J Cryptol (2016) 29:61–114
DOI: 10.1007/s00145-014-9189-6

Journal of
CRYPTOLOGY

How to Build an Ideal Cipher: The Indifferentiability of the Feistel Construction*

Jean-Sébastien Coron
University of Luxembourg, Luxembourg, Luxembourg
jean-sebastien.coron@uni.lu

Thomas Holenstein · Robin Künzler
Department of Computer Science, ETH Zurich, 8092 Zurich, Switzerland
thomas.holenstein@inf.ethz.ch, robink@inf.ethz.ch

Jacques Patarin
University of Versailles-Saint-Quentin, Versailles, France
jacques.patarin@uvsq.fr

Yannick Seurin
ANSSI, Paris, France
yannick.seurin@m4x.org

Stefano Tessaro
Department of Computer Science, University of California,
Santa Barbara, Santa Barbara, CA, USA
tessaro@cs.ucsb.edu

Communicated by Phillip Rogaway.

Received 23 March 2012
Online publication 1 November 2014

Abstract. This paper provides the *first* provably secure construction of an invertible random permutation (and of an ideal cipher) from a *public* random function that can be evaluated by all parties in the system, including the adversary. The associated security goal was formalized via the notion of *indifferentiability* by Maurer et al. (TCC 2004). The problem is the natural extension of that of building (invertible) random permutations from (private) random functions, first solved by Luby and Rackoff (SIAM J Comput 17(2):373–386, 1988) via the four-round Feistel construction. As our main result, we prove that the Feistel construction with fourteen rounds is indifferentiable from an invertible random permutation. We also provide a new lower bound showing that five rounds are *not* sufficient to achieve indifferentiability. A major corollary of our result is the *equivalence* (in a well-defined sense) of the *random oracle model* and the *ideal cipher model*.

* The results of this paper were presented in [19] and [29].

Keywords. Random oracle model, Ideal cipher model, Feistel construction, Indifferentiability.

1. Introduction

1.1. *Random Oracles, Random Permutations, and Ideal Ciphers*

Hash-functions and block-ciphers are the fundamental building blocks of practical cryptography. A multitude of practical designs for these primitives have been developed and standardized over the years, and commodity algorithms like the SHA hash-function family and the Advanced Encryption Standard (AES) find nowadays ubiquitous usage. Following a well-established approach, we seek to rigorously prove security of applications using these primitives in the so-called *standard model*, i.e., via a reduction from a well-defined security assumption such as collision resistance of hash-functions or block-cipher pseudorandomness.

Unfortunately, this is not always possible: Many widely in-use cryptographic schemes elude standard-model security proofs under non-trivial assumptions, *despite a lack of attacks threatening their security.* In these cases, we often settle with proving security in an *ideal model* where invocations of the underlying primitive by the scheme are replaced by calls to a randomized oracle representing an *idealized* version of the primitive. This oracle is accessible by *all* parties, including the adversary. It is well known [6,17,39] that such proofs deliver solely a heuristic argument. Nevertheless, these security statements enjoy a natural interpretation in terms of lack of generic structural weaknesses of the scheme at hand, hence raising hope for secure instantiation via algorithms like SHA-3 and AES.

More concretely, following an approach formalized by Bellare and Rogaway [9] (but used even earlier by Fiat and Shamir [28]), a hash-function is idealized as a randomized function $\mathbf{R}$ called a *random oracle* mapping arbitrary bit strings to independent random n-bit digests. Work on the *random oracle model* (ROM) encompasses by now thousands of papers. Many widely employed practical schemes, including OAEP [10], FDH [9], PSS [11], as well as truly efficient pairing-based cryptography [3,7], only enjoy security proofs in the ROM.

A corresponding ideal model for block ciphers dates back to Shannon's work [46]. An *ideal cipher* $\mathbf{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ associates each κ-bit key k with an independent randomly chosen permutation $\mathbf{E}_k$ on the n-bit strings and allows for both *forward queries* $\mathbf{E}(k,x) = \mathbf{E}_k(x)$ and *backward queries* $\mathbf{E}^{-1}(k,y) = \mathbf{E}_k^{-1}(y)$ for all k, x, y. Again, application examples of the ideal cipher abound, and we only mention a few. They range from analyses of block-cipher-based hash-function constructions (e.g., in [14]), small-domain encryption [12], authenticated key agreement protocols [8], and block-cipher key extension [13,30], to studying generic related-key attacks [4]. If $\kappa = 0$, we call the ideal cipher a *random permutation*, and simply denote it by $\mathbf{P} : \{0,1\}^n \to \{0,1\}^n$. The *random permutation model* has been used in the analysis of hash [2,26,40,41] and block-cipher constructions [5,18,21,27,33].

1.2. *Building Ideal Primitives and Indifferentiability*

It is natural to ask whether one ideal primitive (and hence model) is stronger or weaker than another one. At first, an ideal cipher seems to provide a stronger structure and functionality than a random oracle, as first pointed out in [8]. But, most results in the ICM ended up having a ROM counterpart, indicating that this may not be true. Conversely, how to provide a provably sound instantiation of a random oracle using an ideal cipher is also not immediately clear. In summary, the following two questions appear natural:

> **Question 1.** *Can we find an efficient construction* $\mathbf{C}_1$ *invoking a random oracle* $\mathbf{R}$ *such that* $\mathbf{C}_1^{\mathbf{R}}$ *is "as good as" an ideal cipher* $\mathbf{E}$, *meaning that any secure application using* $\mathbf{E}$ *remains secure when using* $\mathbf{C}_1^{\mathbf{R}}$ *instead?*

> **Question 2.** *Conversely to Question 1, is there* $\mathbf{C}_2$ *such that* $\mathbf{C}_2^{\mathbf{E}}$ *is "as good as" a random oracle* $\mathbf{R}$*?*

While the first question was already asked informally in [8], formalizing *the* proper notion of being *"as good as"* has proved itself a difficult task. For instance, indistinguishability of $\mathbf{C}_1^{\mathbf{R}}$ and $\mathbf{E}$ is necessary but not sufficient, as the adversary can exploit access to the underlying primitive $\mathbf{R}$. For this reason, Maurer et al. [39] put forward the notion of *indifferentiability*: The system $\mathbf{C}_1^{\mathbf{R}}$ is indifferentiable from $\mathbf{E}$ if there exists a *simulator*[1] $\mathbf{S}$ accessing $\mathbf{E}$ such that $(\mathbf{C}_1^{\mathbf{R}}, \mathbf{R})$ and $(\mathbf{E}, \mathbf{S}^{\mathbf{E}})$ are indistinguishable. The definition of $\mathbf{C}_2^{\mathbf{E}}$ being indifferentiable from $\mathbf{R}$ is analogous, and the notion in fact generalizes to arbitrary primitives.

As shown in [39], indifferentiability is the "right" notion to instantiate the "as good as" relation when security of a scheme can be expressed in terms of the real/ideal world paradigm of simulation-based security frameworks such as [15,38]. However, with respect to more traditional game-based security definitions, Ristenpart et al. [42] later showed that indifferentiability is *not* sufficient to instantiate the "as good as" relation under a more general view including multistage security games, where the attacker is forced to forget some information during its attack, while still sufficient for *single*-stage security games. Unfortunately, as recently shown by Demay et al. [23], there does not seem to be any hope to positively answer the above two questions for arbitrary security games. For this reason, *this work focuses on indifferentiability*, the strongest notion for which it currently seems possible answering these questions. We in particular note that a positive answer to both questions implies that the random oracle and the ideal cipher models are *equivalent* with respect to most security games, namely all single-stage ones.

When focusing on indifferentiability, **Question 2** is well understood and has given rise to an impactful line of research: Coron et al. [16], and long series of subsequent works, have presented several constructions of random oracles from ideal ciphers leveraging hash-function designs such as the *Merkle-Damgård* construction [22,36] as well as block-cipher-based compression functions. As a consequence of these results, indifferentiability has become a de facto standard security requirement for hash-function constructions. In a similar vein, answering **Question 1** could provide new approaches to designing block ciphers from non-invertible primitives. But in contrast, the problem

[1] Usually required to be efficient, i.e., with running time polynomial in the number of queries it processes.

appears more challenging and has remained unsolved to date. This work fills the gap by answering this question in the affirmative.

1.3. *Our Main Result: Ideal Ciphers Via the Feistel Construction*

The main result of this paper is the first positive answer to **Question 1** above:

> **Main Result (Informal)** There exists an efficient construction $\mathbf{C}$ such that $\mathbf{C}^{\mathbf{R}}$ for a random oracle $\mathbf{R}$ is indifferentiable from an ideal cipher $\mathbf{E}$.

Our approach relies on the *r-round Feistel construction* Ψ_r, which implements a permutation with a $2n$-bit input (x_0, x_1) (where x_0, x_1 are n-bit strings), and a $2n$-bit output (x_r, x_{r+1}), such that for $i = 1, \ldots, r$, the i-th round computes

$$x_{i+1} := x_{i-1} \oplus \mathbf{F}_i(x_i) \ ,$$

where $\mathbf{F}_1, \ldots, \mathbf{F}_r : \{0,1\}^n \to \{0,1\}^n$ are the so-called *round functions*. Luby and Rackoff [32] first proved that if the round functions are independent random functions, then Ψ_3 is *information theoretically* indistinguishable from a random permutation that does not allow backward queries, whereas Ψ_4 is indistinguishable from a full-fledged random permutation. *Can we expect a similar statement to be true for indifferentiability?* More concretely, is it possible to prove that if the round functions are independent random functions, Ψ_r is indifferentiable from a random permutation for some $r \geq 4$?[2] Concretely, with $\mathbf{F} = (\mathbf{F}_1, \ldots, \mathbf{F}_r)$ being r independent random functions, and $\mathbf{P}$ being a random invertible permutation, we seek for an efficient simulator $\mathbf{S}$ such that $(\Psi_r^{\mathbf{F}}, \mathbf{F})$ and $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ are indistinguishable for all distinguishers making overall a polynomial number of queries to the given systems.

This suffices to build an ideal cipher form a random oracle $\mathbf{R}$: For each value k of the ideal cipher key, one implements the r independent random round functions from the random oracle $\mathbf{R}$ by enforcing domain separation, e.g., letting $\mathbf{F}_{k,i}(x) = \mathbf{R}(k, \langle i \rangle, x)$, where $\langle i \rangle$ is the $\lceil \log r \rceil$-bit encoding of $i \in \{1, \ldots, r\}$. The ideal cipher with key k is implemented by using the Feistel construction with the round functions $\mathbf{F}_{k,1}, \ldots, \mathbf{F}_{k,r}$.

Dodis and Puniya [24] were the first to study indifferentiability of the Feistel construction. They showed that $\omega(\log n)$ rounds of the Feistel construction are sufficient in the *honest-but-curious model* of indifferentiability, where the adversary only gets to *see* queries made by the construction to the round functions, but is *not* allowed to issue chosen queries. In the same work, while no positive results for full indifferentiability where shown, it was first noted that four rounds are insufficient.

Our Contributions This paper provides the first positive result showing indifferentiability of the Feistel construction with a sufficiently large number of rounds, providing in particular both upper and lower bounds on the number of rounds necessary for indifferentiability:

[2] Note that in contrast to the case of indistinguishability considered by Luby and Rackoff, we cannot construct a *non-invertible* random permutation from a random oracle, regardless of the number of rounds. This follows from a well-known result by Rudich [43] and Kahn et al. [31].

- First, we start by providing a new lower bound on the number of rounds that are necessary in order to ensure indifferentiability of the Feistel construction: Specifically, we prove in Sect. 2 that the five-round Feistel construction Ψ_5 is *not* indifferentiable from a random permutation, therefore showing that *at least* six rounds are necessary.
- Our main contribution is then given in Sect. 3: We prove that the fourteen-round Feistel construction Ψ_{14} is indifferentiable from a random permutation. In terms of concrete parameters, if Ψ_{14} implements a permutation on $2n$ bit strings, whenever interacting with a distinguisher making q queries overall, the simulator makes at most $1400q^8$ queries and runs in time $O(q^8)$. The distinguishing advantage is at most $\frac{10^8 \cdot q^{16}}{2^{2n}} + \frac{10^{22} \cdot q^{10}}{2^n}$.

Somewhat surprisingly, our result does *not* improve the work of [24] on honest-but-curious indifferentiability: In particular, we show that this notion is *not* implied by full indifferentiability in Appendix 1.

The remainder of the introduction provides a high-level outline of the techniques behind our contributions, as well as a specification of the formal model and notation used throughout the paper. Before turning to these, however, we find it appropriate to provide some background on the time line behind the results that constitute the contents of this paper and on related results.

Further Background Establishing our main result is the outcome of an intricate line of works whose end result is summarized by the present paper. Coron et al. [19] presented a first proof that the six-round Feistel construction Ψ_6 with independent random round functions is indifferentiable from a random permutation. Seurin [44] also presented a somewhat simpler indifferentiability proof for the ten-round Feistel construction Ψ_{10}. Upon publication of these works, the stated equivalence of the random oracle and ideal cipher models has been used for example to infer security in the random oracle model using an ideal cipher (or random permutation) as an intermediate step [25] and to prove impossibility of black box constructions from block ciphers [34].

Holenstein et al. [29] then gave a distinguishing attack showing that the proof of [20] (the full version of [19]) is *not* correct: For the simulator given in the proof, they exhibit an attacker that distinguishes with overwhelming advantage. A further stronger attack appears to succeed against a large class of simulators following the natural approach of [20], suggesting that it may be difficult to give a proof for six rounds. Later, Seurin [45] found a distinguishing attack showing that the proof of [44] is also *not* correct.

The main contribution of Holenstein et al. [29] was a proof that the fourteen-round Feistel construction is indifferentiable from a random permutation. The proof was partially based on [19], but used several new ideas. More precisely, the simulator is similar to the one in [44], and for bounding the simulator's running time, the elegant idea in [19] is used.

This paper includes both the negative result for five rounds from [19], and the positive result for fourteen rounds from [29].

We also note that Mandal et al. [37] showed that the six-round Feistel construction satisfies the weaker notion of *public* indifferentiability from a random permutation. Moreover, Lampe and Seurin [33] have adapted the techniques of the present paper to provide a construction of an ideal cipher from a small number of random permutations using a generalization of the so-called Even-Mansour construction [27]. (An alternative way to use the Even-Mansour construction for the same goal has been analyzed by Andreeva et al. [1].)

1.4. *Technical Overview*

1.4.1. *Five Rounds are Not Enough*

Let us first briefly address why the five-round Feistel construction is *not* indifferentiable from a random permutation. At the high level, we leverage the fact that it is possible—given oracle access to the round functions—to efficiently find inputs to the construction that, together with the resulting outputs, satisfy some relation that is hard to satisfy for a random permutation. More precisely, we show how a distinguisher can find four inputs (x_0, x_1), (x_0', x_1'), (x_0'', x_1''), and (x_0''', x_1''') with corresponding outputs (x_5, x_6), (x_5', x_6'), (x_5'', x_6''), and (x_5''', x_6'''), satisfying the following two relations:

$$x_1 \oplus x_1' \oplus x_1'' \oplus x_1''' = 0\ , \quad x_5 \oplus x_5' \oplus x_5'' \oplus x_5''' = 0.$$

Since finding such inputs for a random permutation is hard, any efficient simulator will necessarily fail to return consistent answers to the distinguisher.

1.4.2. *Indifferentiability of the 14-Round Feistel Construction*

We now discuss the techniques behind our main result. To this end, we first discuss the basic approach to *proving* that the r-round Feistel construction Ψ_r, for a sufficiently large number of rounds $r \geq 6$, is indifferentiable from a random permutation, and discuss our concrete instantiation of this approach. For reference, an illustration of the Feistel construction is provided on Page 11.

Recall that our task is to devise a simulator $\mathbf{S}$ that uses a given random permutation $\mathbf{P} : \{0,1\}^{2n} \to \{0,1\}^{2n}$ (accepting both forward and backward queries) to simulate r independent functions $\mathbf{F}_1, \ldots, \mathbf{F}_r$ so that $\mathbf{P}$ is consistent with Ψ_r using these simulated round functions. Concretely, when asked to evaluate $\mathbf{F}_i$ on input x_i, the simulator needs to set the value $\mathbf{F}_i(x_i)$ to some value y_i, but could in fact already have set this value proactively when answering an earlier query.

Our Strategy: Simulation Via Chain Completion To convey the main idea, suppose that a distinguisher queries the *simulated* round functions to evaluate Ψ_r on input $(x_0, x_1) \in \{0,1\}^{2n}$ obtaining the resulting output (x_r, x_{r+1}) by computing $x_{i+1} = x_{i-1} \oplus \mathbf{F}_i(x_i)$ for all $i = 1, \ldots, r$. Then, (x_r, x_{r+1}) *must equal* the output of $\mathbf{P}$ on input (x_0, x_1), for otherwise the distinguisher could easily detect it is not interacting with the real world.[3] To this end, the simulator needs to *recognize* that the queries $x_1, \ldots, x_r$ belong to an evaluation of Ψ_r, and to set the values $\mathbf{F}_i(x_i)$ to enforce consistency with $\mathbf{P}$. In the following, a sequence of values $x_1, \ldots, x_r$ such that $\mathbf{F}_i(x_i)$ is defined by the simulator for

[3] Of course, much more is needed, as this is only one specific distinguisher. But it will be convenient right now to restrict ourselves to thwarting this type of distinguishing attacks.

all $i = 1, \ldots, r$, and such that $x_{i+1} = x_{i-1} \oplus \mathbf{F}_i(x_i)$ for all $i = 2, \ldots, r-1$, will be called a *chain*. *Partial* chains, corresponding to a contiguous subsequence of a chain, are defined analogously. In addition, such partial chains may also "wrap around": For example, the sequence (x_1, x_2, x_{r-1}, x_r) constitutes a partial chain in case $\mathbf{P}(x_0, x_1) = (x_r, x_{r+1})$, where $x_0 = x_2 \oplus \mathbf{F}_1(x_1)$ and $x_{r+1} = x_{r-1} \oplus \mathbf{F}_r(x_r)$. Also, a length-two partial chain (x_i, x_{i+1}) corresponds simply to two values for which $\mathbf{F}_i(x_i)$ and $\mathbf{F}_{i+1}(x_{i+1})$ have been defined by the simulator. Note that any given chain of length at least two allows to evaluate forward and backward w.r.t. the Feistel construction. We stress that whether a sequence is a partial chain or not is a property that depends on the values of the round functions which have been defined by the simulator so far.

Our simulation strategy will consider a carefully chosen set of relevant partial chains (i.e., not all types of partial chains will be detected). Upon a query for $\mathbf{F}_i$ with input x_i, the simulator sets $\mathbf{F}_i(x_i)$ to a fresh random value and looks for new relevant partial chains involving x_i, adding them to a FIFO queue. (There may be many new partial chains!) Then, the simulator repeats the following, until the queue is empty: It removes the first partial chain from the queue and *completes* it to a (full) chain $x_1, x_2, \ldots, x_r$ such that $\mathbf{P}(x_0, x_1) = (x_r, x_{r+1})$, where $x_0 = \mathbf{F}_1(x_1) \oplus x_2$ and $x_{r+1} = \mathbf{F}_r(x_r) \oplus x_{r-1}$. In particular, is sets each undefined $\mathbf{F}_i(x_i)$ to a fresh uniform random string, with the exception of two consecutive values $\mathbf{F}_\ell(x_\ell)$ and $\mathbf{F}_{\ell+1}(x_{\ell+1})$ set adaptively for consistency. We refer to this step as *adapting* the values of $\mathbf{F}_\ell(x_\ell)$ and $\mathbf{F}_{\ell+1}(x_{\ell+1})$ and showing that such adapting is always possible (for some well chosen ℓ) will be a major challenge of our analysis below. For example, the simulator could complete the partial chain (x_7, x_8) as follows. First, it evaluates backward to obtain (x_0, x_1), setting each undefined $\mathbf{F}_i(x_i)$ to a fresh uniform random string. It then sets $(x_r, x_{r+1}) := \mathbf{P}(x_0, x_1)$ and continues to evaluate backward and forward (again setting undefined values to fresh random strings), until only $\mathbf{F}_{10}(x_{10})$ and $\mathbf{F}_{11}(x_{11})$ are undefined. These two values are then defined as $\mathbf{F}_{10}(x_{10}) := x_9 \oplus x_{11}$, and $\mathbf{F}_{11}(x_{11}) := x_{10} \oplus x_{12}$. However, note that within the process, new values $\mathbf{F}_j(x_j)$ are defined, which may result in new chains being detected and added to the queue. When the queue is finally empty, the simulator returns $\mathbf{F}_i(x_i)$.

We now face three main challenges, and our choice of which partial chains are relevant and how they are completed will be crucial in order to solve them:

(1) **Efficiency** We need to show that the simulation terminates with high probability when answering a query, i.e., early enough the queue becomes empty.

(2) **Consistency** We need to show that the values $\mathbf{F}_\ell(x_\ell)$ and $\mathbf{F}_{\ell+1}(x_{\ell+1})$ which are adapted to ensure consistency are always undefined whenever a chain is completed.

(3) **Indistinguishability** Even if the above two points are successfully shown, it is still necessary to show that the simulated world cannot be distinguished from the real world.

Our Instantiation We will fix $r = 14$. As indicated in the illustration on Page 11, upon a query to $\mathbf{F}_2$ or $\mathbf{F}_{13}$, the simulator will detect partial chains of the form $(x_1, x_2, x_{13}, x_{14})$, while upon a query to $\mathbf{F}_7$ or $\mathbf{F}_8$, it will detect partial chains of the form (x_7, x_8) (we henceforth refer to the two subsets of rounds $\{1, 2, 13, 14\}$ and $\{7, 8\}$ as *detect zones*). The simulator always adapts either $\mathbf{F}_4(x_4)$ and $\mathbf{F}_5(x_5)$, or $\mathbf{F}_{10}(x_{10})$ and $\mathbf{F}_{11}(x_{11})$, depending on the round function queried when the chain is first detected. (We refer to $\{4, 5\}$ and

$\{10, 11\}$ as *adapt zones*.) In particular, note that function values in rounds 3,6,9,12 (which are called the *buffer rounds*) are always set to uniform random values when completing. Concretely, upon a query to $\mathbf{F}_2$ or $\mathbf{F}_7$, in case one or more partial chains $(x_1, x_2, x_{13}, x_{14})$ or (x_7, x_8) are detected, they will be completed using the adapt zone $\{4, 5\}$. Symmetrically, chains detected upon queries to $\mathbf{F}_{13}$ or $\mathbf{F}_8$ are completed using the adapt zone $\{10, 11\}$. Let us now elaborate shortly on how the above three challenges are addressed.

Addressing Challenge 1 We will show that the recursion stops after at most poly(q) steps, where q is the overall number of queries of the distinguisher. To this end, we rely on the observation that unless some unlikely collision occurs, each detected partial chain $(x_1, x_2, x_{13}, x_{14})$ is associated with an earlier $\mathbf{P}$ query by the distinguisher (either a forward query on input (x_0, x_1) or a backward query on input (x_{14}, x_{15})); hence, at most q, such chains will ever be detected and completed. Furthermore, the number of values $\mathbf{F}_7(x_7)$ that are defined by the simulator is at most $2q$: either the distinguisher queries $\mathbf{F}_7(x_7)$ directly, or the value is defined when completing a chain detected in zone $\{1, 2, 13, 14\}$. As the same argument holds for $\mathbf{F}_8$, we get that the total number of chains that are completed is upper bounded by $q + (2q)^2$.

