simulator makes calls to $\mathbf{R}.\mathrm{P}/\mathbf{R}.\mathrm{P}^{-1}$ only in procedures EvalFwdEnq, EvalBwdEnq, EvalFwdComp and EvalBwdComp. By bounds on the number of partial chains that are enqueued for completion given in Claims 5.7, 5.8 and 5.9 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 5.12. *In H_2, the simulator makes at most $3,166,000q^6$ queries to the procedures $\mathbf{R}.$CheckFwd and $\mathbf{R}.$CheckBwd.*

Proof. The number of 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 EvalFwdEnq or EvalBwdEnq during the completion of a $(x_1, x_2, 1)$ chain where $(x_1, x_2, 1) \notin \mathsf{CompletedChains}$ (c) the simulator made a call to EvalFwdEnq or EvalBwdEnq during the completion of a $(x_5, x_6, 5)$ chain.

There are at most q distinguisher queries and by Claims 5.8 and 5.9, there are at most q and $90q^2$ such calls to EvalFwdEnq and EvalBwdEnq. So, $\sum_{j=i}^{q} A_{10}^j \leq 90q^2 + 2q \leq 92q^2$. In a similar manner, we can bound $\sum_{j=i}^{q} A_2^j \leq 90q^2 + 2q \leq 92q^2$.

So, the number of CheckFwd queries made by the simulator is bounded by $1582860q^6$. By a similar argument, the number of CheckBwd queries made by the simulator is bounded by $1582860q^6$. Hence, there are at most $O(q^6)$ queries to $\mathbf{R}$.CheckFwd and $\mathbf{R}$.CheckBwd. □

Corollary 5.13. *In $H_{1.5}$ and H_1, the simulator makes at most $3.2 \times 10^6 q^6$ queries to $\mathbf{P}^+$ and $\mathbf{P}$ 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 $\mathbf{D}$ 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 $\mathbf{P}^+$ with probability greater than $\frac{10^{15}q^{12}}{2^n}$. By Claims 5.11 and 5.12, the simulator makes at most $3,166,644q^6$ queries to $\mathbf{R}$ in H_2. Now, consider a distinguisher $\mathbf{S}'$ which aims to distinguish $\mathbf{P}^+$ and $\mathbf{R}$ making only $r'(q)$ queries. The distinguisher $\mathbf{S}'$ consists of $\mathbf{D}$ and the simulator together. $\mathbf{S}'$ outputs 1 if $\mathbf{D}$ and the simulator make more than $3.2 \times (10q)^6$ queries. Then, $\mathbf{S}'$ issues at most $3.2 \times (10q)^6$ queries and distinguishes $\mathbf{R}$ from $\mathbf{P}^+$ 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 5.3. Hence, the simulator makes at most $3.2 \times 10^6 q^6$ queries to $\mathbf{P}^+$ except with probability $\frac{10^{15}q^{12}}{2^n}$. Combining this with Lemma 5.2, we get the result. □

Lemma 5.14. *The probability that $\mathbf{D}$ 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 5.2, we have that the distinguisher $\mathbf{D}$ outputs 1 in H_1 and $H_{1.5}$ with the same probability. Now, for the sake of contradiction, assume that $\mathbf{D}$ distinguishes $H_{1.5}$ from H_2 with advantage greater than $\frac{2 \cdot 10^{15}q^{12}}{2^n}$. We construct a distinguisher $\mathbf{S}'$ that consists of $\mathbf{D}$ and the simulator where $\mathbf{S}'$ makes at most $3.3 \times (10q)^6$ queries and distinguishes $\mathbf{R}$ from $\mathbf{P}^+$. $\mathbf{S}'$ works as follows: $\mathbf{S}'$ outputs 1 if the number of queries issued by $\mathbf{D}$ and the simulator exceeds $3.3 \times (10q)^6$; otherwise, it outputs whatever is output by $\mathbf{D}$. By Corollary 5.13, 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 5.11 and 5.12, the simulator makes at most $3,166,644q^6$ queries in H_2. So, distinguisher $\mathbf{S}'$ making at most $3.3 \times (10q)^6$ queries can distinguish $\mathbf{R}$ and $\mathbf{P}^+$ 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 5.3. □