Addressing Challenge 2 As an illustrative example, suppose that upon setting $\mathbf{F}_2(x_2)$ uniformly at random, a new chain $C = (x_1, x_2, x_{13}, x_{14})$ is detected and enqueued. The simulator will eventually complete C into a chain $(x_1, x_2, \ldots, x_{14})$ using the adapt zone $\{4, 5\}$, i.e., it sets $x_i := \mathbf{F}_{i+1}(x_{i+1}) \oplus x_{i+2}$ for all $i = 12, 11, \ldots, 5, 4$, and $x_3 := \mathbf{F}_2(x_2) \oplus x_1$, where all undefined values $\mathbf{F}_i(x_i)$ for $i \notin \{4, 5\}$ are set uniformly at random. Finally, if possible, it sets $\mathbf{F}_i(x_i) := x_{i-1} \oplus x_{i+1}$ for $i = 4, 5$.

In order for the final step to be possible, our hope is that $\mathbf{F}_3(x_3)$ and $\mathbf{F}_6(x_6)$ are unset when C is dequeued, and are set to uniform random values at completion. By doing so, x_4 and x_5 also become fresh random values, and hence $\mathbf{F}_4(x_4)$ and $\mathbf{F}_5(x_5)$ are unset with high probability. The proof that this hope is true is one of our main technical contributions. We now illustrate the issue for the case of $\mathbf{F}_3(x_3)$.

First, note that when setting $\mathbf{F}_2(x_2)$, the value $x_3 := x_1 \oplus \mathbf{F}_2(x_2)$ is fresh and random, and thus with very high probability, at this point, $\mathbf{F}_3(x_3)$ is unset. However, it may be that many other values are defined *after* C is detected and *before* C is completed when completing other chains: First, C may be detected during the completion of some other chain, and second, several partial chains containing x_2 may be enqueued just before C. The crucial observation is that by using a FIFO queue, *every partial chain C' completed in between either shares the same x_2 and is added together with C, or was already in the queue when defining* $\mathbf{F}_2(x_2)$.

For all partial chains $C' = (x'_1, x_2, x'_{13}, x'_{14})$ of the former type, they must have $x'_1 \neq x_1$, and hence $x'_3 = x'_1 \oplus \mathbf{F}_2(x_2) \neq x_3$. Moreover, for those C' which were already in the queue, if they are of the form $C' = (x'_1, x'_2, x'_{13}, x'_{14})$ for $x'_2 \neq x_2$, then we must have $x'_1 \oplus \mathbf{F}_2(x'_2) \neq x_3$ with very high probability because x_3 is fresh and random. Therefore, we are left with proving that completing chains (x'_7, x'_8) which are already in the queue cannot set $\mathbf{F}_3(x_3)$, except with negligible probability, which is the hardest part of our analysis.

Addressing Challenge 3 To see why proving indistinguishability can be difficult, consider a distinguisher which in the ideal world first queries the given permutation $\mathbf{P}(x_0, x_1)$, giving values (x_{14}, x_{15}). The distinguisher then checks (say) the first bit of x_{14}, and depending on it, starts querying the simulator to evaluate the Feistel construction from the top with the input values (x_0, x_1), or from the bottom with values (x_{14}, x_{15}). Inspection of our simulator reveals that the choice of the adapt zone of the simulator then depends on the first bit of x_{14}.

The problem which now comes in is that the randomness inherent in (x_{14}, x_{15}) is needed in order to show that the values of $\mathbf{F}$ in the adapt zones look random. However, conditioned on using the upper adapt zone, one bit of x_{14} is already fixed.

In order to solve this problem, we take the following, very explicit approach: we consider the two experiments which we want to show to behave almost the same and define a map associating randomness in one experiment to randomness in the other experiment. We then study this map. This leads to a fine-grained understanding and a formal treatment of the indistinguishability proof.

1.5. *Model and Notational Conventions*

The results throughout this paper are information-theoretic and consider random experiments where a *distinguisher* $\mathbf{D}$ interacts with some given system $\mathbf{T}$, outputting a value $\mathbf{D}(\mathbf{T})$. In the context of this paper, such systems consist of the composition $\mathbf{T} = (\mathbf{T}_1, \mathbf{T}_2)$ of two (generally correlated) systems accessible in parallel, where $\mathbf{T}_i$ is either a random primitive (such as a random function $\mathbf{F}$, a random permutation $\mathbf{P}$ defined above), or a construction $\mathbf{C}^{\mathbf{T}}$ accessing the random primitive $\mathbf{T}$. The advantage $\Delta^{\mathbf{D}}(\mathbf{T}, \mathbf{T}')$ of a distinguisher $\mathbf{D}$ in distinguishing two systems $\mathbf{T}$ and $\mathbf{T}'$ is defined as the absolute difference $\big|\Pr[\mathbf{D}(\mathbf{T}) = 1] - \Pr[\mathbf{D}(\mathbf{T}') = 1]\big|$.

We dispense to the largest extent with a formal definition of such systems (cf. e.g., the framework of Maurer [35] for a formal treatment). Most systems we consider will be defined formally using pseudocode in a RAM model of computation, following the approach of [13,47]. The time complexity of a system/distinguisher is also measured with respect to such a model.

Defining indifferentiability is somewhat subtle, as different definitions [16,39] are used in the literature. In particular, it will be convenient to use the following definition:

Definition 1.1. For a construction $\mathbf{C}$ accessing independent random functions $\mathbf{F} = (\mathbf{F}_1, \ldots, \mathbf{F}_r)$,[4] we say that $\mathbf{C}^{\mathbf{F}}$ is *indifferentiable* from a random permutation $\mathbf{P}$ if there exists a simulator $\mathbf{S}$ such that for all polynomially bounded q, the advantage $\Delta^{\mathbf{D}}((\mathbf{C}^{\mathbf{F}}, \mathbf{F}), (\mathbf{P}, \mathbf{S}^{\mathbf{P}}))$ is negligible for all distinguishers $\mathbf{D}$ issuing a total of at most q queries to the two given systems, and furthermore, there exists a fixed polynomial $p(q)$, such that $\mathbf{S}$ runs in time $p(q)$ except with negligible probability.

Our definition allows exponential worst-case running time of the simulator. However, given an upper bound on the number q of overall distinguisher queries, the simulator

[4] Such a tuple can also be seen as a random primitive.

can be made to run in polynomial time by aborting after $p(q)$ steps. This modification makes the simulator distinguisher dependent, since the simulator cannot see (and count) queries to **P**, and consequently cannot determine q. But we point out that this dependence is quite weak: The simulator only depends on the number of queries the distinguisher makes. In other words, our definition implies the original one in [39], but does not imply the stronger one of [16], which requires a universal simulator.

2. The Five-Round Feistel Construction is Not Sufficient

In this section, we prove the following theorem.

Theorem 2.1. *The five-round Feistel construction using five independent random functions is not indifferentiable from a random permutation.*

For this, we construct a polynomial-time distinguisher **D** which, for any polynomial-time simulator **S**, distinguishes with overwhelming advantage $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ from $(\Psi^{\mathbf{F}}, \mathbf{F})$, where **P** is a random permutation, Ψ is a five-round Feistel construction, and **F** is a collection of five uniform random functions. Toward this end, we first show the following lemma.

Lemma 2.2. *Let* $\mathbf{P} : \{0,1\}^{2n} \to \{0,1\}^{2n}$ *be a random permutation, and consider a system issuing at most q queries to* **P**. *Denote generically* (x_0^i, x_1^i), (x_5^i, x_6^i) *the input/output of* **P** *corresponding to the i-th query (independently of whether this is a query to* **P** *or* $\mathbf{P}^{-1}$*). Then, assuming* $q \le 2^{2n-1}$, *the probability that there exist four queries* i_1, i_2, i_3 *and* i_4 *with pairwise different input values such that*

$$\begin{cases} x_1^{i_1} \oplus x_1^{i_2} \oplus x_1^{i_3} \oplus x_1^{i_4} = 0 \\ x_5^{i_1} \oplus x_5^{i_2} \oplus x_5^{i_3} \oplus x_5^{i_4} = 0 \end{cases}$$

is less than $q^4/2^n$.

Proof. Denote by Bad the event that such queries exist among all q queries, and by Bad_i the event that such queries exist among the first i queries. We will upper bound $\Pr[\mathsf{Bad}_i | \overline{\mathsf{Bad}}_{i-1}]$. Consider the i-th query, and assume *wlog* that it is a query to **P**. Then, Bad_i happens only if the input is different from all previous inputs and x_5^i hits one of at most $\binom{i-1}{3} \le i^3$ values, hence with probability less than $2^n i^3/(2^{2n} - (i-1)) \le 2i^3/2^n$ (using $q \le 2^{2n-1}$). The result follows by summing over i and using $\sum_{i=1}^{q} i^3 \le q^4/2$. □

The distinguisher **D** interacts with a system $\Sigma = (P, F)$ which is either $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ or $(\Psi^{\mathbf{F}}, \mathbf{F})$. It proceeds as follows (the attack is depicted in Fig. 1):

1. Choose arbitrary values x_3, x_3', x_4 that are pairwise different.
2. Compute $x_2 = x_4 \oplus F_3(x_3)$ and $x_2' = x_4 \oplus F_3(x_3')$.

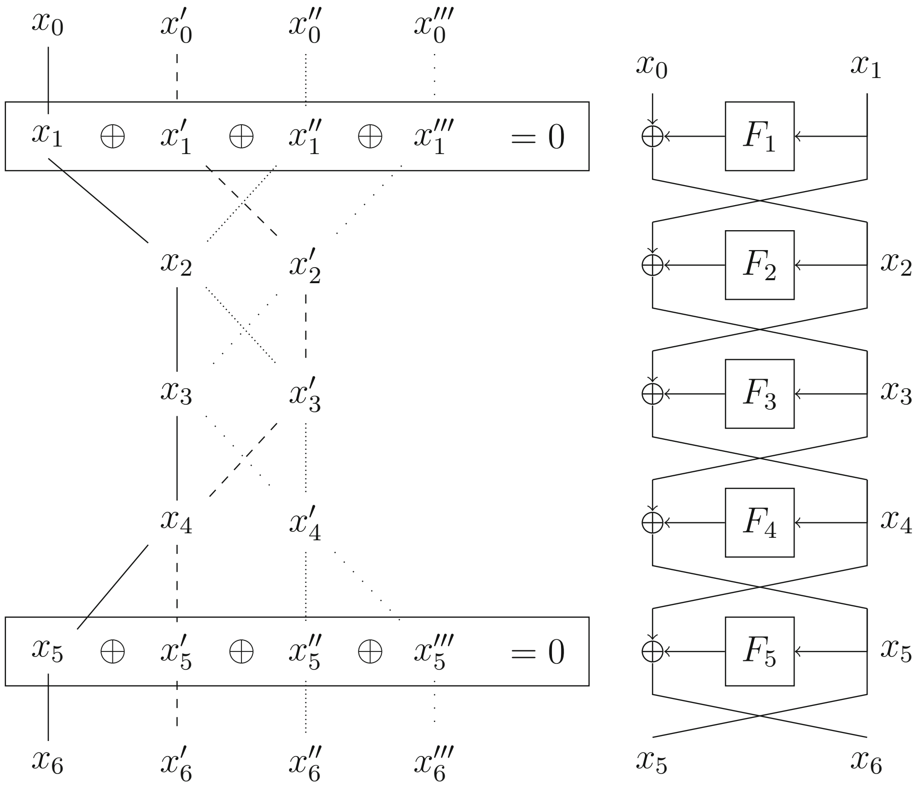
x_0 x'_0 x''_0 x'''_0
$x_1 \oplus x'_1 \oplus x''_1 \oplus x'''_1 = 0$
x_2 x'_2
x_3 x'_3
x_4 x'_4
$x_5 \oplus x'_5 \oplus x''_5 \oplus x'''_5 = 0$
x_6 x'_6 x''_6 x'''_6
x_0 x_1
F_1
F_2 x_2
F_3 x_3
F_4 x_4
F_5 x_5
x_5 x_6

Fig. 1. The five-round distinguishing attack. The *lines* with four distinct patterns on the *left side* represent the computation paths in the Feistel construction for each input/output (x_0^i, x_1^i), (x_5^i, x_6^i) involved in the attack .

3. Compute

$$\begin{cases} x_1 = x_3 \oplus F_2(x_2), & x_0 = x_2 \oplus F_1(x_1) \\ x'_1 = x'_3 \oplus F_2(x'_2), & x'_0 = x'_2 \oplus F_1(x'_1) \\ x''_1 = x'_3 \oplus F_2(x_2), & x''_0 = x_2 \oplus F_1(x''_1) \\ x'''_1 = x_3 \oplus F_2(x'_2), & x'''_0 = x'_2 \oplus F_1(x'''_1) \end{cases}$$

4. If x_1, x'_1, x''_1, x'''_1 are *not* pairwise different, then return 0.
5. Query $(x_5, x_6) = P(x_0, x_1)$, $(x'_5, x'_6) = P(x'_0, x'_1)$, $(x''_5, x''_6) = P(x''_0, x''_1)$, and $(x'''_5, x'''_6) = P(x'''_0, x'''_1)$.
6. If $x_5 \oplus x'_5 \oplus x''_5 \oplus x'''_5 = 0$ then return 1, else return 0.

We have the following lemma, from which Theorem 2.1 is a simple consequence.

Lemma 2.3. *For any polynomial-time simulator* $\mathbf{S}$*, there is a negligible function* ν *such that the advantage* $\Delta^{\mathbf{D}}((\Psi^{\mathbf{F}}, \mathbf{F}), (\mathbf{P}, \mathbf{S}^{\mathbf{P}}))$ *is greater that* $1 - \nu$.

Proof. We first show that $\mathbf{D}$ outputs 1 with overwhelming probability when interacting with $(\Psi^{\mathbf{F}}, \mathbf{F})$. Since $x_3 \neq x_3'$ by definition, the probability that $\mathbf{F}_3(x_3) \neq \mathbf{F}_3(x_3')$ is $1 - 1/2^n$. If this inequality holds, we have $x_2 \neq x_2'$. This in turn implies that $\mathbf{F}_2(x_2) \neq \mathbf{F}_2(x_2')$ and $x_1 \neq x_1'$ and $x_1'' \neq x_1'''$ with probability at least $1 - 3/2^n$. Thus, with probability at least $1 - 4/2^n$, we have that x_1, x_1', x_1'', x_1''' are pairwise different, and thus step 4 does not return. Denote $x_4' = x_2 \oplus F_3(x_3') = x_2' \oplus F_3(x_3)$ (the last two values are equal by definition of x_2 and x_2'). Then, computing the Feistel forward, one has:

$$\begin{aligned} x_5 &= x_3 \oplus F_4(x_4) \\ x_5' &= x_3' \oplus F_4(x_4) \\ x_5'' &= x_3' \oplus F_4(x_4') \\ x_5''' &= x_3 \oplus F_4(x_4') \end{aligned}$$

Hence, the equality $x_5 \oplus x_5' \oplus x_5'' \oplus x_5''' = 0$ is always satisfied.

We show that $\mathbf{D}$ outputs 0 with overwhelming probability when interacting with $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$. We may assume that x_1, x_1', x_1'', x_1''' are pairwise different, as otherwise $\mathbf{D}$ outputs 0 in step 4. Note that by construction of the distinguisher, the equality $x_1 \oplus x_1' \oplus x_1'' \oplus x_1''' = 0$ is always satisfied. Consider the union of $\mathbf{D}$ and $\mathbf{S}$ as a single system interacting with the random permutation $\mathbf{P}$, and let q be an upper bound on the total number of queries issued by this system to $\mathbf{P}$. Clearly, q is polynomial if $\mathbf{S}$ is polynomial time. Then, by Lemma 2.2, the probability that $x_5 \oplus x_5' \oplus x_5'' \oplus x_5''' = 0$ is less than $q^4/2^n$, which is negligible. The result follows. □

3. Indifferentiability of the Fourteen-Round Feistel Construction

We now turn to the main result of this paper. We prove that the fourteen-round Feistel construction is indifferentiable from a random permutation, as summarized by the following theorem.

Theorem 3.1. *The fourteen-round Feistel construction using fourteen independent random functions is indifferentiable from a random permutation.*

For a random permutation on $2n$ *bits and any distinguisher that issues at most* q *queries, except with probability* $\frac{10^8 \cdot q^{16}}{2^{2n}}$, *the simulator makes at most* $1400q^8$ *queries and runs in time* $O(q^8)$. *The distinguishing advantage is at most* $\frac{10^8 \cdot q^{16}}{2^{2n}} + \frac{10^{22} \cdot q^{10}}{2^n}$.

Even though we state explicit bounds in the theorem, we have not tried to optimize them, and aim for a simple proof instead.

Moreover, we can extend this result to provide a construction of an ideal cipher from a random oracle, as we now explain. To implement the ideal cipher, we use a keyed version of the Feistel construction, which we define as follows. Given a random oracle $\mathbf{R} : \{0,1\}^* \to \{0,1\}^n$, for understood parameters κ and n, we define the r-round *keyed* Feistel construction $\widetilde{\Psi}_r = \widetilde{\Psi}_r^{\mathbf{R}}$ which, on inputs (k, x) for a forward query and (k, y) for a backward query (where $k \in \{0,1\}^\kappa$ and $x, y \in \{0,1\}^{2n}$) behaves as $\Psi_r^{\mathbf{F}^k}$ on a forward query x and on a backward query y, respectively, where $\mathbf{F}^k = (\mathbf{F}_1^k, \ldots, \mathbf{F}_r^k)$ are shorthands for $\mathbf{F}_i^k(x) = \mathbf{R}(\langle i \rangle \| k \| x)$. Here, $\langle i \rangle$ is the $\lceil \log r \rceil$-bit encoding of $i \in \{1, \ldots, r\}$.

The following theorem states the main result of this paper. It establishes the indifferentiability of the keyed Feistel construction from an ideal cipher. Given that the Feistel construction is indifferentiable from a random permutation (Theorem 3.1), the proof of this theorem is not difficult: it relies on standard techniques and is given in Appendix 2.

Theorem 3.2. *The fourteen-round keyed Feistel construction using a random oracle is indifferentiable from an ideal cipher. For an ideal cipher with* κ*-bit key and* $2n$*-bit inputs, and any distinguisher that issues at most* q *queries, except with probability* $\frac{10^8 \cdot q^{17}}{2^{2n}}$*, the simulator makes at most* $1400q^8$ *queries and runs in time* $O(q^8)$*. The distinguishing advantage is at most* $\frac{10^8 \cdot q^{17}}{2^{2n}} + \frac{10^{22} \cdot q^{11}}{2^n}$.

The remainder of this section is devoted to the proof of Theorem 3.1. Our task is to provide a simulator $\mathbf{S}$ with access to a random permutation $\mathbf{P}$ such that $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ is indistinguishable from $(\Psi^{\mathbf{F}}, \mathbf{F})$, where $\mathbf{F}$ denotes the random functions used in the Feistel construction.

We first define the simulator $\mathbf{S}$ in Sect. 3.1. Then, we transform $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ stepwise to $(\Psi^{\mathbf{F}}, \mathbf{F})$ to prove indistinguishability. The random functions we consider in this section are always from n bits to n bits, and the random permutation $\mathbf{P}$ is over $2n$ bits.

3.1. *Simulator Definition*

We first give a somewhat informal, but detailed description of the simulator. We then use pseudocode to specify the simulator in a more formal manner.

3.1.1. *Informal Description*

The simulator provides an interface $\mathbf{S}.\mathrm{F}(k, x)$ to query the simulated random function $\mathbf{F}_k$ on input x. For each k, the simulator internally maintains a table whose entries are pairs (x, y) of n-bit values. They denote pairs of inputs and outputs of $\mathbf{S}.\mathrm{F}(k, x)$. We denote these tables by $\mathbf{S}.G_k$ or just G_k when the context is clear. We write $x \in G_k$ whenever x is a preimage in this table, often identifying G_k with the set of preimages stored. When $x \in G_k$, $G_k(x)$ denotes the corresponding image.

On a query $\mathbf{S}.\mathrm{F}(k, x)$, the simulator first checks whether $x \in G_k$. If so, it answers with $G_k(x)$. Otherwise the simulator picks a random value y and inserts (x, y) into $G_k(x)$. After this, the simulator takes steps to ensure that its future answers are consistent with the permutation $\mathbf{P}$.

There are two cases in which the simulator performs a specific action for this. First, if $k \in \{2, 13\}$, the simulator considers all newly generated tuples $(x_1, x_2, x_{13}, x_{14}) \in G_1 \times G_2 \times G_{13} \times G_{14}$, and computes $x_0 := x_2 \oplus G_1(x_1)$ and $x_{15} := x_{13} \oplus G_{14}(x_{14})$. It then checks whether $\mathbf{P}(x_0, x_1) = (x_{14}, x_{15})$. Whenever the answer to such a check is positive, the simulator enqueues the detected values in a queue.[5] More precisely, it

[5] Recall that a queue is a first in first out data structure.

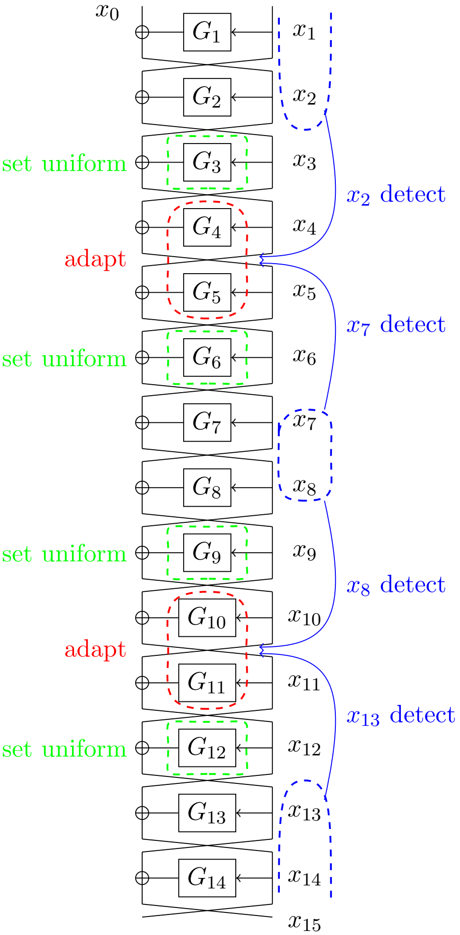

Fig. 2. The fourteen-round Feistel with the zones where our simulator detects chains and adapts them. Whenever a function value $G_2(x_2)$, $G_7(x_7)$, $G_8(x_8)$, or $G_{13}(x_{13})$ is defined, the simulator checks whether the values in the *dashed zones* x_7, x_8 and x_1, x_2, x_{13}, x_{14} (in the colored version these zones are *blue*) form a partial chain. In case a chain is detected, it is completed; the function values in the *dashed zones* x_4, x_5 or x_{10}, x_{11} (in the colored version these zones are *red*) are adapted in order to ensure consistency of the chain .

enqueues a four-tuple $(x_1, x_2, 1, \ell)$. The value 1 ensures that later the simulator knows that the first value x_1 corresponds to G_1. The value ℓ describes where to adapt values of G_ℓ to ensure consistency with the given permutation. If $k = 2$, then $\ell = 4$ and if $k = 13$ then $\ell = 10$. The second case is when $k \in \{7, 8\}$. Then, the simulator enqueues all newly generated pairs $(x_7, x_8) \in G_7 \times G_8$. It enqueues all these pairs into the queue as $(x_7, x_8, 7, \ell)$, where $\ell = 4$ if $k = 7$ and $\ell = 10$ if $k = 8$ (this is illustrated in Fig. 2). We call the tuples added to the queue *partial chains*.

The simulator then does the following, until the queue is empty. (When the queue is finally empty, the simulator returns the answer to the initial query.) It removes the first partial chain (x_i, x_{i+1}, i, ℓ) from the queue, and *completes it*. This means that we compute all values x_j by evaluating the Feistel construction (making at most one query to $\mathbf{P}$ or $\mathbf{P}^{-1}$), and setting all undefined $G_j(x_j)$ for $j \neq \{\ell, \ell+1\}$ to fresh random values. The simulator defines the remaining two values in such a way that consistency with $\mathbf{P}$ is ensured, i.e., $G_\ell(x_\ell) := x_{\ell-1} \oplus x_{\ell+1}$ and $G_{\ell+1}(x_{\ell+1}) := x_\ell \oplus x_{\ell+2}$. If a value for either of these is defined from a previous action of the simulator, the simulator overwrites the value (possibly making earlier chains inconsistent).

Whenever a new value $G_k(x_k)$ for $k \in \{2, 13\}$ is defined when completing a chain, the exact same checks as above are performed on the newly generated tuples $(x_1, x_2, x_{13}, x_{14})$, and a new partial chain can be enqueues. Whenever a value $G_k(x_k)$ for $k \in \{7, 8\}$ is defined, the simulator similarly enqueues all new pairs (x_7, x_8).

In order to make sure the simulator does not complete the same chains twice, the simulator additionally keeps a set CompletedChains that contains all triples (x_k, x_{k+1}, k) which have been completed previously. Whenever the simulator dequeues a chain, it only completes the chain if it is not in the set CompletedChains.

3.1.2. *The Simulator in Pseudocode*

We provide pseudocode to describe the simulator as explained above in full detail below. Later, during the analysis, we will consider a slightly different simulator $\mathbf{T}$. For this, we replace whole lines; the replacements are put into boxes next to these lines. The reader can ignore these replacements at the moment.

First, the simulator internally uses a queue and some hashtables to store the function values, and a set CompletedChains to remember the chains that have been completed already. The queue Q provides the procedure Q.Enqueue to add elements to the end of the queue. The procedure Q.Dequeue removes the element in the front of the queue and returns it.

The procedure F(i, x) provides the interface to a distinguisher. It first calls the corresponding internal procedure F^INNER^, which defines the value and fills the queue if necessary. Then, the procedure F(i, x) completes the chains in the queue that were not completed previously, until the queue is empty.

The procedure Adapt adapts the values. It first sets the values in the buffer rounds (namely rounds 3 and 6, or 9 and 12 in Fig. 2) uniformly at random. It then adapts the values of $G_\ell(x_\ell)$ and $G_{\ell+1}(x_{\ell+1})$ such that the chain matches the permutation. It would be possible to simplify the code by removing lines 20 to 25 below, and changing the parameters in lines 12 and 13 above. The current notation simplifies notation in the proof.

The procedure F^INNER^ provides the internal interface for evaluations of the simulated function. It only fills the queue, but does not empty it.

The procedure enqueueNewChains detects newly created chains and enqueues them. Sometimes, chains may be detected which have been completed before, but they are ignored when they are dequeued.

The helper procedures EvaluateForward and EvaluateBackward take indices k and ℓ and a pair (x_k, x_{k+1}) of input values for G_k and G_{k+1}, and either evaluate forward or backward in the Feistel to obtain the pair $(x_\ell, x_{\ell+1})$ of input values for G_ℓ and $G_{\ell+1}$.

```
System S:                                          $\boxed{\textbf{System T}(f)\text{:}}$
Variables:
    Queue Q
    Hashtable $G_1, \ldots, G_{14}$
    Set CompletedChains := $\emptyset$

public procedure F$(i, x)$
    F$^{\text{INNER}}(i, x)$
    while $\neg Q$.EMPTY() do
        $(x_k, x_{k+1}, k, \ell) := Q$.DEQUEUE()
        if $(x_k, x_{k+1}, k) \notin$ CompletedChains then   // ignore prev. completed chains
            // complete the chain
            $(x_{\ell-2}, x_{\ell-1}) :=$ EVALUATEFORWARD$(x_k, x_{k+1}, k, \ell - 2)$
            $(x_{\ell+2}, x_{\ell+3}) :=$ EVALUATEBACKWARD$(x_k, x_{k+1}, k, \ell + 2)$
            ADAPT$(x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell)$
            $(x_1, x_2) :=$ EVALUATEBACKWARD$(x_k, x_{k+1}, k, 1)$
            $(x_7, x_8) :=$ EVALUATEFORWARD$(x_1, x_2, 1, 7)$
            CompletedChains := CompletedChains $\cup \{(x_1, x_2, 1), (x_7, x_8, 7)\}$
    return $G_i(x)$

private procedure ADAPT$(x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell)$
    if $x_{\ell-1} \notin G_{\ell-1}$ then
        $G_{\ell-1}(x_{\ell-1}) \leftarrow_R \{0, 1\}^n$          $\boxed{G_{\ell-1}(x_{\ell-1}) := f(\ell - 1, x_{\ell-1})}$
    $x_\ell := x_{\ell-2} \oplus G_{\ell-1}(x_{\ell-1})$
    if $x_{\ell+2} \notin G_{\ell+2}$ then
        $G_{\ell+2}(x_{\ell+2}) \leftarrow_R \{0, 1\}^n$          $\boxed{G_{\ell+2}(x_{\ell+2}) := f(\ell + 2, x_{\ell+2})}$
    $x_{\ell+1} := x_{\ell+3} \oplus G_{\ell+2}(x_{\ell+2})$
    FORCEVAL$(x_\ell, x_{\ell+1} \oplus x_{\ell-1}, \ell)$
    FORCEVAL$(x_{\ell+1}, x_\ell \oplus x_{\ell+2}, \ell + 1)$

private procedure FORCEVAL$(x, y, \ell)$
    $G_\ell(x) := y$

private procedure F$^{\text{INNER}}(i, x)$:
    if $x \notin G_i$ then
        $G_i(x) \leftarrow_R \{0, 1\}^n$                  $\boxed{G_i(x) := f(i, x)}$
        if $i \in \{2, 7, 8, 13\}$ then
            ENQUEUENEWCHAINS$(i, x)$
    return $G_i(x)$

private procedure ENQUEUENEWCHAINS$(i, x)$:
    if $i = 2$ then
        forall $(x_1, x_2, x_{13}, x_{14}) \in G_1 \times \{x\} \times G_{13} \times G_{14}$ do
            if CHECK$(x_2 \oplus G_1(x_1), x_1, x_{14}, x_{13} \oplus G_{14}(x_{14}))$ then
                $Q$.ENQUEUE$(x_1, x_2, 1, 4)$
    else if $i = 13$ then
        forall $(x_1, x_2, x_{13}, x_{14}) \in G_1 \times G_2 \times \{x\} \times G_{14}$ do
            if $\textsc{Check}(x_2 \oplus G_1(x_1), x_1, x_{14}, x_{13} \oplus G_{14}(x_{14}))$ then
                $Q.\textsc{Enqueue}(x_1, x_2, 1, 10)$
    else if $i = 7$ then
        forall $(x_7, x_8) \in \{x\} \times G_8$ do
            $Q.\textsc{Enqueue}(x_7, x_8, 7, 4)$
    else if $i = 8$ then
        forall $(x_7, x_8) \in G_7 \times \{x\}$ do
            $Q.\textsc{Enqueue}(x_7, x_8, 7, 10)$

private procedure $\textsc{Check}(x_0, x_1, x_{14}, x_{15})$
    return $\mathbf{P}(x_0, x_1) = (x_{14}, x_{15})$        $\boxed{\textbf{return } \mathbf{R}.\textsc{Check}(x_0, x_1, x_{14}, x_{15})}$

private procedure $\textsc{EvaluateForward}(x_k, x_{k+1}, k, \ell)$:
    while $k \neq \ell$ do
        if $k = 14$ then
            $(x_0, x_1) := \mathbf{P}^{-1}(x_{14}, x_{15})$        $\boxed{(x_0, x_1) := \mathbf{R}.\mathrm{P}^{-1}(x_{14}, x_{15})}$
            $k := 0$
        else
            $x_{k+2} := x_k \oplus \mathrm{F}^{\mathrm{INNER}}(k + 1, x_{k+1})$
            $k := k + 1$
    return $(x_\ell, x_{\ell+1})$

private procedure $\textsc{EvaluateBackward}(x_k, x_{k+1}, k, \ell)$:
    while $k \neq \ell$ do
        if $k = 0$ then
            $(x_{14}, x_{15}) := \mathbf{P}(x_0, x_1)$        $\boxed{(x_{14}, x_{15}) := \mathbf{R}.\mathrm{P}(x_0, x_1)}$
            $k := 14$
        else
            $x_{k-1} := x_{k+1} \oplus \mathrm{F}^{\mathrm{INNER}}(k, x_k)$
            $k := k - 1$
    return $(x_\ell, x_{\ell+1})$
```

3.1.3. *An Example of Chain Completion*

We provide an illustrative example of a simulator execution for the following distinguisher queries: First, choose x_9 and x_{10} arbitrarily. For $i = 10, \ldots, 14$ let $x_{i+1} := x_{i-1} \oplus \mathrm{F}(i, x_i)$, define $(x_0, x_1) := \mathbf{P}^{-1}(x_{14}, x_{15})$, and for $i = 1, 2$ let $x_{i+1} := x_{i-1} \oplus \mathrm{F}(i, x_i)$.

Suppose all hash tables and the queue are initially empty. For the queries $\mathrm{F}(i, x_i)$ for $i = 10, \ldots, 14$, $G_i(x_i)$ is set uniformly in $\mathrm{F}^{\mathrm{INNER}}$, as G_i is empty before the call. After the call to $\mathrm{F}^{\mathrm{INNER}}$, the queue is empty: only for $i = 13$, ENQUEUENEWCHAINS is called, but since G_{14} is empty at this point, no chain is enqueued. In $\mathrm{F}(1, x_1)$, $G_1(x_1)$ is set uniformly in $\mathrm{F}^{\mathrm{INNER}}$, and no chain is enqueued. Finally, when $\mathrm{F}(2, x_2)$

is called, $G_2(x_2)$ is set uniformly at random in $\mathrm{F}^{\text{INNER}}$, and ENQUEUENEWCHAINS$(2, x_2)$ is called. In this call, the tuple $(x_1, x_2, x_{13}, x_{14})$ makes CHECK evaluate to true, and $(x_1, x_2, 1, 4)$ is enqueued. When the call to $\mathrm{F}^{\text{INNER}}$ returns in F, $(x_1, x_2, 1, 4)$ is dequeued. As it is not in CompletedChains, the chain gets completed as follows. EVALUATEFORWARD$(x_1, x_2, 1, 2)$ returns (x_2, x_3) where $x_3 = x_1 \oplus G_2(x_2)$, and no new entries are added to G_i's. In EVALUATEBACKWARD$(x_1, x_2, 1, 6)$, for $i = 14, \dots, 10$, we have $x_i \in G_i$ already, and for $i = 9, \dots, 7$, letting $x_i = x_{i+2} \oplus G(x_{i+1})$, the values $G(x_i)$ are defined uniformly in $\mathrm{F}^{\text{INNER}}$. When $G(x_7)$ is defined in $\mathrm{F}^{\text{INNER}}$, ENQUEUENEWCHAINS enqueues $(x_7, x_8, 7, 4)$, and this is the only chain that is enqueued during EVALUATEBACKWARD. Finally, for $x_6 := x_8 \oplus G(x_7)$, ADAPT$(x_2, x_3, x_6, x_7, 4)$ is called, and both $(x_1, x_2, 1)$, $(x_7, x_8, 7)$ are added to CompletedChains. Thus, when $(x_7, x_8, 7)$ is dequeued in the next iteration, it is skipped. Finally, Q is empty, and F returns $G_2(x_2)$.

3.2. *Proof of Indifferentiability*

In this section, we provide the indifferentiability analysis.

3.2.1. *Overview*

Our overall plan is to show that for any deterministic distinguisher $\mathbf{D}$ that makes at most q queries[6], the probability that $\mathbf{D}$ outputs 1 when interacting with $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ differs by at most $\frac{\text{poly}(q)}{2^n}$ from the probability it outputs 1 when interacting with $(\Psi^{\mathbf{F}}, \mathbf{F})$, where Ψ is a fourteen-round Feistel construction, and $\mathbf{F}$ is a collection of 14 uniform random functions.

We denote the scenario where a distinguisher $\mathbf{D}$ interacts with $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ by S_1, and the scenario where $\mathbf{D}$ interacts with $(\Psi^{\mathbf{F}}, \mathbf{F})$ by S_4. Both S_1 and S_4 are depicted in Fig. 3. The scenarios S_2 and S_3 will be intermediate scenarios, that we describe in the following. When we use the term "execution of S_i", we always have a fixed (deterministic) distinguisher in mind, without mentioning it explicitly. Also, whenever we prove a statement about an "execution of S_i", this means that the statement holds for any fixed distinguisher that issues at most q queries in scenario S_i.

The transition from S_1 *to* S_2 To obtain S_2 from S_1, we replace the random permutation $\mathbf{P}$ by a two-sided random function $\mathbf{R}$. Informally, $\mathbf{R}$ can be described as follows. Fresh queries are always answered with uniform random bitstrings, and once a query $\mathbf{R}(x_0, x_1)$ was answered by choosing uniform random (x_{14}, x_{15}), the system will answer consistently in the future, i.e., future queries will be answered as $\mathbf{R}(x_0, x_1) = (x_{14}, x_{15})$ and $\mathbf{R}^{-1}(x_{14}, x_{15}) = (x_0, x_1)$. Clearly, it may happen that when a random answer for the second query $\mathbf{R}(x_0', x_1')$ for some $(x_0', x_1') \neq (x_0, x_1)$ is chosen randomly, it again equals (x_{14}, x_{15}). It is very intuitive that such collisions occur only with small probability. Also, given that collisions rarely occur, it is intuitive that $\mathbf{R}$ behaves like a random permutation. We make the randomness used by the simulator and $\mathbf{R}$ explicit, and write

[6] We may assume that $\mathbf{D}$ is deterministic, since we are only interested in the advantage of the optimal distinguisher, and for any probabilistic distinguisher, the advantage can be at most the advantage of the optimal deterministic distinguisher.

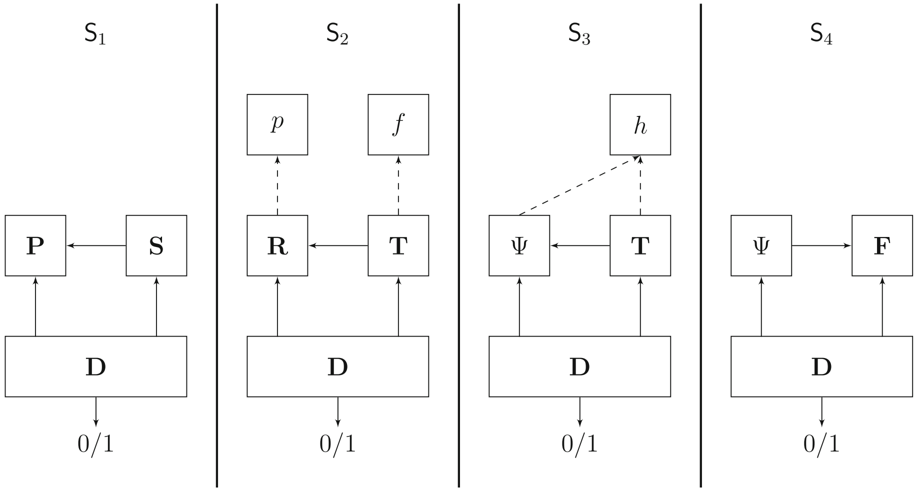

Fig. 3. Scenarios used in the indifferentiability proof.

$\mathsf{S}_2(f, p)$ for the scenario where the simulator (now denoted by $\mathbf{T}(f)$) uses randomness from f and $\mathbf{R}(p)$ uses randomness from p. In Lemma 3.6 we formally prove that $\mathbf{P}$ and $\mathbf{R}(p)$ can be distinguished only with negligible probability. This, together with the fact that the simulator is efficient in S_2 (as discussed below), directly gives that S_1 and $\mathsf{S}_2(f, p)$ can be distinguished with negligible probability for uniformly chosen f and p. This is formally stated in Lemma 3.11, and treated in Sect. 3.4.

The simulator is efficient in S_2 It is easier to prove the simulator's efficiency in scenario S_2. Lemmas 3.4 and 3.5 state that the query complexity and the running time, respectively, are poly(q). We are going to prove the simulator's efficiency in S_2 before analyzing the transition from S_1 to S_2, since we need it there (i.e., for proving Lemma 3.11). This is treated in Sect. 3.3.

The simulator is efficient in S_1 This directly follows from the fact that the simulator is efficient in S_2, and S_1 and S_2 can be distinguished only with negligible probability. The formal statement can be found in Lemma 3.10.

The transition from S_2 *to* S_3 In scenario S_3, we replace the two-sided random function $\mathbf{R}(p)$ by the fourteen-round Feistel construction $\Psi(h)$, which uses randomness in h. The same randomness h is accessed by the simulator $\mathbf{T}(h)$. The main part of our indifferentiability proof is to show that $\mathsf{S}_2(f, p)$ for uniform random (f, p) and $\mathsf{S}_3(h)$ for uniform random h can be distinguished only with negligible probability. This is formally stated in Lemma 3.37, which will be proved in Sect. 3.5. The proof of this lemma is the main part of the indifferentiability proof. A large part of the proof consists in showing that the simulator does not overwrite a value in calls to ForceVal. An interesting feature of the proof is that in a second part it directly maps pairs (f, p) to elements $h = \tau(f, p)$ such that $\mathsf{S}_2(f, p)$ and $\mathsf{S}_3(h)$ behave the same for most pairs (f, p), and the distribution induced by τ is close to uniform.

The transition from S_3 *to* S_4 It follows by definition that whenever a query is answered by the simulator in S_3, then it is answered with the corresponding entry of h that is used in Ψ (see Lemma 3.38). Since S_2 and S_3 are close, and in S_2 the simulator is efficient, this implies that with overwhelming probability the simulator gives an answer after a polynomial number of steps in S_3.[7] This implies that S_3 and S_4 can be distinguished only with negligible probability, as stated in Lemma 3.39, which will be proved in Sect. 3.6.

We now describe the two intermediate scenarios $\mathsf{S}_2(f, p)$ and $\mathsf{S}_3(h)$ in detail.

3.2.2. *Detailed Description of the Second Scenario*

Scenario $\mathsf{S}_2(f, p)$ is similar to S_1. However, instead of the simulator **S** we use the simulator $\mathbf{T}(f)$, and instead of a random permutation **P** we use a two-sided random function $\mathbf{R}(p)$. The differences between these systems are as follows:

Explicit randomness We make the randomness used by the simulator explicit. Whenever **S** sets $G_i(x_i)$ to a random value, $\mathbf{T}(f)$ takes it from $f(i, x_i)$ instead, where f is a table which contains an independent uniform random bitstring of length n for each $i \in \{1, 2, \ldots, 14\}$ and $x_i \in \{0, 1\}^n$. This modification does not change the distribution of the simulation, because it is clear that the simulator considers each entry of f at most once.

As can be seen in the pseudocode below, the randomness of the two-sided random function $\mathbf{R}(p)$ is also explicit: It is taken from $p(\downarrow, x_0, x_1)$ or $p(\uparrow, x_{14}, x_{15})$, a table in which each entry is an independent uniform random bitstring of length $2n$.

When we say that some entry of a table is "queried", this just means that the entry is read from the table.

Two-sided random function We replace the random permutation **P** by a two-sided random function $\mathbf{R}(p)$ (see below for pseudocode). This function keeps a hashtable P that contains elements $(\downarrow, x_0, x_1)$ and $(\uparrow, x_{14}, x_{15})$. Whenever the procedure $\mathbf{R}.\mathrm{P}(x_0, x_1)$ is queried, **R** checks whether $(\downarrow, x_0, x_1) \in P$, and if so, answers accordingly. Otherwise, an independent uniform random output (x_{14}, x_{15}) is picked (by considering p), and $(\downarrow, x_0, x_1)$ as well as $(\uparrow, x_{14}, x_{15})$ are added to P, mapping to each other.

Check **procedure** The two-sided random function **R** has a procedure Check$(x_0, x_1, x_{14}, x_{15})$. If $(\downarrow, x_0, x_1) \in P$, it returns true if P maps $(\downarrow, x_0, x_1)$ to (x_{14}, x_{15}), and false otherwise. If $(\uparrow, x_{14}, x_{15}) \in P$, it returns true if P maps $(\uparrow, x_{14}, x_{15})$ to (x_0, x_1), and false otherwise. If both $(\downarrow, x_0, x_1) \notin P$ and $(\uparrow, x_{14}, x_{15}) \notin P$, Check returns false. The simulator $\mathbf{T}(f)$ also differs from **S** in that $\mathbf{T}(f)$.Check simply calls **R**.Check.

Pseudocode for $\mathbf{T}(f)$ can be obtained by using the boxed contents on the right hand side in the pseudocode of **S** instead of the corresponding line. For the two-sided random function **R**, the pseudocode looks as follows:

```
System Two-sided random function $\mathbf{R}(p)$:
Variables:
```

[7] It is actually not hard to see that the simulator always gives an answer in S_3 after a finite number of steps, but we don't need to show this as S_2 and S_3 behave almost the same anyway.

```
3        Hashtable $P$
4
5   **public procedure** P$(x_0, x_1)$
6        **if** $(\downarrow, x_0, x_1) \notin P$ **then**
7             $(x_{14}, x_{15}) := p(\downarrow, x_0, x_1)$
8             $P(\downarrow, x_0, x_1) := (x_{14}, x_{15})$
9             $P(\uparrow, x_{14}, x_{15}) := (x_0, x_1)$        // (May overwrite an entry)
10       **return** $P(\downarrow, x_0, x_1)$
11
12  **public procedure** P$^{-1}(x_{14}, x_{15})$
13       **if** $(\uparrow, x_{14}, x_{15}) \notin P$ **then**
14            $(x_0, x_1) := p(\uparrow, x_{14}, x_{15})$
15            $P(\downarrow, x_0, x_1) := (x_{14}, x_{15})$        // (May overwrite an entry)
16            $P(\uparrow, x_{14}, x_{15}) := (x_0, x_1)$
17       **return** $P(\uparrow, x_{14}, x_{15})$
18
19  **public procedure** CHECK$(x_0, x_1, x_{14}, x_{15})$
20       **if** $(\downarrow, x_0, x_1) \in P$ **then return** $P(\downarrow, x_0, x_1) = (x_{14}, x_{15})$
21       **if** $(\uparrow, x_{14}, x_{15}) \in P$ **then return** $P(\uparrow, x_{14}, x_{15}) = (x_0, x_1)$
22       **return false**
```

Note that the CHECK procedure never returns true in line 21, because $P(\uparrow, x_{14}, x_{15}) = (x_0, x_1)$ implies that $(\downarrow, x_0, x_1) \in P$. Still, we think this is the most intuitive way of writing the CHECK procedure.

3.2.3. *Detailed Description of the Third Scenario*

In $\mathsf{S}_3(h)$, we replace the above two-sided random function $\mathbf{R}(p)$ by a Feistel construction $\Psi(h)$. Similar to S_2, h is used to make the randomness explicit. $\Psi(h)$ is defined as follows:

```
**System** $\Psi(h)$:

**Variables**:
     Hashtable $P$

**public procedure** P$(x_0, x_1)$
     **for** $i := 2$ **to** 15 **do**
          $x_i := x_{i-2} \oplus h(i-1, x_{i-1})$
     $P(\downarrow, x_0, x_1) := (x_{14}, x_{15})$
     $P(\uparrow, x_{14}, x_{15}) := (x_0, x_1)$
     **return** $(x_{14}, x_{15})$

**public procedure** P$^{-1}(x_{14}, x_{15})$
     **for** $i := 13$ **to** 0 **step** $-1$ **do**
          $x_i := x_{i+2} \oplus h(i+1, x_{i+1})$
     $P(\downarrow, x_0, x_1) := (x_{14}, x_{15})$
```

```
        $P(\uparrow, x_{14}, x_{15}) := (x_0, x_1)$
        return $(x_0, x_1)$

    public procedure CHECK$(x_0, x_1, x_{14}, x_{15})$
        if $(\downarrow, x_0, x_1) \in P$ then return $P(\downarrow, x_0, x_1) = (x_{14}, x_{15})$
        if $(\uparrow, x_{14}, x_{15}) \in P$ then return $P(\uparrow, x_{14}, x_{15}) = (x_0, x_1)$
        return false
```

We define $\mathsf{S}_3(h)$ to be the scenario where the distinguisher interacts with $(\Psi(h), \mathbf{T}(h)^{\Psi(h)})$. Note that the randomness used by Ψ and $\mathbf{T}$ is the same, and we call it h.