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

Claim 5.15. *In H_2, the simulator runs in time $O(q^6)$.*

Proof. We analyze the running time of the procedures of $\hat{\mathbf{S}}$. 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 5.10 and by the bounds on $\sum_{j=1}^{q} |A_i^j|$ for $i \in \{1, 2, 5, 6, 9, 10\}$ derived in Claims 5.9 and 5.12. Procedure EnqNewMidChains runs in time $O(q^2)$ by the bounds on $|G_5|$ and $|G_6|$ derived in Claim 5.9.

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 5.7, 5.8 and 5.9. 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 5.7, 5.8 and 5.9. 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 5.16. *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 5.11 and 5.12, the simulator makes at most $3.2 \times (10q)^6$ queries to $\mathbf{R}$ in H_2 and by Claim 5.15, 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 $\mathbf{D}$ 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 $\mathbf{P}^+$ with probability greater than $\frac{10^{15}q^{12}}{2^n}$. Now, consider a distinguisher $\mathbf{S}'$ which aims to distinguish $\mathbf{P}^+$ and $\mathbf{R}$ making only $r'(q)$ queries. The distinguisher $\mathbf{S}'$ consists of $\mathbf{D}$ and the simulator together. $\mathbf{S}'$ outputs 1 if the simulator runs for more than $r(q)$ steps or if $\mathbf{D}$ and the simulator make more than $3.3 \times (10q)^6$ queries. Then, $\mathbf{S}'$ issues at most $3.3 \times (10q)^6$ queries and distinguishes $\mathbf{R}$ from $\mathbf{P}^+$ 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 5.3. Combining this with Lemma 5.2, we get the result. □

5.3 Properties of H_2

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 [CHK+14]. 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 2. Fix tables $G = \hat{\mathbf{S}}.G$ and $P = \mathbf{R}.P$ 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:

```
1  procedure next(x_k, x_{k+1}, k):
2      if k < 10 then
3          if x_{k+1} ∉ G_{k+1} then return ⊥
4          x_{k+2} := x_k ⊕ G_{k+1}(x_{k+1})
5          return (x_{k+1}, x_{k+2}, k + 1)
6      else if k = 10 then
7          if (↑, x_10, x_11) ∉ P then return ⊥
8          (x_0, x_1) := P(↑, x_10, x_11)
9          return (x_0, x_1, 0)

10 procedure prev(x_k, x_{k+1}, k):
11     if k > 0 then
12         if x_k ∉ G_k then return ⊥
13         x_{k−1} := x_{k+1} ⊕ G_k(x_k)
14         return (x_{k−1}, x_k, k − 1)
15     else if k = 0 then
16         if (↓, x_0, x_1) ∉ P then return ⊥
17         (x_10, x_11) := P(↓, x_0, x_1)
18         return (x_10, x_11, 10)

19 procedure val_i^+(C):
20     while (C ≠⊥) ∧ (C[3] ∉ {i − 1, i}) do
21         C := next(C)
22     if C =⊥ then return ⊥
23     if C[3] = i then return C[1]
24     else return C[2]

25 procedure val_i^−(C):
26     while (C ≠⊥) ∧ (C[3] ∉ {i − 1, i}) do
27         C := prev(C)
28     if C =⊥ then return ⊥
29     if C[3] = i then return C[1]
30     else return C[2]

31 procedure val_i(C):
32     if val_i^+(C) ≠⊥ then return val_i^+(C)
33     else return val_i^−(C)
```

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

Definition 3. 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 4. The set of *table-defined* chains contains all chains C for which $\mathsf{next}(C) \neq \perp$ and $\mathsf{prev}(C) \neq \perp$.