3.2.4. *Indifferentiability*

We will prove the following four lemmas.

Lemma 3.10. *Consider an execution of* S_1. *Then with probability at least* $1 - \frac{2 \cdot 10^7 \cdot q^{16}}{2^{2n}}$, *the simulator runs for at most* $O(q^8)$ *steps and issues at most* $1400q^8$ *queries to* $\mathbf{P}$.

Lemma 3.11. *The probability that a fixed distinguisher answers 1 in* S_1 *differs at most by* $\frac{4 \cdot 10^7 \cdot q^{16}}{2^{2n}}$ *from the probability that it answers 1 in* $\mathsf{S}_2(f, p)$ *for uniform random* (f, p).

Lemma 3.37. *The probability that a fixed distinguisher answers 1 in* $\mathsf{S}_2(f, p)$ *for uniform random* (f, p) *differs at most by* $\frac{10^{21} \cdot q^{10}}{2^n}$ *from the probability that it answers 1 in* $\mathsf{S}_3(h)$ *for uniform random* h.

Lemma 3.39. *The probability that a fixed distinguisher answers 1 in* $\mathsf{S}_3(h)$ *for uniformly chosen* h *differs at most by* $\frac{10^{21} \cdot q^{10}}{2^n}$ *from the probability that it answers 1 in* S_4.

Collecting these results allows to prove Theorem 3.1.

Proof of Theorem 3.1. Fix a distinguisher **D** which makes at most q queries. Lemmas 3.11, 3.37, and 3.39 give that the probability that **D** outputs 1 in $\mathsf{S}_1 = (\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ differs at most by

$$\frac{4 \cdot 10^7 \cdot q^{16}}{2^{2n}} + 2 \cdot \frac{10^{21} \cdot q^{10}}{2^n} < \frac{10^8 \cdot q^{16}}{2^{2n}} + \frac{10^{22} \cdot q^{10}}{2^n}$$

from the probability that it outputs 1 in $\mathsf{S}_4 = (\Psi^{\mathbf{F}}, \mathbf{F})$. Lemma 3.10 gives the desired bounds on the running time and query complexity of the simulator. □

3.3. *Complexity of the Simulator*

In this section, we show that the simulator is efficient in scenario $\mathsf{S}_2(f, p)$ for any f, p. Throughout the paper, for a hashtable G we denote by $|G|$ the number of entries in G.

Lemma 3.3. *Consider an execution of* $\mathsf{S}_2(f, p)$ *for some* (f, p). *Then, the simulator dequeues at most* q *times a partial chain of the form* $(x_1, x_2, 1, \ell)$ *for which* $(x_1, x_2, 1) \notin$ CompletedChains.

Proof. Consider such a dequeue call and let $(x_1, x_2, 1, \ell)$ be the partial chain dequeued for which $(x_1, x_2, 1) \notin$ CompletedChains. The chain $(x_1, x_2, 1, \ell)$ must have been enqueued when $(x_2 \oplus G_1(x_1), x_2, x_{14}, x_{13} \oplus G_{14}(x_{14}))$ was detected in line 40 or 43 of ENQUEUENEWCHAINS. Since neither $G_1(x_1)$ nor $G_{14}(x_{14})$ are ever overwritten, this means that we can find a unique 4-tuple $(x_0, x_1, x_{14}, x_{15})$, where $x_0 = x_2 \oplus G_1(x_1)$ and $x_{15} = x_{13} \oplus G_{14}(x_{14})$, associated with $(x_1, x_2, 1)$ for which CHECK$(x_0, x_1, x_{14}, x_{15})$ was true at the moment $(x_1, x_2, 1, \ell)$ was enqueued. We can now find a unique query to p which corresponds to $(x_0, x_1, x_{14}, x_{15})$: since CHECK$(x_0, x_1, x_{14}, x_{15})$ was true, there must have been a call to P or P^{-1} in $\mathbf{R}(p)$ where either $p(\downarrow, x_0, x_1) = (x_{14}, x_{15})$ or $p(\uparrow, x_{14}, x_{15}) = (x_0, x_1)$, respectively, was accessed in line 7 or 14 of $\mathbf{R}(p)$. This call to P or P^{-1} was made either by the distinguisher or the simulator. We argue that this call cannot have been issued by the simulator. The simulator issues such calls only when it completes a chain (i.e., within calls to EVALUATEBACKWARD and EVALUATEFORWARD in F), and after this completion, it adds $(x_1, x_2, 1)$ to CompletedChains (in line 17 of F). During this chain completion (in lines 12 to 17 of F), no calls to Q.DEQUEUE occur, and so it is not possible that $(x_1, x_2, 1) \notin$ CompletedChains when it was dequeued. Thus, we found a unique query to P or P^{-1} of the distinguisher associated with this dequeue call. Finally, note that after $(x_1, x_2, 1)$ is completed by the simulator, $(x_1, x_2, 1)$ is added to CompletedChains. Thus, there are at most q such dequeue calls. □

Lemma 3.4. *Consider an execution of* $\mathsf{S}_2(f, p)$ *for some* (f, p)*. Then, at any point in the execution we have* $|G_i| \leq 6q^2$ *for all* i*. Furthermore, there are at most* $6q^2$ *queries to both* $\mathbf{R}$.P*, and* $\mathbf{R}.\mathrm{P}^{-1}$*, and at most* $1296q^8$ *queries to* $\mathbf{R}$.CHECK*.*

Proof. We first show that $|G_7| \leq 2q$ and $|G_8| \leq 2q$. Assignments $G_7(x_7) := f(7, x_7)$ and $G_8(x_8) := f(8, x_8)$ only happen in two cases: either when the distinguisher directly queries the corresponding value using F, or when the simulator completes a chain $(x_1, x_2, 1, \ell)$ which it dequeued. There can be at most q queries to F, and according to Lemma 3.3 there are at most q such chains which are completed, which implies the bound.

The set G_i can only be enlarged by 1 in the following cases: if the distinguisher queries $\mathrm{F}(i, \cdot)$, if a chain of the form $(x_1, x_2, 1, \ell)$ is dequeued and not in CompletedChains, or if a chain $(x_7, x_8, 7, \ell)$ is dequeued and not in CompletedChains. There are at most q events of the first kind, at most q events of the second kind (using Lemma 3.3), and at most $|G_7| \cdot |G_8| \leq 4q^2$ events of the last kind, giving a total of $2q + 4q^2 \leq 6q^2$.

A query to $\mathbf{R}$.P or $\mathbf{R}.\mathrm{P}^{-1}$ can be made either by the distinguisher, or by the simulator when it completes a chain. At most q events of the first kind, and at most $q + 4q^2$ events of the second kind are possible. Thus, at most $6q^2$ of these queries occur. The number of CHECK queries by the simulator is bounded by $|G_1 \times G_2 \times G_{13} \times G_{14}| \leq (6q^2)^4$. □

Lemma 3.5. *Consider an execution of* $\mathsf{S}_2(f, p)$ *for some* (f, p)*. Then the simulator runs in time* $O(q^8)$*.*

Proof. We first establish the following claims: (i) The total number of chains that are dequeued and not in CompletedChains is at most $5q^2$. (ii) Any call to ENQUEUENEWCHAINS runs in time $O(q^6)$ and in each such call Q.ENQUEUE is called at most $216q^6$ times. (iii) The total number of calls to ENQUEUENEWCHAINS is at most $24q^2$. (iv) The total number of calls to F^{INNER} is at most $q + 14 \cdot 6q^2$. (v) The total number of calls to Q.ENQUEUE is at most $216 \cdot 24q^8$.

To see (i), note that by Lemma 3.3, the number of chains of the form $(x_1, x_2, 1, \ell)$ that are dequeued and not in CompletedChains is at most q. Furthermore, the number of chains of the form $(x_7, x_8, 7, \ell)$ that are dequeued and not in CompletedChains is at most $|G_7| \cdot |G_8| \leq 4q^2$.

Part (ii) can be seen as follows: By Lemma 3.4, we have that for all i, $|G_i| \leq 6q^2$. Thus for any i, the number of iterations in the forall loop of ENQUEUENEWCHAINS is at most $(6q^2)^3 = 216q^6$.

To see (iii), note that ENQUEUENEWCHAINS is called only in F^{INNER}, and as $|G_i| \leq 6q^2$ for all i and values of G_i are never overwritten, ENQUEUENEWCHAINS is called at most $4 \cdot 6q^2$ times.

To prove item (iv), note that calls F^{INNER} only occur in procedures F, EVALUATEBACKWARD, and EVALUATEFORWARD. The first case occurs at most q times, as F is only queried by the distinguisher. Calls to EVALUATEBACKWARD and EVALUATEFORWARD only occur in case a chain is dequeued and not in CompletedChains. By (i), this occurs at most $6q^2$ times. In each of the four calls to EVALUATEBACKWARD and EVALUATEFORWARD, F^{INNER} is called at most 14 times, and thus in total F^{INNER} is called at most $4 \cdot 14 \cdot 6q^2$ times. Summing up gives that F^{INNER} is called at most $q + 336q^2$ times.

Finally, (v) follows as by (ii) and (iii).

We now bound the simulator's running time. First note that each call to ADAPT, FORCEVAL, and CHECK runs in time $O(1)$. By (v), at most $216 \cdot 24q^8$ chains are ever dequeued, and by (i) at most $5q^2$ of them are ever completed. Thus, the number of steps within EVALUATEBACKWARD and EVALUATEFORWARD (excluding the steps within calls to F^{INNER}) is bounded by $O(q^2)$, and the number of steps within F (excluding the steps within F^{INNER}, EVALUATEBACKWARD and EVALUATEFORWARD) is bounded by $O(q^8)$. By items (ii) and (iii), the total running time of ENQUEUENEWCHAINS is bounded by $O(q^8)$. By (iv), the total running time of F^{INNER} (excluding the steps within calls to ENQUEUENEWCHAINS) is bounded by $O(q^2)$. This implies that the simulator runs in time $O(q^8)$. □

3.4. *Equivalence of the First and the Second Scenarios*

In this section, we show that for uniformly chosen (f, p) and any **D**, the probability that **D** outputs 1 in scenario $\mathsf{S}_2(f, p)$ differs only by $\frac{\text{poly}(q)}{2^n}$ from the probability it outputs 1 in scenario S_1. As a side-result, we will obtain that the simulator is efficient in S_1 with overwhelming probability. To show the first claim, we first note that clearly the simulator can take the randomness from f without any change. Secondly, instead of the procedure CHECK in the simulator **S**, we can imagine that the random permutation **P** has a procedure **P**.CHECK which is implemented exactly as in line 53 of **S**, and **S**.CHECK simply calls **P**.CHECK. The following lemma states that such a system **P** is indistinguishable from **R** as above, which we will use to prove our claim. The proof is neither surprising nor particularly difficult.

Lemma 3.6. *Consider a random permutation over $2n$ bits, to which we add the procedure* Check *as in line 53 of the simulator* **S**. *Then, a distinguisher which issues at most q' queries to either the random permutation or to the two-sided random function* **R** *has advantage at most $\frac{6(q')^2}{2^{2n}}$ in distinguishing the two systems.*

Throughout the proof, we will consider distinguishers that issue at most q' queries. For the indistinguishability proof we will use four scenarios $\mathsf{E}_1, \ldots, \mathsf{E}_4$, where E_1 will correspond to **D** interacting with **P**, and E_4 to **D** interacting with **R**. E_2 and E_3 are intermediate scenarios.

Scenario E_1: D interacts with $\mathbf{P}'(p)$, which is defined as follows: The procedures $\mathbf{P}'$.P and $\mathbf{P}'.\mathrm{P}^{-1}$ are the same as **R**.P and $\mathbf{R}.\mathrm{P}^{-1}$. The Check procedure is defined as

```
public procedure $\mathbf{P}'(p)$.Check$(x_0, x_1, x_{14}, x_{15})$
    if $(\downarrow, x_0, x_1) \in P$ then return $P(\downarrow, x_0, x_1) = (x_{14}, x_{15})$
    if $(\uparrow, x_{14}, x_{15}) \in P$ then return $P(\uparrow, x_{14}, x_{15}) = (x_0, x_1)$
    return $\mathrm{P}(x_0, x_1) = (x_{14}, x_{15})$        // Note that the proc. $\mathbf{P}'$.P is called!
```

Finally, p is the table of a uniform random permutation (i.e., $p(\downarrow, x_0, x_1) = (x_{14}, x_{15})$ if and only if $p(\uparrow, x_{14}, x_{15}) = (x_0, x_1)$).

In E_2, we introduce an alternative way to sample the random permutation.

Scenario E_2: D interacts with $\mathbf{P}''(p)$. In $\mathbf{P}''$, the procedure $\mathbf{P}''$.P is defined as follows:

```
public procedure $\mathbf{P}''.\mathrm{P}(x_0, x_1)$
    if $(\downarrow, x_0, x_1) \notin P$ then
        $(x_{14}, x_{15}) := p(\downarrow, x_0, x_1)$
        if $(\uparrow, x_{14}, x_{15}) \in P$ then
            $(x_{14}, x_{15}) \leftarrow_{\mathsf{R}} \{0,1\}^{2n} \setminus \{(x'_{14}, x'_{15}) | (\uparrow, x'_{14}, x'_{15}) \in P\}$
        $P(\downarrow, x_0, x_1) := (x_{14}, x_{15})$
        $P(\uparrow, x_{14}, x_{15}) := (x_0, x_1)$
    return $P(\downarrow, x_0, x_1)$
```

The procedure $\mathbf{P}''.\mathrm{P}^{-1}$ is defined analogously, i.e., picks (x_0, x_1) from p, and replaces it in case $(\downarrow, x_0, x_1) \in P$. The procedure Check is defined as in $\mathbf{P}'$.Check above. Finally, the entries of p are chosen uniformly at random from $\{0,1\}^{2n}$.

We next replace the table p of the random permutation by a table that has uniform random entries:

Scenario E_3: D interacts with $\mathbf{P}'(p)$, where the entries of p are chosen uniformly at random from $\{0,1\}^{2n}$.

Finally, we consider the experiment where **D** interacts with our two-sided random function.

Scenario E_4: D interacts with $\mathbf{R}(p)$, where the entries of p are chosen uniformly at random from $\{0,1\}^{2n}$.

The only difference between E_3 and E_4 is the change in the last line in the procedure Check.

Our goal is to prove that the consecutive scenarios behave almost identically. We state a lemma for each transition from E_i to E_{i+1}.

We first note that E_1 corresponds to an interaction with **P**: If we let **D** interact with **P** (adding a Check-procedure to **P** in the most standard way, i.e., as in the simulator **S**), then this behaves exactly as E_1.

Lemma 3.7. (Transition from E_1 to E_2) *The probability that a fixed distinguisher answers 1 in* E_1 *equals the probability that it answers 1 in* E_2.

Proof. The procedure Check is the same in both scenarios. Furthermore, a distinguisher can keep track of the table P and it is also the same in both scenarios, and so we only need to consider the procedures P and P^{-1}: the procedure Check could be a part of the distinguisher.

Now, in both scenarios, the values chosen in the procedures P and P^{-1} are chosen uniformly at random from the set of values that do not correspond to an earlier query. Thus, E_1 and E_2 behave identically. □

Lemma 3.8. (Transition from E_2 to E_3) *The probability that a fixed distinguisher outputs 1 in* E_1 *differs by at most* $\frac{(q')^2}{2^{2n}}$ *from the probability that it outputs 1 in* E_3.

This proof is very similar to the proof that a (one-sided) random permutation can be replaced by a (one-sided) random function.

Proof. Consider E_2 and let $\mathsf{BadQuery}$ be the event that in P we have $(\uparrow, x_{14}, x_{15}) \in P$, or in P^{-1} we have $(\downarrow, x_0, x_1) \in P$. We show that this event is unlikely and that the two scenarios behave identically if $\mathsf{BadQuery}$ does not occur in E_2.

There are at most q' queries to P or P^{-1} in an execution of E_2, since each Check query issues at most one query to P. Observe that each table entry in p is accessed at most once and thus each time p is accessed it returns a fresh uniform random value. Since for each query there are at most q' values in P, and p contains uniform random entries, we have $\Pr[\mathsf{BadQuery} \text{ occurs in } \mathsf{E}_2] \leq \frac{(q')^2}{2^{2n}}$. The scenarios E_2 and E_3 behave identically if $\mathsf{BadQuery}$ does not occur. Thus, $\Big|\Pr[\mathbf{D} \text{ outputs 1 in } \mathsf{E}_2] - \Pr[\mathbf{D} \text{ outputs 1 in } \mathsf{E}_3]\Big| \leq \Pr_p[\mathsf{BadQuery} \text{ occurs in } \mathsf{E}_2] \leq \frac{(q')^2}{2^{2n}}$. □

Lemma 3.9. (Transition from E_3 to E_4) *The probability that a fixed distinguisher outputs 1 in* E_3 *differs by at most* $\frac{5(q')^2}{2^{2n}}$ *from the probability that it outputs 1 in* E_4.

Proof. The event $\mathsf{BadCheck}$ occurs for some p if $\mathbf{P}'$.Check returns true in the last line in E_3 in an execution using p. The event $\mathsf{BadOverwrite}$ occurs for some p if either in E_3 or in E_4, in any call to P or P^{-1}, an entry of P is overwritten.[8] The event

[8] It would actually be sufficient to consider the scenario E_3 here, but we can save a little bit of work by considering both E_3 and E_4.

$\mathsf{BadBackwardQuery}$ occurs if in E_3 there exist (x_0, x_1), (x^*_{14}, x^*_{15}) such that all of the following hold:

(i) The query $\mathrm{P}(x_0, x_1)$ is issued in the last line of a Check query, and $P(\downarrow, x_0, x_1)$ is set to $(x^*_{14}, x^*_{15}) = p(\downarrow, x_0, x_1)$.
(ii) After (i), the query $\mathrm{P}^{-1}(x^*_{14}, x^*_{15})$, or the query $\textsc{Check}(x_0, x_1, x^*_{14}, x^*_{15})$ is issued.
(iii) The query $\mathrm{P}(x_0, x_1)$ is not issued by the distinguisher between point (i) and point (ii).

We show that these events are unlikely and that E_3 and E_4 behave identically if the events do not occur for a given p.

For $\mathsf{BadCheck}$ to occur in a fixed call $\mathbf{P}'.\textsc{Check}(x_0, x_1, x_{14}, x_{15})$, it must be that $(\downarrow, x_0, x_1) \notin P$ and $(\uparrow, x_{14}, x_{15}) \notin P$. Thus, in the call $\mathrm{P}(x_0, x_1)$ in the last line of Check, $P(\downarrow, x_0, x_1)$ will be set to a fresh uniform random value $p(\downarrow, x_0, x_1)$, and this value is returned by P. Therefore, the probability over the choice of p that $\mathrm{P}(x_0, x_1) = (x_{14}, x_{15})$ is at most $\frac{1}{2^{2n}}$. Since Check is called at most q' times, we see that $\Pr_p[\mathsf{BadCheck}] \leq \frac{q'}{2^{2n}}$.

We now bound the probability that $\mathsf{BadOverwrite}$ occurs in E_3. This only happens if a fresh uniform random entry read from p collides with an entry in P. Since there are at most q' queries to P and P^{-1} and at most q' entries in P, we get $\Pr_p[\mathsf{BadOverwrite}$ occurs in $\mathsf{E}_3] \leq \frac{(q')^2}{2^{2n}}$. The same argument gives a bound on $\mathsf{BadOverwrite}$ in E_4, and so $\Pr_p[\mathsf{BadOverwrite}] \leq \frac{2(q')^2}{2^{2n}}$.

We next estimate the probability of $(\mathsf{BadBackwardQuery} \wedge \neg\mathsf{BadCheck})$ in E_3. Consider any pairs (x_0, x_1), (x^*_{14}, x^*_{15}) such that (i) holds. Clearly, since $\mathsf{BadCheck}$ does not occur, the Check query returns false. Now, as long as none of the queries $\mathrm{P}(x_0, x_1)$, $\mathrm{P}^{-1}(x^*_{14}, x^*_{15})$ or $\textsc{Check}(x_0, x_1, x^*_{14}, x^*_{15})$ is made by the distinguisher, the value (x^*_{14}, x^*_{15}) is distributed uniformly in the set of all pairs (x'_{14}, x'_{15}) for which $\textsc{Check}(x_0, x_1, x'_{14}, x'_{15})$ was not queried. Thus, the probability that in a single query, the distinguisher queries one of $\mathrm{P}^{-1}(x^*_{14}, x^*_{15})$ or $\textsc{Check}(x_0, x_1, x^*_{14}, x^*_{15})$ is at most $\frac{q'}{2^{2n}-q'} \leq \frac{2q'}{2^{2n}}$ (assuming $q' < \frac{2^{2n}}{2}$). Since there are at most q' Check queries, we find

$$\Pr_p[(\mathsf{BadBackwardQuery} \wedge \neg\mathsf{BadCheck})] \leq \frac{2(q')^2}{2^{2n}}.$$

We proceed to argue that if the bad events do not occur, the two scenarios behave identically. Thus, let p be a table such that none of $\mathsf{BadCheck}$, $\mathsf{BadBackwardQuery}$, and $\mathsf{BadOverwrite}$ occurs.

We first observe that the following invariant holds in both E_3 and E_4: after any call to P, P^{-1} or Check, if $P(\downarrow, x_0, x_1) = (x_{14}, x_{15})$ for some values $(x_0, x_1, x_{14}, x_{15})$, then $P(\uparrow, x_{14}, x_{15}) = (x_0, x_1)$, and vice versa. The reason is simply that no value is ever overwritten in the tables, and whenever $P(\uparrow, \cdot, \cdot)$ is set, then $P(\downarrow, \cdot, \cdot)$ is also set.

Next, we argue inductively that for a p for which none of the bad events occur, all queries and answers in E_3 and E_4 are the same.

For this, we start by showing that (assuming the induction hypothesis), if a triple $(\downarrow, x_0, x_1)$ is in P in the scenario E_4, then the triple is in P in E_3 as well, and both have the same image (x_{14}, x_{15}). This holds because of two reasons: First, in E_4, each such entry corresponds to an answer to a previously issued query to P or P^{-1}. This query was also issued in E_3, and at that point the answer was identical, so that the entry $P(\downarrow, x_0, x_1)$ was identical (this also holds if the response in E_3 is due to the entry $P(\uparrow, x_{14}, x_{15})$, because we saw above that this implies $P(\downarrow, x_0, x_1) = (x_{14}, x_{15})$). Since the event $\mathsf{BadOverwrite}$ does not occur, the property will still hold later. (We remark that entries in the table P in E_3 may exist which are not in E_4.)

We now establish our claim that all queries and answers of the distinguisher in E_3 and E_4 are the same.

Consider first a P-query $\mathrm{P}(x_0, x_1)$. If $(\downarrow, x_0, x_1) \in P$ in E_4, the previous paragraph gives the result for this query. If $(\downarrow, x_0, x_1) \notin P$ in both E_3 and E_4, the same code is executed. The only remaining case is $(\downarrow, x_0, x_1) \in P$ in E_3 and $(\downarrow, x_0, x_1) \notin P$ in E_4. The only way this can happen is if the query $\mathrm{P}(x_0, x_1)$ was invoked previously from a query to CHECK in E_3, in which case the same entry $p(\downarrow, x_0, x_1)$ was used to set P, and we get the result.