Definition 5. 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 6. The set Q^*_{all} contains chains that are enqueued in Q_{all} but not in Q_1, Q_5, Q_6, Q_{10}.

Definition 7. 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 random 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 $\textsc{ForceVal}(x_k, \cdot, k)$ call, then it is not a uniform assignment.

Definition 8. We say that a *uniform assignment to* P occurs in a call to $\mathbf{R}.\mathrm{P}(x_0, x_1)$ 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 at random from the set of $2n$-bit strings.

Similarly, it occurs in a call to $\mathbf{R}.\mathrm{P}^{-1}(x_{10}, x_{11})$ 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 at random from the set of $2n$-bit strings.

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

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.

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 $\textsc{Adapt}$ procedure in an execution of H_2. The bound on T is derived from Claims 5.7–5.11.

5.3.1 Bad Executions

Definition 9. We say that event BadP occurs in H_2 if either:

- Immediately after choosing (x_{10}, x_{11}) in a call to $\mathbf{R}.\mathrm{P}(\cdot, \cdot)$, 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 $\mathbf{R}.\mathrm{P}^{-1}(\cdot, \cdot)$, either $(\downarrow, x_0, x_1) \in P$ or $x_1 \in G_1$.

Lemma 5.17. *The probability of event* BadP *in* H_2 *is at most* $2\,T^2/2^n$.

Proof. The proof follows exactly as in [CHK+14, Lemma 3.18]. □

Definition 10. 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 5.18. *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 11. 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 of 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 5.19. *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. Case 1: After the assignment C and D are table-defined. The proof for this case follows exactly as in [CHK+14, 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 $x_k \in A^j_k$ and $x_{k+1} \in A^j_{k+1}$ 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 $i = k$. 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, we can't have 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 $x_k \in G_k, x_{k+1} \in A^j_{k+1}$ and $x_k \in A^j_k, x_{k+1} \in G_{k+1}$ 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 computed 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 $10T$ such assignments of the form $G_i(x_i) := f(i, x_i)$ and there are at most 4 possibilities for σ, ρ. Thus, the probability is $10T \cdot T \cdot (11T^2 + 2T) \cdot 4 \cdot 4/2^n \leq 2080T^4/2^n$.

Case 3: The proof follows similar to the proof of Case 2. □

Definition 12. We say that event $\mathsf{BadlyCollideP}$ 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, $\mathsf{val}^\sigma_1(C) = \perp$ or $\mathsf{val}^\rho_1(D) = \perp$.
  - Immediately after the assignment, $\mathsf{val}^\sigma_1(C) = \mathsf{val}^\rho_1(D) = x_1 \neq \perp$.

  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 5.20. *The probability of event* BadlyCollideP *in* H_2 *is at most* $314\,T^5/2^n$.

Proof. Consider the case that after a uniform random choice of (x_0, x_1) leading to an assignment $P(\uparrow, x_{10}, x_{11}) := (x_0, x_1)$ the event 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 is similar. 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 BadlyCollideP is at most $\frac{T\cdot((11T^2+T)^2+T^2+T\cdot(11T^2+T))\cdot 2}{2^n}$. □

Definition 13. We say that event BadlyHitFV occurs in H_2 if 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 5.21. *The probability of event* BadlyHitFV *in* H_2 *is at most* $2\,T^3/2^n$.

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 $x'_{\ell+1} \in G_{\ell+1}$, 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 14. We say that event BadlyCollideFV occurs in H_2 if a uniform assignment to $G_s(x_s)$ that occurs in a call to $\text{ADAPT}(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$, 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 5.22. *The probability of event* BadlyCollideFV *in* H_2 *is at most* $4T^3/2^n$.

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. (This is because T is the upper bound on the number of chains enqueued in Q_{all} by assumption and on the size of $G_{\ell+1}$.) 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, BadlyHit$^+$, BadlyCollide$^+$, BadlyCollideP, BadlyHitFV or BadlyCollideFV occur. Lemmas 5.17–5.22 thus imply:

Lemma 5.23. *The probability that an execution of* H_2 *is good is* $O(T^5)/2^n$.

5.3.2 Properties of Good Executions

The aim of this section is to 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. to prove that a ForceVal call does not "overwrite".

Before we proceed with the proofs, we introduce some notation here. 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 "bad" set uniform position. So, if $b = \ell-1$, then $a = \ell$; Else, if $b = \ell+2$, $a = \ell+1$. Consider a call Adapt$(x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$, if $b = \ell-1$ define $x_a = x_\ell$ as $x_\ell := x_{\ell-2} \oplus G_{\ell-1}(x_{\ell-1})$ if $x_{\ell-1} \in G_{\ell-1}$, and $x_\ell = \perp$ otherwise. Analogously, if $b = \ell+2$, define $x_a = x_{\ell+1} := x_{\ell+3} \oplus G_{\ell+2}(x_{\ell+2})$ if $x_{\ell+2} \notin G_{\ell+2}$ and $x_{\ell+1} = \perp$ otherwise.

Also, for a chain C enqueued in Q_b we say adapting is "safe" if just before the call to 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 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 ForceVal does not "overwrite", we will prove that for every call to 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 Adapt is "safe". In order to prove the above statements, we will 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 just before the assignment that precedes C being enqueued occurs, we will prove that $\mathsf{val}_g(C) = \perp$ and $\mathsf{val}_b(C) = \perp$. 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 5.38). While the structure explained above is similar to the structure of the proof in [CHK+14], 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 5.24. *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 follows 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 5.25. *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 5.24, 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), $\mathsf{val}_i(C)$ 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 5.26. *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 $C = (x_1, x_2, 1, 4, 3, 6)$ where $g = 3$ and $b = 6$. Such a chain C is enqueued in Q_6, only if $x_2 \notin G_2$, by construction of the simulator. Otherwise, EnqNewChains$(2, x_2)$ is not called. Since $x_2 \notin G_2$ when enqueued, $\mathsf{val}_3^+(C) = \perp$ 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 $\mathsf{val}_3(C) = \perp$ and $\mathsf{val}_6(C) \notin G_6$ 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 5.27. *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 [CHK+14, Lemma 3.26(a)]. □

Corollary 5.28. *In a good execution of H_2, consider a call to* $\textsc{Adapt}(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$ *and assume that adapting is "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* $\textsc{ForceVal}(x_\ell, \cdot, \ell)$ *and* $\textsc{ForceVal}(x_{\ell+1}, \cdot, \ell+1)$ *that occurs in the call* $\textsc{Adapt}(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$*, we have $x_\ell \notin G_\ell$ and $x_{\ell+1} \notin G_{\ell+1}$ respectively.*

Proof. The proof follows immediately from Claim 5.27. □

Claim 5.29. *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* $\textsc{Adapt}(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$*. 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* $\textsc{ForceVal}(x_\ell, \cdot, \ell)$ *(respectively* $\textsc{ForceVal}(x_{\ell+1}, \cdot, \ell+1)$*).*

Proof. The proof follows exactly as in Lemma [CHK+14, Lemma 3.26(b)]. □

Claim 5.30. *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* $\textsc{Adapt}(Q, x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell, g, b)$*. 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* $\textsc{ForceVal}(x_\ell, \cdot, \ell)$ *(respectively* $\textsc{ForceVal}(x_{\ell+1}, \cdot, \ell+1)$*) 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 $\textsc{ForceVal}(x_\ell, \cdot, \ell)$ only if $\mathsf{val}^\rho_\ell(C) = x_\ell$ 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 after the assignment. 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 5.31. *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 is "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 that the lemma has held till 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 "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 $\textsc{ForceVal}(x_a, \cdot, a)$ 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 5.26. 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 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 5.30, this cannot be true. (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 5.32. *Consider 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 is "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 $x_5 \notin G_5 \cup A^t_5$ or $x_6 \notin G_6 \cup A^t_6$ or both when Q_{enq}.EMPTY() = true in line 6 of the simulator's execution after $\mathbf{D}$'s t^{th} query.