Next, we consider a P^{-1}-query $\mathrm{P}^{-1}(x_{14}^*, x_{15}^*)$. Again, the only non-trivial case is if $(\uparrow, x_{14}^*, x_{15}^*) \in P$ in E_3 and $(\uparrow, x_{14}^*, x_{15}^*) \notin P$ in E_4. This is only possible if during some query to $\textsc{Check}(x_0, x_1, \cdot, \cdot)$ in E_3, the last line invoked $\mathrm{P}(x_0, x_1)$, and $(x_{14}^*, x_{15}^*) = p(\downarrow, x_0, x_1)$. Since it also must be that until now the distinguisher never invoked $\mathrm{P}(x_0, x_1)$ (otherwise, $P(\uparrow, x_{14}^*, x_{15}^*) = (x_0, x_1)$ in E_4), this implies that the event $\mathsf{BadBackwardQuery}$ must have happened.

Finally, consider a call $\textsc{Check}(x_0, x_1, x_{14}, x_{15})$ to CHECK. In case $(\downarrow, x_0, x_1) \in P$ in E_4 and in case $(\downarrow, x_0, x_1) \notin P$ in E_3, line 20 behaves the same in both E_3 and E_4. If $(\downarrow, x_0, x_1) \in P$ in E_3 and $(\downarrow, x_0, x_1) \notin P$ in E_4, then in E_4, CHECK returns false. In E_3, CHECK can only return true if the event $\mathsf{BadBackwardQuery}$ occurs.

The second if statement in CHECK (in E_4 this is line 21 of $\mathbf{R}$) can only return false in both E_3 and E_4: otherwise, the first if statement in CHECK (in E_4 this is line 20 of $\mathbf{R}$) would already have returned true. This is sufficient, because the event $\mathsf{BadCheck}$ does not occur, and so the last line of CHECK in both scenarios also returns false.

Thus,

$$\begin{aligned}
&\left|\Pr_p[\mathbf{D} \text{ outputs 1 in } \mathsf{E}_3] - \Pr_p[\mathbf{D} \text{ outputs 1 in } \mathsf{E}_4]\right| \\
&\quad \le \Pr_p[(\mathsf{BadCheck} \vee \mathsf{BadOverwrite} \vee \mathsf{BadBackwardQuery})] \\
&\quad \le \frac{q'}{2^{2n}} + \frac{2(q')^2}{2^{2n}} + \frac{2(q')^2}{2^{2n}} \le \frac{5(q')^2}{2^{2n}}.
\end{aligned}$$

□

Proof of Lemma 3.6. Since E_1 corresponds to an interaction with $\mathbf{P}$, while E_4 corresponds to an interaction with $\mathbf{R}$, Lemma 3.6 now follows immediately from Lemmas 3.7, 3.8, and 3.9 as

$$\left|\Pr[\mathbf{D} \text{ outputs 1 in } \mathsf{E}_1] - \Pr[\mathbf{D} \text{ outputs 1 in } \mathsf{E}_4]\right| \le \frac{(q')^2}{2^{2n}} + \frac{5(q')^2}{2^{2n}} \le \frac{6(q')^2}{2^{2n}}.$$

□

We are now able to prove that the simulator is efficient in S_1 with overwhelming probability.

Lemma 3.10. (Efficiency of the simulator) *Consider an execution of* S_1. *Then, with probability at least* $1 - \frac{2 \cdot 10^7 \cdot q^{16}}{2^{2n}}$, *the simulator runs for at most* $O(q^8)$ *steps and issues at most* $1400q^8$ *queries to* $\mathbf{P}$.

Proof. By Lemmas 3.4 and 3.5, in $\mathsf{S}_2(f, p)$ for uniform (f, p), there are at most $6q^2 + 1296q^8 \leq 1400q^8$ queries to $\mathbf{R}(p)$, and the simulator runs in time $r(q) \in O(q^8)$ for some function r.

Toward a contradiction suppose there exists a distinguisher $\mathbf{D}$ that issues at most q queries such that in S_1 the simulator runs for more than $r(q)$ steps or makes more than $1400q^8$ queries to $\mathbf{P}$, with probability larger than $\frac{2 \cdot 10^7 \cdot q^{16}}{2^{2n}}$. We define a new distinguisher $\mathbf{D}'$ that consists of $\mathbf{D}$ and the simulator together. $\mathbf{D}'$ outputs 1 if and only if the simulator takes more than $r(q)$ steps or more than $1400q^8$ queries are issued by $\mathbf{D}$ and the simulator (i.e., $\mathbf{D}'$ stops and outputs 1 immediately before the $1400q^8 + 1$'st query is issued, so that $\mathbf{D}'$ never issues more than $1400q^8$ queries). Then $\mathbf{D}'$ issues at most $1400q^8$ queries and distinguishes $\mathbf{R}(p)$ from $\mathbf{P}$ with probability larger than $\frac{2 \cdot 10^7 \cdot q^{16}}{2^{2n}} > \frac{6(1400q^8)^2}{2^{2n}}$, which contradicts Lemma 3.6. We conclude that such a distinguisher $\mathbf{D}$ cannot exist. □

Finally, this allows us to conclude that scenarios S_1 and S_2 can be distinguished only with negligible probability.

Lemma 3.11. (Transition from S_1 to S_2) *The probability that a fixed distinguisher answers 1 in* S_1 *differs at most by* $\frac{4 \cdot 10^7 \cdot q^{16}}{2^{2n}}$ *from the probability that it answers 1 in* $\mathsf{S}_2(f, p)$ *for uniform random* (f, p).

Proof. Toward a contradiction assume that there exists a distinguisher $\mathbf{D}$ that issues at most q queries and distinguishes with probability larger than $\frac{4 \cdot 10^7 \cdot q^{16}}{2^{2n}}$. We now define a distinguisher $\mathbf{D}'$ for $\mathbf{P}$ and $\mathbf{R}(p)$ as follows: $\mathbf{D}'$ consists of $\mathbf{D}$ and the simulator together. $\mathbf{D}'$ counts the total number of queries of both $\mathbf{D}$ and the simulator, and whenever the query limit of $1400q^8$ queries is exceeded, $\mathbf{D}'$ stops and outputs 1. Otherwise, $\mathbf{D}'$ outputs whatever $\mathbf{D}$ outputs. The query limit is never reached in S_2 by Lemma 3.4, and in S_1 this occurs with probability at most $\frac{2 \cdot 10^7 \cdot q^{16}}{2^{2n}}$ by Lemma 3.10. By definition, $\mathbf{D}'$ issues at most $1400q^8$ queries. Clearly, $\mathbf{D}'$ still distinguishes with probability at least $\frac{(4-2) \cdot 10^7 \cdot q^{16}}{2^{2n}}$. But this contradicts Lemma 3.6, as the probability that $\mathbf{D}'$ outputs 1 when interacting with $\mathbf{P}$ differs by at least $\frac{2 \cdot 10^7 \cdot q^{16}}{2^{2n}} > \frac{6(1400q^8)^2}{2^{2n}}$ from the probability that it outputs 1 when interacting with $\mathbf{R}(p)$. Thus, such a $\mathbf{D}$ cannot exist. □

3.5. *Equivalence of the Second and the Third Scenarios*

This section contains the core of our argument: We prove Lemma 3.37, which states that $\mathsf{S}_2(f, p)$ and $\mathsf{S}_3(h)$ have the same behavior for uniformly chosen (f, p) and h. We let $G = (G_1, \ldots, G_{14})$ be the tuple of tables of the simulator $\mathbf{T}(f)$ in the execution.

3.5.1. *Overview and Intuition*

Bad events and good executions in $\mathsf{S}_2(f, p)$ We will define the events BadP, $\mathsf{BadlyHit}$, and $\mathsf{BadlyCollide}$, which should occur with small probability over the choice of (f, p) in an execution of $\mathsf{S}_2(f, p)$. Intuitively, the event BadP occurs if, when an entry of p is read by the simulator, an unexpected collision with values in P or G occurs. The event $\mathsf{BadlyHit}$ captures unexpected collisions produced by assignments of the form $G_i(x) := f(i, x)$, and the event $\mathsf{BadlyCollide}$ happens if after such an assignment two chains that are defined by G and P suddenly lead to the same value, even though this was not expected. Finally, we call (f, p) good if none of these events occur in an execution of $\mathsf{S}_2(f, p)$. We are going to prove that a randomly chosen pair (f, p) is good with high probability (Lemma 3.22).

Properties of good executions in $\mathsf{S}_2(f, p)$*: No values are overwritten* We will establish the following property for executions of $\mathsf{S}_2(f, p)$ for good (f, p).

(i) No call to ForceVal overwrites an entry. That is, for any call to ForceVal of the form $\textsc{ForceVal}(x, \cdot, \ell)$, we have $x \notin G_\ell$ before the call. (Lemma 3.31)

Proving this requires a careful analysis of good executions and is one of our main technical contributions. We now give a proof sketch.

A *partial chain* is a triple (x_k, x_{k+1}, k). Given a partial chain, e.g., $(x_3, x_4, 3)$, it may be possible to move "forward" or "backward" one step in the Feistel construction, i.e., if $x_4 \in G_4$, we may obtain $x_5 = x_3 \oplus G_4(x_4)$, and if $x_3 \in G_3$, we may obtain $x_2 = x_4 \oplus G_3(x_3)$. Entries in P may also allow such moves: for example, in case of a partial chain $(x_0, x_1, 0)$, moving backward is possible if $(\downarrow, x_0, x_1) \in P$. We say that two partial chains are *equivalent* if it is possible to reach one chain from the other by moving forward and backward as described, given the tables G and P. Note that this relation is not necessarily symmetric, as BadP may occur. We say that a partial chain is *table-defined* if it is possible to move both one step backward and one step forward using G and P. Intuitively, this just means that the two values that describe the partial chain are in the tables G, P.

Now suppose (f, p) is good. Then, as BadP does not occur, at any point in the execution equivalence between partial chains is an equivalence relation (Lemma 3.23).

Next, we show that the equivalence relation among table-defined chains persists: If two partial chains C and D are table-defined, then assignments of the form $G_i(x) := f(i, x)$ and assignments to P in $\mathbf{R}$ do not change the equivalence relation between C and D (Lemma 3.25). Furthermore, calls to ForceVal do not change this equivalence relation, given that the buffer rounds (namely rounds 3 and 6 or 9 and 12) around the adapt zones (see Fig. 2) are still undefined when Adapt is called (Lemma 3.26(c)).

Next we show that indeed the buffer rounds are still undefined when Adapt is called: We first show that they are undefined when a chain is enqueued for which no equivalent chain was enqueued previously (Lemma 3.29), and then, using that the equivalence relation among chains persists, we show that they are still undefined when the chain is dequeued (Lemma 3.30). The proof of the latter crucially relies on the fact that the event $\mathsf{BadlyCollide}$ does not occur.

Finally, we conclude (i) as follows: As the buffer rounds around the adapt zones are still undefined when ADAPT is called, the calls to FORCEVAL do not overwrite a value, as otherwise $\mathsf{BadlyHit}$ would occur (Lemma 3.26(a)).

Further properties of good executions in $\mathsf{S}_2(f, p)$ We say that *the distinguisher completes all chains* if for each query to $\mathrm{P}(x_0, x_1)$ or $(x_0, x_1) = \mathrm{P}^{-1}(x_{14}, x_{15})$ by the distinguisher, it issues the corresponding Feistel queries to F in the end of the execution (i.e., it emulates a call to EVALUATEFORWARD$(x_0, x_1, 0, 14)$). We may assume that the distinguisher completes all chains: This multiplies the number of queries at most by a factor of 15, and the modified distinguisher achieves at least the advantage of the original distinguisher.

For such a modified distinguisher, (i) implies the following properties for good (f, p):

(ii) At the end of an execution of $\mathsf{S}_2(f, p)$, for any table entry $P(\downarrow, x_0, x_1) = (x_{14}, x_{15})$, emulating an evaluation of the Feistel construction on x_0, x_1 using the tables G also yields (x_{14}, x_{15}). The analogous statement holds for $P(\uparrow, x_{14}, x_{15})$. (Lemma 3.32)

(iii) The number of calls to ADAPT equals the number of queries to $p(\cdot, \cdot, \cdot)$ made by the two-sided random function. (Lemma 3.33)

It is intuitive that (ii) holds: As the distinguisher completes all chains, by (i) no values are ever overwritten, and BadP does not occur, it follows that for each query to p there will be a chain completion. Furthermore, the values corresponding to this chain will not be changed afterward. To prove (iii), we will give a one-to-one mapping between ADAPT calls and queries to p.

Mapping randomness of S_2 *to randomness of* S_3 Our final goal is to prove that $\mathsf{S}_2(f, p)$ for random (f, p) and $\mathsf{S}_3(h)$ for random h cannot be distinguished. For this, we give a map τ that maps pairs (f, p) to tables h as follows. If (f, p) is good, run a simulation of $\mathsf{S}_2(f, p)$ in which the distinguisher completes all chains. Consider the tables G at the end of this execution, and for any i and x let $h(i, x) := G_i(x)$ in case $x \in G_i$, and $h(i, x) := \perp$ otherwise. If (f, p) is not good, let $\tau(f, p) := \lambda$. Now (i) and (ii) allow us to show that $\mathsf{S}_2(f, p)$ behaves exactly as $\mathsf{S}_3(\tau(f, p))$ for good (f, p), in the sense that all queries and answers to f (or h) by the simulator, and all queries and answers to $\mathbf{R}$ (or Ψ) are identical in both scenarios (Lemma 3.35).

Finally, using (iii) we can argue that the distribution $\tau(f, p)$ for random (f, p) is close to uniform, if the $\perp$ entries of $\tau(f, p)$ are replaced by uniform random entries. This implies that S_2 and S_3 cannot be distinguished (Lemma 3.37).

3.5.2. *Partial Chains*

Evaluating Partial Chains A *partial chain* is a triple $(x_k, x_{k+1}, k) \in \{0, 1\}^n \times \{0, 1\}^n \times \{0, \ldots, 14\}$. Given such a partial chain C, and a set of tables $\mathbf{T}.G$ and $\mathbf{R}.P$, it can be that we can move "forward" or "backward" one step in the Feistel construction. This is captured by the functions next and prev. Additionally, the functions val^+ and val^- allow us to access additional values of the chain indexed by C, val$^+$ by invoking next, and val$^-$ by invoking prev. The function val finally gives us the same information in case we do not want to bother about the direction.

Definition 3.12. Fix a set of tables $G = \mathbf{T}.G$ and $P = \mathbf{R}.P$ in an execution of $\mathsf{S}_2(f, p)$. Let $C = (x_k, x_{k+1}, k)$ be a partial chain. We define the functions next, prev, val$^+$, val$^-$, and val with the following procedures (for a chain $C = (x_k, x_{k+1}, k)$, we let $C[1] = x_k$, $C[2] = x_{k+1}$ and $C[3] = k$):

```
procedure next($x_k, x_{k+1}, k$):
    if $k < 14$ then
        if $x_{k+1} \notin G_{k+1}$ then return $\perp$
        $x_{k+2} := x_k \oplus G_{k+1}(x_{k+1})$
        return $(x_{k+1}, x_{k+2}, k + 1)$
    else if $k = 14$ then
        if $(\uparrow, x_{14}, x_{15}) \notin P$ then return $\perp$
        $(x_0, x_1) := P(\uparrow, x_{14}, x_{15})$
        return $(x_0, x_1, 0)$

procedure prev($x_k, x_{k+1}, k$):
    if $k > 0$ then
        if $x_k \notin G_k$ then return $\perp$
        $x_{k-1} := x_{k+1} \oplus G_k(x_k)$
        return $(x_{k-1}, x_k, k - 1)$
    else if $k = 0$ then
        if $(\downarrow, x_0, x_1) \notin P$ then return $\perp$
        $(x_{14}, x_{15}) := P(\downarrow, x_0, x_1)$
        return $(x_{14}, x_{15}, 14)$

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

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

procedure val$_i(C)$
    if val$_i^+(C) \neq \perp$ return val$_i^+(C)$ else return val$_i^-(C)$
```

We use the convention that $\perp \notin G_i$ for any $i \in \{1, \ldots, 14\}$. Thus, the expression val$_i(C) \notin G_i$ means that val$_i(C) = \perp$ or that val$_i(C) \neq \perp$ and val$_i(C) \notin G_i$. Furthermore, even though next and prev may return $\perp$, according to our definition of partial chains, $\perp$ is not a partial chain.

Equivalent Partial Chains We use the concept of equivalent partial chains:

Definition 3.13. For a given set of tables G and P, two partial chains C and D are *equivalent* (denoted $C \equiv D$) if they are in the reflexive transitive closure of the relations given by next and prev.

In other words, two partial chains C and D are equivalent if $C = D$, or if D can be obtained by applying next and prev finitely many times on C.

Note that this relation is not an equivalence relation, since it is not necessarily symmetric.[9] However, we will prove that for most executions of $\mathsf{S}_2(f, p)$ it actually is symmetric and thus an equivalence relation. Furthermore, it is possible that two different chains $(x_0, x_1, 0)$ and $(y_0, y_1, 0)$ are equivalent (e.g., by applying next 15 times). While we eventually show that for most executions of $\mathsf{S}_2(f, p)$ this does not happen, this is not easy to show, and we cannot assume it for most of the following proof.

3.5.3. *Bad Events and Good Executions*

As usual in indistinguishability proofs, for some pairs (f, p) the scenario $\mathsf{S}_2(f, p)$ does not behave as "it should." In this section, we collect events which we show later to occur with low probability. We later study $\mathsf{S}_2(f, p)$ for pairs (f, p) for which these events do not occur.

All events occur if some unexpected collision happens to one of the partial chains which can be defined with elements of $G_1, \ldots, G_{14}$ and P.

Definition 3.14. The set of *table-defined partial chains* contains all chains C for which $\mathrm{next}(C) \neq \perp$ and $\mathrm{prev}(C) \neq \perp$.

If $C = (x_k, x_{k+1}, k)$ for $k \in \{1, \ldots, 13\}$, then C is table-defined if and only if $x_k \in G_k$ and $x_{k+1} \in G_{k+1}$. For $k \in \{0, 14\}$, C is table-defined if the "inner" value is in G_1 or G_{14}, respectively, and a corresponding triple is in P.

Hitting Permutations Whenever we call the two-sided random function, a query to the table p may occur. If such a query has unexpected effects, the event BadP occurs.

Definition 3.15. The event BadP occurs in an execution of $\mathsf{S}_2(f, p)$ if immediately after a call $(x_{14}, x_{15}) := p(\downarrow, x_0, x_1)$ in line 7 of **R** we have one of

- $(\uparrow, x_{14}, x_{15}) \in P$,
- $x_{14} \in G_{14}$.

Also, it occurs if immediately after a call $(x_0, x_1) := p(\uparrow, x_{14}, x_{15})$ in line 14 of **R** we have one of

[9] The symmetry can be violated if in the two-sided random function **R** an entry of the table P is overwritten.

- $(\downarrow, x_0, x_1) \in P$,
- $x_1 \in G_1$.

If BadP does not occur, then we will be able to show that evaluating P and P^{-1} is a bijection, since no value is overwritten.

Chains Hitting Tables Consider an assignment $G_i(x_i) := f(i, x_i)$ and a partial chain C that is table-defined after this assignment. Unless something unexpected happens, such an assignment allows evaluating $\mathrm{next}(C)$ at most *once* more.

Definition 3.16. The event BadlyHit occurs if one of the following happens in an execution of $\mathsf{S}_2(f, p)$:

- After an assignment $G_k(x_k) := f(k, x_k)$ there is a table-defined partial chain (x_k, x_{k+1}, k) such that $\mathrm{prev}(\mathrm{prev}(x_k, x_{k+1}, k)) \neq \perp$.
- After an assignment $G_k(x_k) := f(k, x_k)$ there is a table-defined partial chain $(x_{k-1}, x_k, k-1)$ such that $\mathrm{next}(\mathrm{next}(x_{k-1}, x_k, k-1)) \neq \perp$.

Furthermore, if the above happens for some partial chain C, and C' is a partial chain equivalent to C before the assignment, we say that C' badly hits the tables.

To illustrate the definition, we give two examples. First, the event BadlyHit occurs if $x_1 \in G_1$, the assignment $G_2(x_2) := f(2, x_2)$ occurs, and just after that we have $x_3 := x_1 \oplus G_2(x_2) \in G_3(x_3)$. Clearly $(x_1, x_2, 1)$ is table-defined after the assignment, $\mathrm{next}(x_1, x_2, 1) = (x_2, x_3, 2)$, and $\mathrm{next}(x_2, x_3, 2) \neq \perp$ because $x_3 \in G_3$. Second, the event BadlyHit occurs if $x_2 \in G_2$, the assignment $G_1(x_1) := f(1, x_1)$ occurs, and just after that we have for $x_0 := G_1(x_1) \oplus x_2$ that $(\downarrow, x_0, x_1) \in \mathrm{P}$. Clearly $(x_1, x_2, 1)$ is table-defined after the assignment, $\mathrm{prev}(x_1, x_2, 1) = (x_0, x_1, 0)$, and $\mathrm{prev}(x_0, x_1, 0) \neq \perp$ because $(\downarrow, x_0, x_1) \in \mathrm{P}$.

We will later argue that the event BadlyHit is unlikely, because a chain only badly hits the tables if $f(k, x_k)$ takes a very particular value. For this (and similar statements), it is useful to note that the set of table-defined chains after an assignment $G_k(x_k) := f(k, x_k)$ *does not depend* on the value of $f(k, x_k)$, as the reader can verify.

Colliding Chains Two chains C and D collide if after an assignment suddenly $\mathrm{val}_i(C) = \mathrm{val}_i(D)$, even though this was not expected. More exactly:

Definition 3.17. The event BadlyCollide occurs in an execution of $\mathsf{S}_2(f, p)$, if for an assignment of the form $G_i(x_i) := f(i, x_i)$ there exist two partial chains C and D such that for some $\ell \in \{0, \dots, 15\}$ and $\sigma, \rho \in \{+, -\}$ all of the following happen:

- Before the assignment, C and D are not equivalent.
- Before the assignment, $\mathrm{val}_\ell^\sigma(C) = \perp$ or $\mathrm{val}_\ell^\rho(D) = \perp$.
- After the assignment, $\mathrm{val}_\ell^\sigma(C) = \mathrm{val}_\ell^\rho(D) \neq \perp$.
- After the assignment, C and D are table-defined.

Finally, we say that a pair (f, p) is *good* if none of the above three events happen in an execution of $\mathsf{S}_2(f, p)$.

To illustrate the above definition, we give three examples. In all of these, we assume that the tables G and P only contain the values we explicitly mention.

In the first example, suppose $(x_1, x_3, x_4) \in G_1 \times G_3 \times G_4$, the assignment $G_2(x_2) := f(2, x_2)$ occurs, and just after this assignment, we have $x_1 \oplus G_2(x_2) = x_3$. In this case the event BadlyCollide occurs for the given assignment and the chains $C = (x_1, x_2, 1)$, $D = (x_3, x_4, 3)$: before the assignment, C and D are not equivalent because $\text{next}(x_1, x_2) = \perp$, and $\text{val}_3^+(C) = \perp, \text{val}_3^-(D) = x_3$. After the assignment C and D are table-defined, and $\text{val}_3^+(C) = \text{val}_3^-(D) = x_3$. (In this case, C and D are actually equivalent after the assignment.)

For the second example, suppose $(x_1, x_4, x_5) \in G_1 \times G_4 \times G_5$, let $x_3 := G_4(x_4) \oplus x_5$, the assignment $G_2(x_2) := f(2, x_2)$ occurs, and just after this assignment we have $x_1 \oplus G_2(x_2) = x_3$. In this case the event BadlyCollide occurs for the given assignment and the chains $C = (x_1, x_2, 1)$, $D = (x_4, x_5, 4)$: before the assignment, C and D are not equivalent because $\text{next}(x_1, x_2) = \perp$, and $\text{val}_3^+(C) = \perp, \text{val}_3^-(D) = x_3$. After the assignment C and D are table-defined, and $\text{val}_3^+(C) = \text{val}_3^-(D) = x_3$. (In this case, C and D are *not* equivalent after the assignment.)

In the third example, let x_1, x_1', x_0, x_0' be such that $x_1 \neq x_1'$ and $x_0 \neq x_0'$. Suppose $x_1 \in G_1$, $(\downarrow, x_0, x_1), (\downarrow, x_0', x_1') \in$ P, let $x_2 := x_0 \oplus G_1(x_1)$, the assignment $G_1(x_1') := f(1, x_1')$ occurs, and just after this assignment we have $x_0' \oplus G_1(x_1') = x_2$. In this case the event BadlyCollide occurs for the given assignment and the chains $C = (x_0, x_1, 0)$, $D = (x_0', x_1', 0)$: before the assignment, C and D are not equivalent and $\text{val}_2^+(C) = x_2, \text{val}_2^+(D) = \perp$. After the assignment C and D are table-defined, and $\text{val}_2^+(C) = \text{val}_2^+(D) = x_2$. (Also in this case, C and D are *not* equivalent after the assignment.)

3.5.4. *Bad Events are Unlikely*

In this subsection, we show that all the bad events we have introduced are unlikely.

Hitting Permutations We first show that the event BadP is unlikely.

Lemma 3.18. *Suppose for some $T \in \mathbb{N}$, $\mathsf{S}_2(f, p)$ is such that for any (f, p) the tables satisfy $|G_i| \leq T$ for all i and $|P| \leq T$ at any point in the execution. Then, the probability over the choice of (f, p) of the event* BadP *is at most $\frac{2T^2}{2^n}$.*

Proof. For any query to p, only 2 events are possible. In both cases, these events have probability at most $\frac{T}{2^n}$. Since at most T positions of p can be accessed without violating $|P| \leq T$ we get the claim. □

Chains Hitting Tables We now show that the event BadlyHit is unlikely.

Lemma 3.19. *Suppose for some $T \in \mathbb{N}$, $\mathsf{S}_2(f, p)$ is such that for any (f, p) the tables satisfy $|G_i| \leq T$ for all i and $|P| \leq T$ at any point in the execution. Then, the probability over the choice of (f, p) of the event* BadlyHit *is at most $30\frac{T^3}{2^n}$.*

Proof. We first bound the probability of the first event, i.e., that after the assignment $G_k(x_k) := f(k, x_k)$, there is a table-defined chain $C = (x_k, x_{k+1}, k)$ such that

$\text{prev}(\text{prev}(C)) \neq \bot$. This can only happen if $x_{k+1} \oplus G_k(x_k)$ has one of at most T different values (namely, it has to be in G_{k-1} in case $14 \geq k \geq 2$ or in P together with x_1 in case $k = 1$). Thus, for fixed $x_{k+1} \in G_{k+1}$, the probability that $\text{prev}(\text{prev}(C)) \neq \bot$ is at most $T/2^n$. Since there are at most T possible choices for x_{k+1} (this also holds if $k = 14$) the total probability is at most $T^2/2^n$.

The analogous probability for next is exactly the same and thus the probability of BadlyHit for one assignment is at most $2 \cdot T^2/2^n$. In total, there are at most $14 \cdot T$ assignments of the form $G_k(x_k) := f(k, x_k)$, and thus the probability of BadlyHit is at most $28T^3/2^n$. □

Colliding Chains We next show that it is unlikely that chains badly collide. First, we give a useful lemma which explains how the chains behave when they do not badly hit G: for each $\sigma \in \{+, -\}$, at most one value $\text{val}_i^\sigma(C)$ can change from $\bot$ to a different value.

Lemma 3.20. *Consider a set of tables G and P, $x_k \notin G_k$, fix a partial chain C, and suppose that C does not badly hit the tables due to the assignment $G_k(x_k) := f(k, x_k)$, and C is table-defined after the assignment. Then, for each $\sigma \in \{+, -\}$ there is at most one value i such that $\text{val}_i^\sigma(C)$ differs after the assignment from before the assignment. Furthermore, if some value changes, then it changes from $\bot$ to a different value, and*

$$i = \begin{cases} k+1 & \text{if } \sigma = + \\ k-1 & \text{if } \sigma = -, \end{cases}$$

and $\text{val}_k^\sigma(C) = x_k$ *before the assignment.*

Proof. We give the proof for $\sigma = +$, the other case is symmetric. First, we see that if $\text{val}_i^+(C) \neq \bot$ before the assignment, then it does not change due to the assignment. This follows by induction on the number of calls to next in the evaluation of val^+, and by noting that $G_k(x_k) := f(k, x_k)$ is not called if $x_k \in G_k$ in the simulator.

Thus, suppose that $\text{val}_i^+(C) = \bot$. This means that during the evaluation of $\text{val}_i^+(C)$ at some point the evaluation stopped. This was either because a queried triple was not in P, or because a value x_j was not in G_j during the evaluation. In the first case, the evaluation of $\text{val}_i^+(C)$ will not change due to an assignment to $G_k(x_k)$. In the second case, the evaluation can only change if it stopped because $\text{val}_k^+(C) = x_k$. Then after the assignment, $\text{val}_{k+1}^+(C)$ will change from $\bot$ to a different value. Since C is table-defined after the assignment and does not badly hit the tables under the assignment, $\text{val}_{k+1}^+(C) \notin G_{k+1}$ after this assignment (in case $k+1 < 15$), and $(\uparrow, \text{val}_{14}^+(C), \text{val}_{15}^+(C)) \notin P$ (in case $k+1 = 15$). Thus, there is only one change in the evaluation. □

Instead of showing that BadlyCollide is unlikely, it is slightly simpler to consider the event (BadlyCollide $\wedge$ ¬BadlyHit $\wedge$ ¬BadP).

Lemma 3.21. *Suppose for some $T \in \mathbb{N}$, $\mathsf{S}_2(f, p)$ is such that for any (f, p) the tables satisfy $|G_i| \leq T$ for all i and $|P| \leq T$ at any point in the execution. Then, the probability of the event* (BadlyCollide $\wedge$ ¬BadlyHit $\wedge$ ¬BadP) *is at most* $17\,000\frac{T^5}{2^n}$.

Proof. If the event $(\mathsf{BadlyCollide} \wedge \neg\mathsf{BadlyHit} \wedge \neg\mathsf{BadP})$ happens for a pair (f, p), then there is some point in the execution where some assignment $G_k(x_k) := f(k, x_k)$ makes a pair (C, D) of partial chains collide as in Definition 3.17. After this assignment, both (C, D) are table-defined, and $\mathrm{val}_\ell^\sigma(C) = \mathrm{val}_\ell^\rho(D)$.

We distinguish some cases: first suppose that $\mathrm{val}_\ell^-(C) = \mathrm{val}_\ell^-(D) = \perp$ before the assignment, and $\mathrm{val}_\ell^-(C) = \mathrm{val}_\ell^-(D) \neq \perp$ after the assignment. Since $\mathsf{BadlyHit}$ does not happen, Lemma 3.20 implies that before the assignment, $\mathrm{val}_{\ell+1}^-(C) = \mathrm{val}_{\ell+1}^-(D)$, and furthermore $\ell + 1 \in \{1, \ldots, 14\}$. Also, since $C \not\equiv D$ before the assignment, it must be that before the assignment $\mathrm{val}_{\ell+2}^-(C) \neq \mathrm{val}_{\ell+2}^-(D)$. However, this implies that $\mathrm{val}_\ell^-(C) \neq \mathrm{val}_\ell^-(D)$ after the assignment. Therefore, this case is impossible and has probability 0.

Next, we consider the case $\mathrm{val}_\ell^-(C) = \perp$, $\mathrm{val}_\ell^-(D) \neq \perp$ before the assignment, and $\mathrm{val}_\ell^-(C) = \mathrm{val}_\ell^-(D)$ after the assignment. Since D is table-defined after the assignment, and we assume $\mathsf{BadlyHit}$ does not occur, by Lemma 3.20 the value $\mathrm{val}_\ell^-(D)$ does not change due to the assignment. Since $\mathrm{val}_\ell^-(C) = \mathrm{val}_{\ell+2}^-(C) \oplus G_{\ell+1}(x_{\ell+1})$, and $G_{\ell+1}(x_{\ell+1})$ is chosen uniformly at random, the probability that it exactly matches $\mathrm{val}_\ell^-(D)$ is 2^{-n}.

The next two cases are similar to the previous ones, we give them for completeness. The first of these two is that $\mathrm{val}_\ell^+(C) = \mathrm{val}_\ell^-(D) = \perp$ before the assignment, and $\mathrm{val}_\ell^+(C) = \mathrm{val}_\ell^-(D) \neq \perp$ after the assignment. However, due to Lemma 3.20 this is impossible: we would need both $k = \ell + 1$ and $k = \ell - 1$ for both values to change as needed.

Then, we have the case that $\perp = \mathrm{val}_\ell^+(C) \neq \mathrm{val}_\ell^-(D)$ before the assignment, and $\mathrm{val}_\ell^+(C) = \mathrm{val}_\ell^-(D)$ after the assignment. Again, $\mathrm{val}_\ell^-(D)$ does not change by the assignment by Lemma 3.20, and also similarly to before, the probability that $\mathrm{val}_{\ell-2}^+(C) \oplus f(\ell - 1, \mathrm{val}_{\ell-1}^-(C)) = \mathrm{val}_\ell^+(D)$ is 2^{-n}.

Bounds on the probability of the 4 remaining cases follow by symmetry of the construction.

There are 4 possibilities for the values of σ and ρ. As previously, there can be at most $14 \cdot T$ assignments of the form $G_k(x_k) := f(k, x_k)$. For each assignment, there are at most $15 \cdot T^2$ possibilities for a chain to be table-defined before the assignment. Since the chains that are table-defined after the assignment, but not before must involve x_k, there are at most $2 \cdot T$ possibilities for a fixed assignment. Thus the probability of the event $(\mathsf{BadlyCollide} \wedge \neg\mathsf{BadlyHit} \wedge \neg\mathsf{BadP})$ is at most $\frac{4 \cdot 14 \cdot T \cdot (15 \cdot T^2 + 2 \cdot T)^2}{2^n} \leq \frac{4 \cdot 14 \cdot 17^2 \cdot T^5}{2^n}$. □

Most Executions are Good We collect our findings in the following lemma:

Lemma 3.22. *Suppose for some $T \in \mathbb{N}$, $\mathsf{S}_2(f, p)$ is such that for any (f, p) the tables satisfy $|G_i| \leq T$ for all i and $|P| \leq T$ at any point in the execution. Then, the probability that a uniform randomly chosen (f, p) is not good is at most* $18\,000 \cdot \frac{T^5}{2^n}$.

Proof. This follows immediately from Lemmas 3.18, 3.19, and 3.21. □

3.5.5. *Properties of Good Executions*

We now study executions of $\mathsf{S}_2(f, p)$ with good pairs (f, p). One of the main goals of this section is to prove Lemma 3.31, which states that no call to ForceVal overwrites a previous entry. However, we later also use Lemma 3.32 (in good executions, evaluating the Feistel construction for a pair (x_0, x_1) leads to $P(x_0, x_1)$—if not, it would be silly to hope that our simulator emulates a Feistel construction), and Lemma 3.33 (the number of times Adapt is called in $\mathbf{T}(f)$ is exactly the same as the number of times the table p is queried in $\mathbf{R}(p)$).

We first state two basic lemmas about good executions:

Lemma 3.23. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p)*. Then, we have:*

(a) For any partial chain C, if $\text{next}(C) = \perp$ *before an assignment* $G_i(x_i) := f(i, x_i)$ *or a pair of assignments to P in* $\mathbf{R}$*, then if C is table-defined after the assignment(s),* $\text{next}(\text{next}(C)) = \perp$.
For any partial chain C, if $\text{prev}(C) = \perp$ *before an assignment* $G_i(x_i) := f(i, x_i)$ *or a pair of assignments to P in* $\mathbf{R}$*, then if C is table-defined after the assignment(s),* $\text{prev}(\text{prev}(C)) = \perp$.
(b) For all partial chains C and D, we have $\text{next}(C) = D \iff \text{prev}(D) = C$.
(c) The relation $\equiv$ between partial chains is an equivalence relation.

Proof. For assignments of the form $G_i(x_i) := f(i, x_i)$, (a) follows directly since $\mathsf{BadlyHit}$ does not occur. For the assignments to P, it follows because BadP does not occur.

The statement (b) is trivial for chains $C = (x_k, x_{k+1}, k)$ with $k \in \{0, \ldots, 13\}$, since evaluating the Feistel construction one step forward or backward is bijective. For $k = 14$ we get (b) because BadP does not occur: no value is ever overwritten in a call to P or P^{-1}, and thus evaluating P and P^{-1} is always bijective.

To see (c), observe that the relation $\equiv$ is symmetric because of (b), and it is reflexive and transitive by definition. □

Lemma 3.24. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p)*. Suppose that at any point in the execution, two table-defined chains C and D are equivalent. Then, there exists a sequence of partial chains $C_1, \ldots, C_r$, $r \geq 1$, such that*

- *$C = C_1$ and $D = C_r$, or else $D = C_1$ and $C = C_r$,*
- *$C_i =$* $\text{next}(C_{i-1})$ *and $C_{i-1} =$* $\text{prev}(C_i)$*,*
- *and each C_i is table-defined.*

Proof. Since $C \equiv D$, D can be obtained from C by applying next and prev finitely many times. A shortest such sequence can only apply either next or prev, due to Lemma 3.23 (b). The resulting sequence of chains is the sequence we are looking for (possibly backwards)—note that the last bullet point also follows by Lemma 3.23 (b). □

We first show that assignments $G_i(x_i) := f(i, x_i)$ and also assignments to P in $\mathbf{R}$ do not change the equivalence relation for chains which were defined before.

Lemma 3.25. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p)*. Let* C *and* D *be two table-defined partial chains at some point in the execution. Suppose that after this point, there is an assignment* $G_i(x_i) := f(i, x_i)$ *or a pair of assignments to* P *in* $\mathbf{R}$*. Then* $C \equiv D$ *before the assignment(s) if and only if* $C \equiv D$ *after the assignment(s).*

Proof. Suppose that $C \equiv D$ before the assignment. We apply Lemma 3.24 to get a sequence $C_1, \ldots, C_r$ of table-defined chains. This sequence still implies equivalence after the assignment, since no value in P or G can be overwritten by one of the assignments considered (recall that BadP does not occur), i.e., the conditions of Definition 3.13 still hold if they held previously, thus $C \equiv D$ after the assignment(s).

Now suppose that C and D are equivalent after the assignment. Again consider the sequence $C_1, \ldots, C_r$ as given by Lemma 3.24. Suppose first that the assignment was $G_i(x_i) := f(i, x_i)$. If x_i was not part of any chain, then $C_1, \ldots, C_r$ are a sequence which show the equivalence of C and D before the assignment. Otherwise, there is j such that the chains C_{j-1} and C_j have the form $C_{j-1} = (x_{i-1}, x_i, i-1)$ and $C_j = (x_i, x_{i+1}, i)$. It is not possible that $C_j = C_r$, as C_j is not table-defined before the assignment. After the assignment $\text{next}(\text{next}(C_{j-1})) \neq \perp$ which is impossible by Lemma 3.23 (a). Suppose now we have a pair of assignments to P, mapping (x_0, x_1) to (x_{14}, x_{15}). If $(x_{14}, x_{15}, 14)$ is not part of the sequence connecting C and D after the assignment, the same sequence shows equivalence before the assignment. Otherwise, $\text{next}(\text{next}(x_{14}, x_{15}, 14)) = \perp$ by Lemma 3.23 (a), as before. □

Next, we show that calls to ForceVal *also* do not change the equivalence relation for previously defined chains. Also, they never overwrite a previously defined value. However, we only show this under the assumption $x_{\ell-1} \notin G_{\ell-1}$ and $x_{\ell+2} \notin G_{\ell+2}$. Later, we will see that this assumption is safe.

Lemma 3.26. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p)*. Let* $\ell \in \{4, 10\}$ *and suppose that for a call* Adapt$(x_{\ell-2}, x_{\ell-1}, x_{\ell+2}, x_{\ell+3}, \ell)$ *it holds that* $x_{\ell-1} \notin G_{\ell-1}$ *and* $x_{\ell+2} \notin G_{\ell+2}$ *before the call.*

Then, the following properties hold:

(a) For both calls ForceVal$(x, \cdot, j)$ *we have* $x \notin G_j$ *before the call.*
(b) Let C *be a table-defined chain before the call to* Adapt*,* $i \in \{1, \ldots, 14\}$*. Then,* $\text{val}_i(C)$ *stays constant during both calls to* ForceVal*.*
(c) If the chains C *and* D *are table-defined before the call to* Adapt*, then* $C \equiv D$ *before the calls to* ForceVal *if and only if* $C \equiv D$ *after the calls to* ForceVal*.*

Proof. Before Adapt is called, EvaluateForward and EvaluateBackward make sure that all the values $x_{\ell-1}, x_{\ell-2}, \ldots, x_0, x_{15}, \ldots, x_{\ell+3}, x_{\ell+2}$ corresponding to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ are defined in P and G. By Lemma 3.23 (b), all partial chains defined by these values are equivalent to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$. Furthermore, since $x_{\ell-1} \notin G_{\ell-1}$ and $x_{\ell+2} \notin G_{\ell+2}$, these are the only partial chains that are equivalent to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ at this point.

By our assumption, $x_{\ell-1} \notin G_{\ell-1}$ and $x_{\ell+2} \notin G_{\ell+2}$, and thus the procedure Adapt defines $G_{\ell-1}(x_{\ell-1}) := f(\ell-1, x_{\ell-1})$ and $G_{\ell+2}(x_{\ell+2}) := f(\ell+2, x_{\ell+2})$. These assignments lead to $x_\ell \notin G_\ell$ and $x_{\ell+1} \notin G_{\ell+1}$, as otherwise the event $\mathsf{BadlyHit}$ would occur. This shows (a).

We next show (b), i.e., for any C the values $\mathrm{val}_i(C)$ stay constant. For this, note first that this is true for table-defined chains C that are equivalent to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ before the call to ADAPT: val_i gives exactly x_i both before and after the calls to FORCEVAL.

Now consider the table-defined chains that are not equivalent to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ before the call to ADAPT. We show that for such a chain C, even $\mathrm{val}_i^+(C)$ and $\mathrm{val}_i^-(C)$ stay constant, as otherwise $\mathsf{BadlyCollide}$ would occur. A value $\mathrm{val}_i^\sigma(C)$ can only change during the execution of $\text{FORCEVAL}(x_\ell, \cdot, \ell)$ if $\mathrm{val}_\ell^\sigma(C) = x_\ell$. But this implies that the assignment $G(x_{\ell-1}) := f(\ell-1, x_{\ell-1})$ in ADAPT made the two partial chains C and $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ badly collide. For this, note that C is table-defined even before the assignment, since it was table-defined before the call to ADAPT. Moreover, $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ is table-defined after the assignment. The argument for $\text{FORCEVAL}(x_{\ell+1}, \cdot, \ell+1)$ is the same. Thus, this establishes (b).

We now show (c). First, suppose that $C \equiv D$ before the calls to FORCEVAL. The sequence of chains given by Lemma 3.24 is not changed during the calls to FORCEVAL, since by (a), no value is overwritten. Thus, the chains are still equivalent after the calls.

Now suppose that $C \equiv D$ after the calls to FORCEVAL. As by Lemma 3.23 (c), $\equiv$ is symmetric, we may distinguish the following two cases. In the first case, C and D are equivalent to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$. By definition of ADAPT, the only partial chains equivalent to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$ after the ADAPT call are the partial chains with the values $x_0, x_1, \ldots, x_{14}, x_{15}$ corresponding to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$. Only the partial chains on the values $x_{\ell-2}, \ldots, x_0, x_{15}, \ldots, x_{\ell+3}$ were table-defined before the call to ADAPT, and as we observed in the first paragraph, these chains were equivalent before the calls to FORCEVAL.

In the second case, C and D are not equivalent to $(x_{\ell-2}, x_{\ell-1}, \ell-2)$. Let $C_1, \ldots, C_r$ be the sequence given by Lemma 3.24. If C and D were not equivalent before the calls to FORCEVAL, there is i such that before the call, C_i was table-defined, but C_{i+1} was not. Then, $\mathrm{val}^+(C_i)$ changes during a call to FORCEVAL, contradicting the proof of (b). Thus, the chains must have been equivalent before the calls. □

Equivalent chains are put into CompletedChains simultaneously:

Lemma 3.27. *Suppose that (f, p) is good. Fix a point in the execution of $\mathsf{S}_2(f, p)$, and suppose that until this point, for no call to* FORCEVAL *of the form* $\text{FORCEVAL}(x, \cdot, \ell)$ *we had $x \in G_\ell$ before the call. Suppose that at this point $C = (x_k, x_{k+1}, k)$ with $k \in \{1, 7\}$ and $D = (y_m, y_{m+1}, m)$ with $m \in \{1, 7\}$ are equivalent. Then, $C \in$* CompletedChains *if and only if $D \in$* CompletedChains.

Proof. We may assume $k = 1$. We first show that the lemma holds right after C was added to CompletedChains. Since the chain was just adapted, and using Lemma 3.23 (b), the only chains which are equivalent to C are those of the form $(\mathrm{val}_i(C), \mathrm{val}_{i+1}(C), i)$. Thus both C and D are added to CompletedChains, and D is the only chain with index $m = 7$ that is equivalent to C.

Now, the above property can only be lost if the event BadP occurs or else if a value is overwritten by FORCEVAL. Thus, we get the lemma. □

If the simulator detects a chain $(x_7, x_8, 7)$ for which val^+ is defined for sufficiently many values, a chain equivalent to it was previously enqueued:

Lemma 3.28. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p)*. Suppose that at some point, a chain* $C = (x_7, x_8, 7)$ *is enqueued for which* $\mathrm{val}_2^+(C) \in G_2$ *or* $\mathrm{val}_{13}^-(C) \in G_{13}$*. Then, there is a chain equivalent to C which was previously enqueued.*

Proof. We only consider the case $\mathrm{val}_2^+(C) \in G_2$, the other case is symmetric. Define $(x_0, x_1, x_2, x_{13}, x_{14}, x_{15}) := (\mathrm{val}_0^+(C), \mathrm{val}_1^+(C), \mathrm{val}_2^+(C), \mathrm{val}_{13}^+(C), \mathrm{val}_{14}^+(C), \mathrm{val}_{15}^+(C))$. All these must be different from $\perp$, since otherwise $\mathrm{val}_2^+(C) = \perp$.

At some point in the execution, all the following entries are set in their respective hashtables: $G_1(x_1)$, $G_2(x_2)$, $G_{13}(x_{13})$, $G_{14}(x_{14})$, and $P(\uparrow, x_{14}, x_{15})$. The last one of these must have been $G_2(x_2)$ or $G_{13}(x_{13})$: if it was $P(\uparrow, x_{14}, x_{15})$, then the event BadP must have happened. If it was $G_1(x_1)$, then the event $\mathsf{BadlyHit}$ must have happened (as $(x_0, x_1, 0)$ is table-defined after the assignment). Analogously, $G_{14}(x_{14})$ cannot have been the last one. Thus, since $G_2(x_2)$ or $G_{13}(x_{13})$ was defined last among those, the simulator will detect the chain and enqueue it. □

If a chain C is enqueued for which previously no equivalent chain has been enqueued, then the assumptions of Lemma 3.26 actually *do* hold in good executions. The following two lemmas state that these assumptions hold at the moment the chains are enqueued (this is captured by Lemma 3.29), and then that they still hold when the chains are dequeued (this is captured by Lemma 3.30).