Consider a chain $C = (x_5, x_6, 5, 2, 4, 1)$ which 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 $x_6 \in G_6$, $\mathsf{val}_5(C) = \mathsf{val}_5(D) = x_5$ 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 $\mathsf{val}_1(C) \neq \perp$ at the time of the assignment that precedes the enqueueing of C, we need $\mathsf{val}^+_1(C) \neq \perp$. Then, in particular, we have that $x_7 := \mathsf{val}_7(C) \in G_7$ and $x_8 := \mathsf{val}_8(C) \in G_8$

(otherwise, $\mathsf{val}_9^+(C) = \perp$ implying that $\mathsf{val}_1^+(C) = \perp$).

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 $\mathsf{val}_9^+(C) \neq \perp$ just before the assignment that precedes the enqueueing of C, we need (1) $C_1 = \mathsf{next}(C)$, $C_2 = \mathsf{next}(C_1)$ (and hence, $x_6 \in G_6$ and $x_7 \in G_7$) and (2) $x_5 = \mathsf{val}_5(D)$ for a chain D in Q^*_{all} and (3) $x_8 \in G_8$ or $x_8 = \mathsf{val}_8(E)$ of 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 that 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 BadP occurred. It cannot be a ForceVal assignment since ForceVal does not change the value of a chain enqueued in Q_{all} by Claims 5.30 and 5.31. If it were a uniform assignment to $G_i(x_i)$, then, $\mathsf{BadlyCollide}^+$ occurred.

Consider the case that 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 it 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 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}. By assumption for chains in Q^*_{all}, we have $\mathsf{val}_5(X) \notin G_5$ before the Adapt call for such a chain or, (b) $Y = (x_1, x_2, 1)$ such that $(x_1, x_2, 1, 7, 9, 6)$ is enqueued in Q_6. In this case, 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, 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 5.33. *Consider a good execution of H_2. Just before the execution of line 27 during the simulator's execution, if adapting is "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 $\mathbf{R}.\textsc{CheckBwd}(x_{10}, x_9 \oplus G_{10}(x_{10}), x_1) = \mathsf{true}$, 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 $\mathbf{R}.\textsc{CheckFwd}(x_2 \oplus G_1(x_1), x_1, x_{10}) = \mathsf{true}$, 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 "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 $\mathbf{R}.\textsc{CheckBwd}(x_{10}, x_9 \oplus G_{10}(x_{10}), x_1) = \mathsf{true}$. 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 condition holds, the associated partial chain $(x_9, x_{10}, 9) \in \mathsf{CompletedChains}$. Assume that the claim has held 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 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 condition being true, then BadP occurred.

Suppose that there exists a triple T^* such that condition holds at line 27 of the simulator's execution while answering the distinguisher's t^{th} query. If condition held at the end of simulator's execution while answering the previous distinguisher query, then by 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 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's 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 5.25, 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 too $C^* \in \mathsf{CompletedChains}$ 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) $C = \mathsf{next}(C^*) \neq \perp$, $\mathsf{next}(C) \neq \perp$ when $Q_{\mathrm{enq}}.\textsc{Empty}() = \mathsf{true}$ in line 6 of the simulator's execution when answering the distinguisher's t^{th} query and (2) when it's not.

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 $Q_{\mathrm{enq}}.\textsc{Empty}() = \mathsf{true}$ 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 $Q_{\mathrm{enq}}.\textsc{Empty}() = \mathsf{true}$ in line 6. For the condition to be true, we need $x_1 \in G_1$ and $x_9 \in G_9$ and hence, we have that condition does not hold for the triple T^* when $Q_{\mathrm{enq}}.\textsc{Empty}() = \mathsf{true}$ 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. Firstly, C and D are not equivalent as $C \equiv C^*$ and no chain equivalent to C^* (including itself) has been enqueued. Secondly, for all chains D that have been enqueued in Q_{all}, $\mathsf{val}_1(D) \neq x_1$ when 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} $\mathsf{val}_1(D) = \perp$ just before the assignment that precedes the enqueueing of D by Claim 5.32. Since $\mathsf{BadlyHit}^+$ and 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 BadlyCollideP occurred. This cannot be a ForceVal assignment since that would contradict Claims 5.29, 5.30 or 5.31. 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 condition does not hold for the triple T^* at the end of the simulator's execution of the current query.

Consider 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 $Q_{\text{enq}}.\textsc{Empty}() = \mathsf{true}$ 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 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, it 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 that 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 too the condition 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 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 $\mathbf{D}$ 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 exists $x_5 \in G_5$, $x_6 \in G_6$ 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 $\mathbf{D}$'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, $x_5 \in G_5$, $x_6 \in G_6$ such that $(x_5, x_6, 5) \notin \mathsf{CompletedChains}$ happened during the simulator's execution while answering $\mathbf{D}$'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 $x_5 \in G_5$, $x_6 \in G_6$ such that $(x_5, x_6, 5) \notin \mathsf{CompletedChains}$ to be true, 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 assignment causing $x_5 \in G_5$, then either $x_6 \in G_6$ before $\mathbf{D}$'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 $\mathsf{val}_6(D) = x_6 \in G_6 \cup A^t_6$. And so, $(x_5, x_6, 5)$ is enqueued for completion by construction of 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 5.34. *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 is "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 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 5.33, 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 (line 27 of its execution) and the moment when a 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 5.35. *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 is "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, $C \in$ 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 5.35. Assume that the lemma has held until the moment that a 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 5.28.

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 $D \in$ CompletedChains by construction of the simulator. If C and D are equivalent chains such that $D \in$ CompletedChains, then $C \in$ CompletedChains by Claim 5.25.

Let us consider the case where no chain equivalent to C was dequeued before C was dequeued. Say $C \notin$ CompletedChains when dequeued. Note that 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 BadlyHit$^+$ occurred.

By Claim 5.26, 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 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 5.31 and 5.30 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 5.26, 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 5.36. *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 is "safe" for every chain C' in Q_{mid} dequeued so far. Then,*

- *At the moment C is dequeued, $C \in$ 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 that the lemma has held until the moment that a 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 Lemma 5.35 and Corollary 5.28.

Let us 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 $D \in$ CompletedChains by construction of the simulator. If C and D are equivalent chains such that $D \in$ CompletedChains, then $C \in$ CompletedChains by Claim 5.25.

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 Adapt for C since otherwise BadP or BadlyHit$^+$ occurs.

From Claim 5.34, 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 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 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 5.31 and 5.30 and 5.35. 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 5.30 as the current lemma has held thus far.

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

Lemma 5.37. *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, $C \in$ 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. Assume that the lemma has held until the moment that a 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 Lemmas 5.35, 5.36 and Corollary 5.28.

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 this scenario in the following two cases. Firstly, consider the case that when a chain C is enqueued in Q_{mid}, there is an equivalent chain D enqueued previously and $D \in$ CompletedChains when C is enqueued. If C equivalent to D and $D \in$ CompletedChains, then by Claim 5.25, we have that $C \in$ CompletedChains.

Consider the case that when a chain C is enqueued in Q_{mid}, no equivalent chain belongs to 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 5.24. 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 CompletedChains. So, by Claim 5.25, $C \in$ CompletedChains and hence, $C \in$ CompletedChains at the moment it is dequeued.