Lemma 3.29. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p)*. Let C be a partial chain which is enqueued in the execution at some time and to be adapted at position* ℓ*. Suppose that at the moment the chain is enqueued, no equivalent chain has been previously enqueued.*

Then, before the assignment $G_k(x_k) := f(k, x_k)$ *happens which just precedes C being enqueued,* $\mathrm{val}_{\ell-1}(C) = \perp$ *and* $\mathrm{val}_{\ell+2}(C) = \perp$.

Proof. We have $\ell \in \{4, 10\}$. We will assume $\ell = 4$, and due to symmetry of the construction, this also implies the lemma in case $\ell = 10$ for the corresponding rounds.

The assignment sets either the value of $G_7(x_7)$ or $G_2(x_2)$ uniformly at random (otherwise, ENQUEUENEWCHAINS is not called in the simulator). Consider first the case that $G_2(x_2)$ was just set. Then, before this happened, $\mathrm{val}_3^+(C) = \perp$, since $x_2 \notin G_2$. Furthermore, $\mathrm{val}_6^-(C) = \perp$, since otherwise, $\mathrm{val}_7^-(C) \in G_7$, and then $(\mathrm{val}_7^-(C), \mathrm{val}_8^-(C), 7)$ would be an equivalent, previously enqueued chain. This implies the statement in case $G_2(x_2)$ is just set. The second case is if $G_7(x_7)$ was just set. Then, before the assignment, $\mathrm{val}_6^-(C) = \perp$, as $x_7 \notin G_7$, and $\mathrm{val}_3^+(C) = \perp$, since otherwise $\mathrm{val}_2^+(C) \in G_2$ and so an equivalent chain would have been previously enqueued, according to Lemma 3.28. □

Lemma 3.30. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p)*. Let C be a partial chain which is enqueued in the execution at some time and to be adapted at position* ℓ*. Then, either (i) or (ii) holds:*

(i) At the moment C is dequeued, we have $C \in \mathrm{CompletedChains}$.

(ii) At the moment C is dequeued, we have $C \notin \text{CompletedChains}$, and just before the call to Adapt *for C, we have $(\text{val}_{\ell-1}(C) \notin G_{\ell-1}) \wedge (\text{val}_{\ell+2}(C) \notin G_{\ell+2})$.*

Proof. Suppose that the lemma is wrong, and let C be the first chain for which it fails. Because this is the first chain for which it fails, Lemma 3.26(a) implies that until the moment C is dequeued, no call to ForceVal overwrote a value. Now, consider the set $\mathfrak{C}$ of table-defined chains at some point in the execution that is not in an Adapt call, and before C is dequeued. Because of Lemmas 3.25 and 3.26(c), the equivalence relation among chains in $\mathfrak{C}$ stays constant from this point until the moment C is dequeued.

We distinguish two cases to prove the lemma. Consider first the case that at the moment C is enqueued, an equivalent chain D was previously enqueued. The point in the execution where C is enqueued is clearly not in an Adapt call, and both C and D are table-defined. Then, at the moment C is dequeued, clearly $D \in \text{CompletedChains}$. Thus, because of Lemma 3.27 and the remark about equivalence classes of $\mathfrak{C}$ above, this implies that $C \in \text{CompletedChains}$ when it is dequeued.

The second case is when C has no equivalent chain which was previously enqueued. It is sufficient to show $(\text{val}_{\ell-1}(C) \notin G_{\ell-1}) \wedge (\text{val}_{\ell+2}(C) \notin G_{\ell+2})$ at the moment C is dequeued: this property still holds after executing EvaluateBackward and EvaluateForward, as otherwise BadlyHit or BadP occurs.

To simplify notation we assume $\ell = 4$ and show $\text{val}_3(C) \notin G_3$, but the argument is completely generic. From Lemma 3.29 we get that before the assignment which led to C being enqueued, $\text{val}_3(C) = \perp$. Suppose $\text{val}_3(C) \in G_3$ at the time C is dequeued. This cannot have been true just after the assignment which led to C being enqueued, as this would imply that BadlyHit occurred. So it must be that $G_3(\text{val}_3(C))$ was set during completion of a chain D. This chain D was enqueued before C was enqueued and dequeued after C was enqueued. Also, at the moment C is dequeued, $\text{val}_3(C) = \text{val}_3(D)$. From the point C is enqueued, at any point until C is dequeued, it is not possible that $C \equiv D$: We assumed that there is no chain in the queue that is equivalent to C when C is enqueued, and at the point C is enqueued both C and D are table-defined. Furthermore, this point in the execution is not during an Adapt call. Therefore, by our initial remark, the equivalence relation between C and D stays constant until the moment C is dequeued.

Consider the last assignment to a table before $\text{val}_3(C) = \text{val}_3(D) \neq \perp$ was true. We first argue that this assignment cannot have been of the form $G_i(x_i) := f(i, x_i)$, as otherwise the event BadlyCollide would have happened. To see this, we check the conditions for BadlyCollide for C and D. By Lemma 3.29, the assignment happens earliest right before C is enqueued, in which case C is table-defined after the assignment. Since D is enqueued before C, also D is table-defined after the assignment. If the assignment happens later, both C and D are table-defined even before the assignment. Furthermore, we have already seen that $C \equiv D$ is not possible. Clearly, $\text{val}_3(C) = \perp$ or $\text{val}_3(D) = \perp$ before the assignment, and $\text{val}_3(C) = \text{val}_3(D) \neq \perp$ after the assignment.

The assignment cannot have been of the form $P(\downarrow, x_0, x_1) = (x_{14}, x_{15})$ or $P(\uparrow, x_{14}, x_{15}) = (x_0, x_1)$, since val can be evaluated at most one step further by Lemma 3.23(a). Finally, the assignment cannot have been in a call to ForceVal, because of Lemma 3.26(b).

Thus, $\mathrm{val}_3(C) \notin G_3$ when C is dequeued, and the same argument holds for the other cases as well. □

The following lemma is an important intermediate goal. It states that the simulator never overwrites a value in G in case (f, p) is good.

Lemma 3.31. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p)*. Then, for any call to* ForceVal *of the form* $\textsc{ForceVal}(x, \cdot, \ell)$ *we have* $x \notin G_\ell$ *before the call.*

Proof. Assume otherwise, and let C be the first chain during completion of which the lemma fails. Since the lemma fails for C, $C \notin$ CompletedChains when it is dequeued. Thus, Lemma 3.30 implies that $\mathrm{val}_{\ell-1}(C) \notin G_{\ell-1}$ and $\mathrm{val}_{\ell+2}(C) \notin G_{\ell+2}$ just before Adapt is called for C, and so by Lemma 3.26(a) we get the result. □

We say that a *distinguisher completes all chains*, if, at the end of the execution, it emulates a call to $\textsc{EvaluateForward}(x_0, x_1, 0, 14)$ for all queries to $\mathrm{P}(x_0, x_1)$ or to $(x_0, x_1) = \mathrm{P}^{-1}(x_{14}, x_{15})$ which it made during the execution.

Lemma 3.32. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p) *in which the distinguisher completes all chains. Suppose that during the execution* $\mathrm{P}(x_0, x_1)$*, resp.* $\mathrm{P}^{-1}(x_{14}, x_{15})$ *is queried by the simulator or the distinguisher. Then, at the end of the execution it holds that* $P(\downarrow, x_0, x_1) = \big(\mathrm{val}^+_{14}(x_0, x_1, 0), \mathrm{val}^+_{15}(x_0, x_1, 0)\big)$*, resp.* $P(\uparrow, x_{14}, x_{15}) = \big(\mathrm{val}^-_0(x_{14}, x_{15}, 14), \mathrm{val}^-_1(x_{14}, x_{15}, 14)\big)$.

Proof. If the query $\mathrm{P}(x_0, x_1)$ was made by the simulator at some point, then this was while it was completing a chain. Then, right after it finished adapting we clearly have the result. By Lemma 3.31 no value is ever overwritten. Since the event BadP does not occur, the conclusion of the lemma must also be true at the end of the execution.

Consider the case that $\mathrm{P}(x_0, x_1)$ was queried by the distinguisher at some point. Since it eventually issues the corresponding Feistel queries, it must query the corresponding values x_7 and x_8 at some point. Thus, $x_7 \in G_7$ and $x_8 \in G_8$ at the end of the execution. One of the two values was defined later, and in that moment, $(x_7, x_8, 7)$ was enqueued by the simulator. Thus, it is dequeued at some point. If it was not in CompletedChains at this point, it is now completed and the conclusion of the lemma holds right after this completion. Otherwise, it was completed before it was inserted in CompletedChains, and the conclusion of the lemma holds after this completion. Again, by Lemma 3.31 no value is ever overwritten, and again BadP never occurs; hence, the conclusion also holds at the end of the execution.

The case of a query $\mathrm{P}^{-1}(x_{14}, x_{15})$ is handled in the same way. □

Lemma 3.33. *Consider an execution of* $\mathsf{S}_2(f, p)$ *with a good pair* (f, p) *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 the two-sided random function.*

Proof. Since the event BadP does not occur, the number of queries to $p(\cdot, \cdot, \cdot)$ equals half the number of entries in P at the end of the execution.

For each call to Adapt, there is a corresponding pair of entries in P: just before Adapt was called, such an entry was read either in EvaluateForward or EvaluateBackward. Furthermore, for no other call to Adapt the same entry was read, as otherwise a value would have to be overwritten, contradicting Lemma 3.31.

For each query to $p(\cdot, \cdot, \cdot)$, there was a corresponding call to Adapt: if the query to p occurred in a call to P by the simulator, then we consider the call to Adapt just following this call (as the simulator only queries P right before it adapts). If the query to p occurred in a call by the distinguisher, the distinguisher eventually queries the corresponding Feistel chain. At the moment it queries $G_8(x_8)$, consider the set of chains that have been enqueued until now and are equivalent to $(x_7, x_8, 7)$ at this point. Let C be the chain that was enqueued first among the chains in this set. By Lemma 3.31 no values are overwritten, and thus the query to p made in one of EvaluateBackward or EvaluateForward during the completion of C is exactly the query we are interested in, and we associate the subsequent Adapt call to it. □

3.5.6. *Mapping Randomness of* S_2 *to Randomness of* S_3

We next define a map τ which maps a pair of tables (f, p) either to the special symbol λ in case (f, p) is not good, or to a partial table h. A partial table $h : \{1, \dots, 14\} \times \{0, 1\}^n \mapsto \{0, 1\}^n \cup \{\perp\}$ 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 $\mathsf{S}_2(f, p)$ and $\mathsf{S}_3(\tau(f, p))$ have "exactly the same behavior" for good (f, p). In the following, whenever we talk about executions of $\mathsf{S}_2(f, p)$ and $\mathsf{S}_3(h)$, we assume that they are executed for the *same* distinguisher.

Definition 3.34. The function $\tau(f, p)$ is defined as follows: If (f, p) is good, run a simulation of $\mathsf{S}_2(f, p)$ in which the distinguisher completes all chains. Consider the tables G at the end of this execution, and for any i and x let $h(i, x) := G_i(x)$ in case $x \in G_i$, and $h(i, x) := \perp$ otherwise. If (f, p) is not good, let $\tau(f, p) := \lambda$.

For a table $h \neq \lambda$ we say "h has a good preimage" if there exists (f, p) such $\tau(f, p) = h$ (in which case (f, p) is good).

Lemma 3.35. *Suppose h has a good preimage. Consider any execution of* $\mathsf{S}_3(h)$ *and suppose the distinguisher completes all chains. Then,* $\mathsf{S}_3(h)$ *never queries h on an index* (i, x) *for which* $h(i, x) = \perp$*. Furthermore, the following two conditions on* (f, p) *are equivalent:*

(1) The pair (f, p) *is good and* $\tau(f, p) = h$*.*

(2) The queries and answers to the two-sided random function in $\mathsf{S}_2(f, p)$ *are exactly the same as the queries and answers to the Feistel construction in* $\mathsf{S}_3(h)$*; and* $h(i, x) = f(i, x)$ *for any query* (i, x) *issued to* f *or* h *by the simulator.*

Proof. We first show that (1) implies (2). It is sufficient to show the following:

- When the simulator sets $G_i(x) := f(i, x)$ in $\mathsf{S}_2(f, p)$, then $G_i(x) = f(i, x)$ in the end of the execution (and thus by definition of τ we have $h(i, x) = f(i, x)$).

- When the simulator or the distinguisher queries $\mathrm{P}(x_0, x_1)$ or $\mathrm{P}^{-1}(x_{14}, x_{15})$ it gets the same answer in $\mathsf{S}_2(f, p)$ and $\mathsf{S}_3(h)$.

To see the first point, note that if the simulator sets $G_i(x_i) := f(i, x_i)$ in $\mathsf{S}_2(f, p)$, this value will remain unchanged until the end of the execution since table entries in G are never overwritten (Lemma 3.31), and thus h will be set accordingly by definition of τ.

Now consider a query to $\mathrm{P}(x_0, x_1)$ by the simulator or the distinguisher (queries to P^{-1} are handled in the same way). Recall that we assume that the distinguisher completes all chains. Because of Lemma 3.32, the answer of the query to P is exactly what we obtain by evaluating the Feistel construction at the end in scenario S_2. But each query in the evaluation of the Feistel construction was either set as $G_i(x_i) := f(i, x_i)$ or in a ForceVal call, and in both cases the values of h must agree, since in good executions no value is ever overwritten (Lemma 3.31). Thus, the query to P is answered by the Feistel in $\mathsf{S}_3(h)$ in the same way.

We now show that (2) implies (1). So assume that (2) holds, and let (f_h, p_h) be a good preimage of h. As (f_h, p_h) satisfies (1), and (1) implies (2) as shown above, condition (2) holds for (f_h, p_h). As we assume that (2) holds for (f, p), we see that in the two executions $\mathsf{S}_2(f_h, p_h)$ and $\mathsf{S}_2(f, p)$ all queries to the two-sided random function are the same, and also the entries $f(i, x)$ and $f_h(i, x)$ for values considered match. This implies that (i) (f, p) is good, and (ii) $\tau(f, p) = \tau(f_h, p_h)$. To see this, note that the simulator's behavior only depends on the query answers it sees, and so all the steps of the simulator in $\mathsf{S}_2(f, p)$ and $\mathsf{S}_2(f_h, p_h)$ are identical. So (i) follows because if a bad event occurred in $\mathsf{S}_2(f, p)$, it would also occur in $\mathsf{S}_2(f_h, p_h)$, and (ii) follows because the tables G must be identical in the end of the executions of $\mathsf{S}_2(f, p)$ and $\mathsf{S}_2(f_h, p_h)$.

Finally, we argue that $\mathsf{S}_3(h)$ never queries h on an index (i, x) for which $h(i, x) = \perp$. Let (f_h, p_h) be a good preimage of h. Clearly (1) holds for h and (f_h, p_h), which implies (2) as shown above. Thus, it cannot be that a query to h in $\mathsf{S}_3(h)$ returns $\perp$, as otherwise the answers in $\mathsf{S}_2(f_h, p_h)$ and $\mathsf{S}_3(h)$ would differ. □

Lemma 3.36. *Suppose h has a good preimage. Pick (f, p) uniformly at random. Then,*

$$\Pr_{(f,p)}[(f, p) \text{ is good} \wedge \tau(f, p) = h] = 2^{-n|h|}, \tag{1}$$

where $|h|$ is the number of pairs (i, x) for which $h(i, x) \neq \perp$.

Proof. Let (f_h, p_h) be a good preimage of h. We first show that

$$\Pr_{(f,p)}[\text{all queries and answers in } \mathsf{S}_2(f, p) \text{ and } \mathsf{S}_2(f_h, p_h) \text{ are identical}] = 2^{-n|h|}. \tag{2}$$

To see this, note that every query to f is answered the same with probability 2^{-n}, and every query to p with probability 2^{-2n}. Because of Lemma 3.33 the number $|h|$ of non-nil entries in h is exactly the number of queries to f plus twice the number of queries to p.

We now conclude that

$$\begin{aligned}
2^{-n|h|} &= \Pr_{(f,p)}[\text{all queries and answers in } \mathsf{S}_2(f,p) \text{ and } \mathsf{S}_2(f_h,p_h) \text{ are identical}]\\
&= \Pr_{(f,p)}[\text{all queries and answers in } \mathsf{S}_2(f,p) \text{ and } \mathsf{S}_3(h) \text{ are identical}]\\
&= \Pr_{(f,p)}[(f,p) \text{ is good} \wedge \tau(f,p) = h].
\end{aligned}$$

The first equality is Equation 2 above. The second equality follows because (f_h, p_h) is good and $\tau(f_h, p_h) = h$, which by Lemma 3.35 (direction (1) $\Longrightarrow$ (2)) gives that all queries and answers in $\mathsf{S}_2(f_h, p_h)$ and $\mathsf{S}_3(h)$ are identical. The third equality then follows by Lemma 3.35, where we use the equivalence (1) $\Longleftrightarrow$ (2). □

Lemma 3.37. (Transition from S_2 to S_3) *The probability that a fixed distinguisher* **D** *answers 1 in* $\mathsf{S}_2(f,p)$ *for uniform random* (f,p) *differs at most by* $\frac{10^{21}\cdot q^{10}}{2^n}$ *from the probability that it answers 1 in* $\mathsf{S}_3(h)$ *for uniform random* h.

Proof. First, modify **D** such that it completes all chains, i.e., for each query to $\mathrm{P}(x_0, x_1)$ or to $(x_0, x_1) = \mathrm{P}^{-1}(x_{14}, x_{15})$ which it made during the execution (to either the two-sided random function in S_2 or the Feistel construction in S_3), it issues the corresponding Feistel queries to F in the end (i.e., it emulates a call to EVALUATEFORWARD$(x_0, x_1, 0, 14)$). We denote the modified distinguisher by $\mathbf{D}'$. This increases the number of queries of the distinguisher by at most a factor of 15. Furthermore, any unmodified distinguisher **D** that achieves some advantage will achieve the same advantage as the modified distinguisher $\mathbf{D}'$, and it is thus sufficient to bound the advantage of $\mathbf{D}'$.

We now consider the following distribution that outputs values h^*. To pick an element h^*, we pick a pair (f, p) uniformly at random. If $\tau(f, p) = \lambda$, we set $h^* := \lambda$. Otherwise, we let $h := \tau(f, p)$, and then for each entry in h where $h(i, x) = \bot$ we replace the entry by a string that is chosen independently and uniformly at random from $\{0, 1\}^n$. The result is h^*. Let H be the random variable that takes values according to this distribution.

We now claim that the probability that any fixed table $h^* \neq \lambda$ is output is at most $2^{-n|h^*|}$. To prove this, we first show that it cannot be that two different values $h \neq \lambda$ which both have a good preimage can yield the same h^*. Toward a contradiction assume that $h \neq \lambda$ and $h' \neq \lambda$ are different and both have a good preimage, and they yield the same h^*. Let (f_h, p_h) and $(f_{h'}, p_{h'})$ be good preimages of h and h', respectively. Then, Lemma 3.35 (direction (1) $\Longrightarrow$ (2)) implies that the queries and answers in $\mathsf{S}_2(f_h, p_h)$ and $\mathsf{S}_3(h)$ are the same. Furthermore, since $\mathsf{S}_3(h)$ never queries h on an index (i, x) where $h(i, x) = \bot$ (Lemma 3.35), we get that the queries and answers in $\mathsf{S}_3(h)$ and $\mathsf{S}_3(h^*)$ are the same. Arguing symmetrically for $(f_{h'}, p_{h'})$, we see that the queries and answers in $\mathsf{S}_3(h')$ and $\mathsf{S}_3(h^*)$ are the same, and so the queries and answers in $\mathsf{S}_2(f_h, p_h)$ and $\mathsf{S}_2(f_{h'}, p_{h'})$ must be the same. Since the simulator's behavior only depends on the query answers it sees, we get that all the steps of the simulator in $\mathsf{S}_2(f, p)$ and $\mathsf{S}_2(f_h, p_h)$ are identical. In particular, the tables G must be identical in the end of the execution, and thus by definition of τ, this implies that $h = h'$, a contradiction.

We now calculate the probability of getting a fixed table $h^* \neq \lambda$. In the first case, suppose there exists h with a good preimage that can lead to h^*. Let ρ be the randomness that is used to replace the $\perp$ entries in h by random entries. We have

$$\begin{aligned}&\Pr_{(f,p),\rho}[H = h^*]\\&\quad = \Pr_{(f,p),\rho}[(f, p) \text{ is good } \wedge h = \tau(f, p) \text{ can lead to } h^* \wedge \text{ filling with } \rho \text{ leads to } h^*].\end{aligned}$$

Now, as we have seen above, no two different values for h can yield the same h^*. Thus, we can assume that $h^* = (h, \rho^*)$, where h is the unique table that leads to h^*, and ρ^* stands for the entries that occur in h^*, but are $\perp$ in h. Then, the above probability equals

$$\begin{aligned}&\Pr_{(f,p),\rho}[(f, p) \text{ is good } \wedge \tau(f, p) = h \wedge \rho = \rho^*]\\&\quad = \Pr_{(f,p)}[(f, p) \text{ is good } \wedge \tau(f, p) = h] \cdot \Pr_{\rho}[\rho = \rho^*]\\&\quad = 2^{-n|h|} \cdot 2^{-n(|h^*|-|h|)} = 2^{-n|h^*|}.\end{aligned}$$

The first equality above holds because ρ is chosen independently and uniformly at random, and the second equality follows by Lemma 3.36 and since ρ is chosen uniformly at random.

In the second case, there exists no h with a good preimage that can lead to h^*. Then we have $\Pr_{(f,p),\rho}[H = h^*] = 0$, and so in both cases

$$\Pr_{(f,p),\rho}[H = h^*] \leq 2^{-n|h^*|}. \tag{3}$$

This implies that the statistical distance of the distribution over h^* which we described to the uniform distribution is exactly the probability that (f, p) is not good. For completeness, we give a formal argument for this. Consider H as above, and let U be a random variable taking uniform random values from $\{0, 1\}^{|h^*|}$. We have

$$\begin{aligned}d(U, H) &= \frac{1}{2}\sum_{h*}\left|\Pr[U = h^*] - \Pr_{(f,p),\rho}[H = h^*]\right|\\&= \frac{1}{2}\Big|\underbrace{\Pr[U = \lambda]}_{=0} - \underbrace{\Pr_{(f,p),\rho}[H = \lambda]}_{= \Pr_{(f,p)}[(f,p) \text{ is not good}]}\Big| + \frac{1}{2}\sum_{h^*\neq\lambda}\left|\Pr[U = h^*] - \Pr_{(f,p),\rho}[H = h^*]\right|\\&= \frac{1}{2}\Pr_{(f,p)}[(f, p) \text{ is not good}] + \frac{1}{2}\underbrace{\sum_{h^*\neq\lambda}\Pr[U = h^*]}_{=1} - \frac{1}{2}\underbrace{\sum_{h^*\neq\lambda}\Pr_{(f,p),\rho}[H = h^*]}_{=1-\Pr_{(f,p)}[(f,p) \text{ is not good}]}\\&= \Pr_{(f,p)}[(f, p) \text{ is not good}],\end{aligned}$$

where the third equality uses (3).