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 BadlyHit$^+$ occurs.

Since no chain equivalent to C was enqueued previously, from Claim 5.32, 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 BadlyHit$^+$ or 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 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 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 5.29.

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

Theorem 5.38 (No overwrites)**.** *In a good execution of H_2, for any call to* ForceVal$(x, \cdot, j)$ *we have $x \notin G_j$ before the call.*

Proof. Combining the result of Lemmas 5.35, 5.36 and 5.37 with Corollary 5.28, we have that for every call to ForceVal$(x, \cdot, j)$, $x \notin G_j$ before the call. □

As in [CHK+14], 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 5.39. *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 5.38 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 5.38, we have that the equivalence holds till $C \in$ CompletedChains and that 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 $\mathbf{R}$ explicit. The randomness p is a list containing $2 \cdot 2^{2n}$ strings of length $2n$. Then $\mathbf{R}$ runs deterministically given p. So, whenever the procedure $\mathbf{R}.\mathrm{P}(x_0, x_1)$ is queried, $\mathbf{R}$ checks if $(\downarrow, x_0, x_1) \in P$ and if so, answers accordingly. Otherwise, $\mathbf{R}$ reads $(x_{10}, x_{11}) := p(\downarrow, x_0, x_1)$ and $(\downarrow, x_0, x_1)$ as well as $(\uparrow, x_{10}, x_{11})$ are added to P, mapping to each other. The procedure $\mathbf{R}.\mathrm{P}^{-1}(x_{10}, x_{11})$ is implemented analogously.

Lemma 5.40. *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 $\mathbf{R}$.*

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 5.38.

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 $(x_5, x_6, 5) \in$ 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 $(x_5, x_6, 5) \notin$ 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. □

5.4 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 $\mathbf{D}$ interacts with $(\mathsf{Feistel}(z), \hat{\mathbf{S}}(z)^{\mathsf{Feistel}^+(z)})$. 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 $\hat{\mathbf{S}}$ is the same and the simulator answers queries to the round functions by running the procedure $\hat{\mathbf{S}}.\mathrm{F}(i,x)$ 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:

```
1  procedure P(x_0, x_1):
2      for i := 2 to 11 do
3          x_i := x_{i-2} ⊕ z(i-1, x_{i-1})
4      P(↓, x_0, x_1) := (x_10, x_11)
5      P(↑, x_10, x_11) := (x_0, x_1)
6      return (x_10, x_11)

7  procedure P^{-1}(x_10, x_11):
8      for i := 9 to 0 do
9          x_i := x_{i+2} ⊕ z(i+1, x_{i+1})
10     P(↓, x_0, x_1) := (x_10, x_11)
11     P(↑, x_10, x_11) := (x_0, x_1)
12     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 $\mathsf{Feistel}.\mathrm{P}^{-1}$.

```
1  procedure CheckFwd(x_0, x_1, x_10):
2      if (↓, x_0, x_1) ∈ P then
3          (x'_10, x'_11) := P(↓, x_0, x_1)
4          return x'_10 ?= x_10
5      return false

6  procedure CheckBwd(x_10, x_11, x_1):
7      if (↑, x_10, x_11) ∈ P then
8          (x'_0, x'_1) := P(↑, x_10, x_11)
9          return x'_1 ?= x_1
10     return false
```

5.4.1 Mapping Randomness of H_2 to Randomness of H_3

Before we describe the mapping, we make the randomness p used by $\mathbf{R}$ explicit. The randomness p is a list containing $2 \cdot 2^{2n}$ strings of length $2n$. Then $\mathbf{R}$ runs deterministically given p. 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 $\hat{\mathbf{S}}$ 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, p) where f is the randomness used by the simulator $\hat{\mathbf{S}}$ and p is the randomness used by the random two-sided function $\mathbf{R}$ either to the symbol λ in case (f, p) 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 [CHK$^+$14]. 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 $H_2(f, p)$ and $H_3(\tau(f, p))$ have "exactly the same behaviour" for good (f, p) (where a "good" (f, p) leads to a good execution of H_2). Whenever we refer to executions of $H_2(f, p)$ and $H_3(z)$ below, we assume that they are executed for the same distinguisher.

Definition 15. The function $\tau(f, p)$ is defined as follows: If (f, p) 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, p) is not good, let $\tau(f, p) := \lambda$.

Lemma 5.41. *The probability that a distinguisher* $\mathbf{D}$ *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 [CHK$^+$14, Lemma 3.37]. So, the probability that a distinguisher $\mathbf{D}$ 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 5.23, this is given by $O(q^{10})/2^n$. □

5.5 Equivalence 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 5.42. *Suppose that in* $H_3(z)$ *the simulator* $\hat{\mathbf{S}}(z)$ *eventually answers a query* $\mathrm{F}(i, x)$*. Then that query is answered with* $z(i, x)$*.*

Proof. The proof follows exactly as in [CHK$^+$14, Lemma 3.38]. □

Lemma 5.43 (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 [CHK$^+$14, Lemma 3.38]. □

References

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances*

in Cryptology—Crypto 2005, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005.

[CHK+14] Jean-Sebastien Coron, Thomas Holenstein, Robin Künzler, Yannick Seurin Jacques Patarin, and Stefano Tessaro. How to build an ideal cipher: The indifferentiability of the Feistel construction. *Journal of Cryptology*, 2014.

[CPS08] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In David Wagner, editor, *Advances in Cryptology—Crypto 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2008.

[DP06] Yevgeniy Dodis and Prashant Puniya. On the relation between the ideal cipher and the random oracle models. In Shai Halevi and Tal Rabin, editors, *3rd Theory of Cryptography Conference (TCC)*, volume 3876 of *Lecture Notes in Computer Science*, pages 184–206. Springer, March 2006.

[DP07] Yevgeniy Dodis and Prashant Puniya. Feistel networks made public, and applications. In Moni Naor, editor, *Advances in Cryptology—Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 534–554. Springer, May 2007.

[DS15] Yuanxi Dai and John P. Steinberger. Feistel networks: Indifferentiability at 10 rounds, 2015. Manuscript.

[EM93] 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, *Advances in Cryptology—Asiacrypt'91*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer, November 1993.

[Fei73] Horst Feistel. Cryptography and computer privacy. *Scientific American*, 228(5):15–23, 1973.

[GR04] Craig Gentry and Zulfikar Ramzan. Eliminating random permutation oracles in the Even-Mansour cipher. In Pil Joong Lee, editor, *Advances in Cryptology—Asiacrypt 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 32–47. Springer, December 2004.

[HKT11] 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 Annual ACM Symposium on Theory of Computing*, pages 89–98. ACM Press, June 2011.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[MPS12] Avradip Mandal, Jacques Patarin, and Yannick Seurin. On the public indifferentiability and correlation intractability of the 6-round Feistel construction. In Ronald Cramer, editor, *9th Theory of Cryptography Conference (TCC)*, volume 7194 of *Lecture Notes in Computer Science*, pages 285–302. Springer, March 2012.

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

[RR00] Zulfikar Ramzan and Leonid Reyzin. On the round security of symmetric-key cryptographic primitives. In Mihir Bellare, editor, *Advances in Cryptology—Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 376–393. Springer, 2000.

[Seu09] Yannick Seurin. *Primitives et Protocoles Cryptographiques à Sécurité Prouvée*. PhD thesis, Versailles University, 2009.

[YMO08] Kazuki Yoneyama, Satoshi Miyagawa, and Kazuo Ohta. Leaky random oracle. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *2nd International Conference on Provable Security*, volume 5324 of *Lecture Notes in Computer Science*, pages 226–240. Springer, 2008.