We proceed to argue that $\Pr_{(f,p)}[(f,p) \text{ is not good}]$ is small. In $\mathsf{S}_2(f,p)$, by Lemma 3.4 we have that $|G_i| \leq 6 \cdot (15 \cdot q)^2$ and $|P| \leq 6 \cdot (15 \cdot q)^2$, where the additional factor of 15 comes in because the distinguisher completes all chains. By Lemma 3.22,

$$\Pr_{(f,p)}[(f,p) \text{ is not good}] \leq 18\,000 \cdot \frac{(6 \cdot (15 \cdot q)^2)^5}{2^n} < \frac{10^{20} \cdot q^{10}}{2^n}.$$

By Lemma 3.35 (direction (1) $\implies$ (2)), for good (f,p), the behavior of $\mathsf{S}_2(f,p)$ and $\mathsf{S}_3(H)$ is identical. Thus,

$$\Big| \Pr_{(f,p)}[\mathbf{D}' \text{ outputs 1 in } \mathsf{S}_2(f,p)] - \Pr_{(f,p),\rho}[\mathbf{D}' \text{ outputs 1 in } \mathsf{S}_3(H)] \Big| \\ \leq \Pr_{(f,p)}[(f,p) \text{ is not good}].$$

Furthermore,

$$\Big| \Pr_{(f,p),\rho}[\mathbf{D}' \text{ outputs 1 in } \mathsf{S}_3(H)] - \Pr[\mathbf{D}' \text{ outputs 1 in } \mathsf{S}_3(U)] \Big| \leq d(H,U) \\ = \Pr_{(f,p)}[(f,p) \text{ is not good}],$$

and therefore

$$\Big| \Pr_{(f,p)}[\mathbf{D}' \text{ outputs 1 in } \mathsf{S}_2(f,p)] - \Pr[\mathbf{D}' \text{ outputs 1 in } \mathsf{S}_3(U)] \Big| \\ \leq 2 \cdot \Pr_{(f,p)}[(f,p) \text{ is not good}] \\ < \frac{10^{21} \cdot q^{10}}{2^n},$$

using our bound on the probability that (f,p) is good above. □

3.6. *Equivalence of the Third and the Fourth Scenarios*

In S_3, the distinguisher accesses the random functions through the simulator. We want to show that the distinguisher can instead access the random functions directly.

Lemma 3.38. *Suppose that in* $\mathsf{S}_3(h)$ *the simulator* $\mathbf{T}(h)$ *eventually answers a query* $\mathrm{F}(i,x)$. *Then, it is answered with* $h(i,x)$.

Proof. The simulator $\mathbf{T}(h)$ either sets $G_i(x) := h(i,x)$ or $G_i(x_i) := x_{i-1} \oplus x_{i+1}$ in a call to Adapt. For pairs (i,x) which are set by the first call the lemma is clear. Otherwise, consider the Adapt call: just before the call, the Feistel construction was evaluated either forward or backward in a call to $\Psi(h).\mathrm{P}(x_0,x_1)$ or $\Psi(h).\mathrm{P}^{-1}(x_{14},x_{15})$. Since $\Psi(h)$ evaluates P and P^{-1} with calls to h, the value $G_i(x_i)$ must be $h(i,x)$ as well. □

Lemma 3.39. (Transition from S_3 to S_4) *The probability that a fixed distinguisher answers 1 in* $\mathsf{S}_3(h)$ *for uniformly chosen h differs at most by* $\frac{10^{21} \cdot q^{10}}{2^n}$ *from the probability that it answers 1 in* S_4.

Proof. By Lemma 3.37, the probability that the distinguisher outputs 1 does not differ by more than $\frac{10^{21} \cdot q^{10}}{2^n}$ in S_2 and S_3. As the simulator is efficient in S_2 by Lemma 3.5, this implies that with probability $1 - \frac{10^{21} \cdot q^{10}}{2^n}$ the simulator must give an answer in S_3. Thus, using Lemma 3.38 we get that that the probability that the distinguisher answers 1 differs in S_3 and S_4 by at most $\frac{10^{21} \cdot q^{10}}{2^n}$. □

Acknowledgements

It is a pleasure to thank Ueli Maurer for his insightful feedback. We also would like to thank the reviewers of the Journal of Cryptology for their very detailed comments, which helped us in substantially improving the presentation of the paper. Robin Künzler was partially supported by the Swiss National Science Foundation (SNF), Project No. 200021-132508.

Appendix 1: A Note on Honest-but-Curious Indifferentiability

In this section, we show that the Feistel construction with up to a logarithmic number of rounds is *not* indifferentiable from a random permutation in the honest-but-curious model [24]. Combined with our main result in the general model, this shows that honest-but-curious indifferentiability is not implied in general by full indifferentiability. This does not contradict any result of [24], where it was shown that the Feistel construction with a *super-logarithmic* number of rounds is indifferentiable from a random permutation in the honest-but-curious model (we note, though, that [24] proved that for up to a logarithmic number of rounds of the Feistel construction, indifferentiability in the honest-but-curious model would have implied indifferentiability in the general model; but since we show in the following that the premise is false, the implication becomes void).

Informally, in the honest-but-curious indifferentiability model, the distinguisher cannot query the round functions of the Feistel construction directly. It can only make two types of queries: direct queries to the construction, and queries to the construction where in addition the intermediate round function values are provided. When interacting with a random permutation $\mathbf{P}$ and a simulator $\mathbf{S}$, the first type of queries are sent directly to $\mathbf{P}$, while the second type are sent to $\mathbf{S}$ which makes the corresponding query to $\mathbf{P}$, and in addition provides a simulated transcript of the intermediate round function values. Note that the simulator $\mathbf{S}$ is not allowed to make additional queries to $\mathbf{P}$ apart from forwarding the queries from the distinguisher; see [24] for a precise definition.

The authors of [24] introduced the notion of *transparent construction*. A construction $\mathbf{C}^{\mathbf{F}}$ is said to be transparent if for any x, the value of $\mathbf{F}(x)$ can be computed efficiently by making a polynomial number of queries to $\mathbf{C}^{\mathbf{F}}$, where in addition to $\mathbf{C}^{\mathbf{F}}(y)$ one gets the inputs and outputs of $\mathbf{F}$ used by $\mathbf{C}$ to compute the answer to each query y. It is shown in [24] that the Feistel construction with up to a logarithmic number of rounds is a transparent construction. Namely, the authors construct an extracting algorithm E achieving the following: given oracle access to $\Psi^{\mathbf{F}}$ and the intermediate round function values $\mathbf{F}_i(x)$ used to compute the answer to any query to the construction, E can compute the value of $\mathbf{F}_i(x)$ for any round i and any $x \in \{0,1\}^n$. An important property of E is that it only makes *forward* queries to the Feistel construction.

Algorithm E implies that for a Feistel construction with up to a logarithmic number of rounds r, it is possible to find an input message (x_0, x_1) such that the left half of the output (x_r, x_{r+1}) has an arbitrary value x_r (say, 0^n), by only making forward queries to $\Psi^{\mathbf{F}}$: this corresponds to how algorithm E can obtain $\mathbf{F}_r(x_r)$, where r is the last round. But this task is clearly impossible with a random permutation $\mathbf{P}$: namely, it is infeasible to find (x_0, x_1) such that the left half of $\mathbf{P}(x_0, x_1)$ has a fixed arbitrary value by making only forward queries to $\mathbf{P}$. This implies that a simulator in the honest-but-curious model will necessarily fail (recall that such a simulator only forwards queries from the distinguisher to $\mathbf{P}$ and cannot make additional queries). Therefore, the Feistel construction with up to a logarithmic number of rounds is *not* indifferentiable from a random permutation in the honest-but-curious model. Since our main result is that the Feistel construction with fourteen rounds is fully indifferentiable from a random permutation, this shows that honest-but-curious indifferentiability does not imply in general full indifferentiability.

Appendix 2: Building an Ideal Cipher from a Random Oracle

Theorem 3.2. *The 14-round keyed Feistel construction using a random oracle is indifferentiable from an ideal cipher. For an ideal cipher with κ-bit key and $2n$-bit inputs, and any distinguisher that issues at most q queries, except with probability $\frac{10^8 \cdot q^{17}}{2^{2n}}$, the simulator makes at most $1400q^8$ queries and runs in time $O(q^8)$. The distinguishing advantage is at most $\frac{10^8 \cdot q^{17}}{2^{2n}} + \frac{10^{22} \cdot q^{11}}{2^n}$.*

Proof (Sketch). The simulator $\overline{\mathbf{S}}$ is built in a nearly black box way from simulator $\mathbf{S}$ from the proof of Theorem 3.1. In particular, for all κ-bit keys k, we run an independent copy of the simulator $\mathbf{S}$, called $\mathbf{S}_k$, each one maintaining its own state—the only difference between $\mathbf{S}_k$ and $\mathbf{S}$ is that queries to $\mathbf{P}$ and $\mathbf{P}^{-1}$ are replaced by queries to $\mathbf{E}_k$ and $\mathbf{E}_k^{-1}$.[10] Upon a query $x \in \{0,1\}^*$, $\overline{\mathbf{S}}$ first checks whether it can be parsed as $x = \langle i \rangle \| k \| x'$ for $i \in \{1, \ldots, 14\}$, $k \in \{0,1\}^\kappa$, and $x' \in \{0,1\}^n$. If so, it calls $\mathbf{S}_k.\mathrm{F}(i, x')$, and returns the resulting answer. Otherwise, $\overline{\mathbf{S}}$ perfectly simulates the random oracle by keeping an appropriate random table.

We say that a *key k is associated with a query Q* (and symmetrically, a query Q is associated with the key k) if it is a query of the form $Q = (k, \cdot)$ to the Feistel construction or to the ideal cipher, or it is a query of the form $Q = \langle i \rangle \| k \| x'$ to the random oracle

[10] Of course, to avoid running an exponential number of simulator instances, we use lazy evaluation, running only $\mathbf{S}_k$ for keys k that are actually queried.

or its simulation. For all integers $i \geq 0$, we define a system $\mathbf{H}_i$ which keeps track of keys associated with queries. For the first i distinct keys associated with some query, queries associated with these keys are always answered by either $\mathbf{E}$ or $\overline{\mathbf{S}}^{\mathbf{E}}$ depending on the query type, whereas queries associated with later appearing keys are answered by either $\overline{\Psi}^{\mathbf{R}}_{14}$ or $\mathbf{R}$, depending on the query type. Now, for a distinguisher $\mathbf{D}$ making overall at most q queries, we can easily see by inspection that

$$\Pr[\mathbf{D}(\mathbf{H}_0) = 1] = \Pr[\mathbf{D}(\overline{\Psi}^{\mathbf{R}}_{14}, \mathbf{R}) = 1] \text{ and } \Pr[\mathbf{D}(\mathbf{H}_q) = 1] = \Pr[\mathbf{D}(\mathbf{E}, \overline{\mathbf{S}}^{\mathbf{E}}) = 1] ,$$

and therefore, using the triangle inequality

$$\Delta^{\mathbf{D}}((\overline{\Psi}^{\mathbf{R}}_{14}, \mathbf{R}), (\mathbf{E}, \overline{\mathbf{S}}^{\mathbf{E}})) \leq \sum_{i=0}^{q-1} \Delta^{\mathbf{D}}(\mathbf{H}_i, \mathbf{H}_{i+1}) .$$

For all $i \in \{0, 1, \ldots, q-1\}$, a $2n$-bit random permutation $\mathbf{P}$, and $\mathbf{F} = (\mathbf{F}_1, \ldots, \mathbf{F}_{14})$ being independent n-bit to n-bit random functions, it is now easy to construct a distinguisher $\mathbf{D}_i$ making at most q queries to either of $(\Psi^{\mathbf{F}}_{14}, \mathbf{F})$ and $(\mathbf{P}, \mathbf{S}^{\mathbf{P}})$ such that $\Delta^{\mathbf{D}_i}((\Psi^{\mathbf{F}}_{14}, \mathbf{F}), (\mathbf{P}, \mathbf{S}^{\mathbf{P}})) = \Delta^{\mathbf{D}}(\mathbf{H}_i, \mathbf{H}_{i+1})$: The distinguisher uses the given system to simulate queries associated with the i-th key, and simulates the answers to all queries associated with other keys internally. The indistinguishability bound follows using the one from Theorem 3.1 for $\Delta^{\mathbf{D}}(\mathbf{H}_i, \mathbf{H}_{i+1})$.

By Theorem 3.1, every $\mathbf{S}_k$ can issue too many queries and run for too long with probability at most $\frac{10^8 \cdot q^{16}}{2^{2n}}$, and the overall probability that this happens for some k is obtained via the union bound. Now, assuming this event does not happen, let q_k be the number of queries associated with key k the distinguisher has made in an execution—hence $\sum_k q_k \leq q$. Then, the simulator $\overline{\mathbf{S}}$ has made at most $\sum_k 1400 q_k^8 \leq 1400 q^8$ queries, and similarly, has run for time at most $O(q^8)$. □

The resulting concrete parameters incur an additional factor q loss with respect to the original bound for the random permutation case. This is however just to ease exposition—the same bounds as in Theorem 3.1 can be obtained by giving a more direct proof adding the handling of keys to the proof of the random permutation case.

References

[1] E. Andreeva, A. Bogdanov, Y. Dodis, B. Mennink, J.P. Steinberger, On the indifferentiability of key-alternating ciphers, in R. Canetti, J.A. Garay, editors, *Advances in Cryptology—CRYPTO 2013 (Proceedings, Part I)*, *Lecture Notes in Computer Science*, vol. 8042 (Springer, Berlin, 2013), pp. 531–550. Full version available at http://eprint.iacr.org/2013/061

[2] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, On the indifferentiability of the sponge construction, in N.P. Smart, editor, *Advances in Cryptology—EUROCRYPT 2008*, *Lecture Notes in Computer Science*, vol. 4965 (Springer, Berlin, 2008), pp. 181–197

[3] D. Boneh, M.K. Franklin, Identity-based encryption from the weil pairing. *SIAM J. Comput.* **32**(3), 586–615 (2003)

[4] M. Bellare, T. Kohno, A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications, in *Advances in Cryptology—EUROCRYPT 2003*, *Lecture Notes in Computer Science*, vol. 2656, pp. 491–506 (2003)

[5] A. Bogdanov, L.R. Knudsen, G. Leander, F.-X. Standaert, J.P. Steinberger, E. Tischhauser, Key-alternating ciphers in a provable setting: encryption using a small number of public permutations—(Extended Abstract), in D. Pointcheval, T. Johansson, editors, *Advances in Cryptology—EUROCRYPT 2012*, *Lecture Notes in Computer Science*, vol. 7237 (Springer, Berlin, 2012), pp. 45–62

[6] J. Black, The ideal-cipher model, revisited: an uninstantiable blockcipher-based hash function, in *FSE 2006*, *Lecture Notes in Computer Science*, vol. 4047, pp. 328–340 (2006)

[7] D. Boneh, B. Lynn, H. Shacham, Short signatures from the weil pairing. *J. Cryptol.***17**(4), 297–319 (2004)

[8] M. Bellare, D. Pointcheval, P. Rogaway, Authenticated key exchange secure against dictionary attacks, in *EUROCRYPT00*, *Lecture Notes in Computer Science*, vol. 1807, pp. 139–155 (2000)

[9] M. Bellare, P. Rogaway, Random oracles are practical: a paradigm for designing efficient protocols, in *CCS '93: Proceedings of the 1st ACM Conference on Computer and Communications Security* (ACM, New York, NY, USA, 1993), pp. 62–73

[10] M. Bellare, P. Rogaway. Optimal asymmetric encryption, in *Advances in Cryptology—EUROCRYPT '94*, Lecture Notes in Computer Science, pp. 92–111 (1994)

[11] M. Bellare, P. Rogaway, The exact security of digital signatures—how to sign with RSA and Rabin, in *Advances in Cryptology—EUROCRYPT '96*, *Lecture Notes in Computer Science*, pp. 399–416 (1996)

[12] J. Black, P. Rogaway, Ciphers with arbitrary finite domains, in *CT-RSA 2002*, *Lecture Notes in Computer Science*, pp. 114–130 (2002)

[13] M. Bellare, P. Rogaway, The security of triple encryption and a framework for code-based game-playing proofs, in *Advances in Cryptology—EUROCRYPT 2006*, *Lecture Notes in Computer Science*, vol. 4004, pp. 409–426 (2006)

[14] J. Black, P. Rogaway, T. Shrimpton, Black-box analysis of the block-cipher-based hash-function constructions from PGV, in *Advances in Cryptology—CRYPTO 2002*, *Lecture Notes in Computer Science*, vol. 2442, pp. 320–335 (2002)

[15] R. Canetti, Universally composable security: a new paradigm for cryptographic protocols, in *FOCS '01: Proceedings of the 42nd IEEE Annual Symposium on Foundations of Computer Science*, pp. 136–145 (2001)

[16] J.-S. Coron, Y. Dodis, C. Malinaud, P. Puniya, Merkle-Damgård revisited: how to construct a hash function, in V. Shoup, editor, *Advances in Cryptology—CRYPTO 2005*, *Lecture Notes in Computer Science*, vol. 3621 (Springer, Berlin, 2005), pp. 430–448

[17] R. Canetti, O. Goldreich, S. Halevi, The random oracle methodology, revisited. *J. ACM***51**(4), 557–594 (2004)

[18] S. Chen, R. Lampe, J. Lee, Y. Seurin, J.P. Steinberger, Minimizing the two-round even-mansour cipher, in J.A. Garay, R. Gennaro, editors, *Advances in Cryptology—CRYPTO 2014 (Proceedings, Part I)*, *Lecture Notes in Computer Science*, vol. 8616 (Springer, Berlin, 2014), pp. 39–56. Full version available at http://eprint.iacr.org/2014/443

[19] J.-S. Coron, J. Patarin, Y. Seurin, The random oracle model and the ideal cipher model are equivalent, in D. Wagner, editor, *CRYPTO*, *Lecture Notes in Computer Science*, vol. 5157 (Springer, Berlin, 2008), pp. 1–20

[20] J.-S. Coron, J. Patarin, Y. Seurin, The random oracle model and the ideal cipher model are equivalent. Cryptology ePrint Archive, Report 2008/246, August 2008. Version: 20080816:121712, http://eprint.iacr.org/, Extended Abstract at CRYPTO 2008

[21] S. Chen, J. Steinberger, Tight security bounds for key-alternating ciphers, in P.Q. Nguyen, E. Oswald, editors, *Advances in Cryptology—EUROCRYPT 2014*, *Lecture Notes in Computer Science*, vol. 8441, pp. 327–350 (Springer, Berlin, 2014). Full version available at http://eprint.iacr.org/2013/222

[22] I.B. Damgård, A design principle for hash functions, in *Advances in Cryptology—CRYPTO '89*, *Lecture Notes in Computer Science*, vol. 435, pp. 416–427 (1989)

[23] G. Demay, P. Gazi, M. Hirt, U. Maurer, Resource-restricted indifferentiability, in *EUROCRYPT13*, *Lecture Notes in Computer Science*, vol. 7881, pp. 664–683 (2013)

[24] Y. Dodis, P. Puniya, On the relation between the ideal cipher and the random oracle models, in *Theory of Cryptography—TCC 2006*, *Lecture Notes in Computer Science*, vol. 3876, pp. 184–206 (2006)

[25] S. Dziembowski, K. Pietrzak, D. Wichs, Non-malleable codes, in *Innovations in Computer Science—ICS 2010*, pp. 434–452 (2010)

[26] Y. Dodis, L. Reyzin, R.L. Rivest, E. Shen, Indifferentiability of permutation-based compression functions and tree-based modes of operation, with applications to MD6, in O. Dunkelman, editor, *Fast Software Encryption—FSE 2009*, *Lecture Notes in Computer Science*, vol. 5665 (Springer, Berlin, 2009), pp. 104–121

[27] S. Even, Y. Mansour, A construction of a cipher from a single pseudorandom permutation. *J. Cryptol.* **10**(3), 151–162 (1997)

[28] A. Fiat, A. Shamir, How to prove yourself: practical solutions to identification and signature problems, in *Advances in Cryptology—CRYPTO '86*, *Lecture Notes in Computer Science*, vol. 263, pp. 186–194 (1986)

[29] T. Holenstein, R. Künzler, S. Tessaro, The equivalence of the random oracle model and the ideal cipher model, revisited, in L. Fortnow, S.P. Vadhan, editors, *STOC* (ACM, New York, 2011), pp. 89–98

[30] J. Kilian, P. Rogaway, How to protect DES against exhaustive key search (an analysis of DESX). *J. Cryptol.* **14**(1), 17–35 (2001)

[31] J. Kahn, M.E. Saks, C.D. Smyth, A dual version of Reimer's inequality and a proof of Rudich's conjecture, in *IEEE Conference on Computational Complexity*, pp. 98–103 (2000)

[32] M. Luby, C. Rackoff, How to construct pseudorandom permutations from pseudorandom functions. *SIAM J. Comput.* **17**(2), 373–386 (1988)

[33] R. Lampe, Y. Seurin, How to construct an ideal cipher from a small set of public permutations, in K. Sako, P. Sarkar, editors, *Advances in Cryptology—ASIACRYPT 2013 (Proceedings, Part I)*, *Lecture Notes in Computer Science*, vol. 8269 (Springer, Berlin, 2013), pp. 444–463. Full version available at http://eprint.iacr.org/2013/255

[34] Y. Lindell, H. Zarosim, Adaptive zero-knowledge proofs and adaptively secure oblivious transfer, in *Theory of Cryptography Conference—TCC 2009*, *Lecture Notes in Computer Science*, vol. 5444, pp. 183–201 (2009)

[35] U. Maurer, Indistinguishability of random systems, in *Advances in Cryptology—EUROCRYPT 2002*, *Lecture Notes in Computer Science*, vol. 2332, pp. 110–132 (2002)

[36] R.C. Merkle, A certified digital signature, in *Advances in Cryptology—CRYPTO '89*, *Lecture Notes in Computer Science*, vol. 435, pp. 218–238 (1989)

[37] A. Mandal, J. Patarin, Y. Seurin, On the public indifferentiability and correlation intractability of the 6-round Feistel construction, in *TCC* (2012). Full version available at http://eprint.iacr.org/2011/496.pdf

[38] U. Maurer, R. Renner. Abstract cryptography, in *Innovations in Computer Science—ICS 2011*, pp. 1–21 (2011)

[39] U. Maurer, R. Renner, C. Holenstein, Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology, in *Theory of Cryptography Conference—TCC 2004*, *Lecture Notes in Computer Science*, vol. 2951, pp. 21–39, February 2004

[40] P. Rogaway, J.P. Steinberger, Constructing cryptographic hash functions from fixed-key blockciphers, in D. Wagner, editor, *Advances in Cryptology—CRYPTO 2008*, *Lecture Notes in Computer Science*, vol. 5157 (Springer, Berlin, 2008), pp. 433–450

[41] P. Rogaway, J.P. Steinberger, Security/efficiency tradeoffs for permutation-based hashing, in N.P. Smart, editor, *Advances in Cryptology—EUROCRYPT 2008*, *Lecture Notes in Computer Science*, vol. 4965 (Springer, Berlin, 2008), pp. 220–236

[42] T. Ristenpart, H. Shacham, T. Shrimpton, Careful with composition: limitations of the indifferentiability framework, in K.G. Paterson, editor, *EUROCRYPT*, *Lecture Notes in Computer Science*, vol. 6632 (Springer, Berlin, 2011), pp. 487–506

[43] S. Rudich, *Limits on the Provable Consequences of One-way Functions*. PhD thesis (1989)

[44] Y. Seurin, *Primitives et protocoles cryptographiques à sécurité prouvée.* PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, UFR de Sciences - École doctorale SoFt - Laboratoire PRiSM (2009)

[45] Y. Seurin, A note on the indifferentiability of the 10-round feistel construction, March 2011. Unpublished note available from the author

[46] C.E. Shannon, Communication theory of secrecy systems. *Bell Syst. Tech. J.***28**, 656–715 (1949)
[47] V. Shoup, Sequences of games: a tool for taming complexity in security proofs (2004)