On the Indifferentiability of Key-Alternating Feistel Ciphers with No Key Derivation*

Chun Guo[1,2], and Dongdai Lin[1,**]

[1] State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences, China
[2] University of Chinese Academy of Sciences, China
{guochun,ddlin}@iie.ac.cn

Abstract. Feistel constructions have been shown to be indifferentiable from random permutations at STOC 2011. Whereas how to properly mix the keys into an un-keyed Feistel construction without appealing to domain separation technique to obtain a block cipher which is provably secure against known-key and chosen-key attacks (or to obtain an ideal cipher) remains an open problem. We study this, particularly the basic structure of NSA's SIMON family of block ciphers. SIMON family takes a construction which has the subkey xored into a halve of the state at each round. More clearly, at the i-th round, the state is updated according to

$$(x_i, x_{i-1}) \mapsto (x_{i-1} \oplus F_i(x_i) \oplus k_i, x_i)$$

For such key-alternating Feistel ciphers, we show that 21 rounds are sufficient to achieve indifferentiability from ideal ciphers with $2n$-bit blocks and n-bit keys, assuming the n-to-n-bit round functions $F_1, \ldots, F_{21}$ to be random and public and an identical user-provided n-bit key to be applied at each round. This gives an answer to the question mentioned before, which is the first to our knowledge.

Keywords: Block cipher, ideal cipher, indifferentiability, key-alternating cipher, Feistel cipher.

1 Introduction

Block Ciphers, and the Security Notions. Block ciphers are among the most important primitives in cryptography. For a block cipher, the standard security notion is the indistinguishability from a random permutation when the key is fixed to some unknown random values. Such pseudorandomness captures the security in traditional single secret key setting. However, block ciphers find numerous and essential uses beyond encryption. For instance, block ciphers have been used to build hash functions and message authentication codes. These applications require the security in the *open key model*, where the adversary knows

* A full version of this paper is available [16].
** Corresponding Author.

Y. Dodis and J.B. Nielsen (Eds.): TCC 2015, Part I, LNCS 9014, pp. 110–133, 2015.
© International Association for Cryptologic Research 2015

or even chooses the keys. To assess such stronger-than-pseudorandomness security, the *indifferentiability framework* has to be employed. As a generalization of the indistinguishability notion, the indifferentiability framework provides a formal way to assess the security of idealized constructions of block ciphers and hash functions. It can be used to evaluate the "closeness" of a block cipher construction to an *ideal cipher*[1]. Despite the uninstantiability of idealized models [12,23,9], such indifferentiability proofs are widely believed to be able to show the nonexistence of generic attacks which do not exploit the inner details of the implementations of the underlying building blocks.

Feistel Constructions. Existing block cipher designs can be roughly split into two families, namely Feistel-based ciphers and substitution-permutation networks (SPNs). Starting from the seminal Luby-Rackoff paper [20], Feistel constructions have been extensively studied. Most of the provable security works fall in the Luby-Rackoff framework [20], in which the round functions are idealized as being uniformly random (and secret). Such works covered indistinguishability/provable security in the single secret key model (e.g. [24,22,25]), provable security in the open key model (Mandal *et al.* [21] and Andreeva *et al.* [3]), and provable security under related-key attacks (Manuel *et al.* [5]). A recent series of works studied the indifferentiability from random permutations of Luby-Rackoff construction, including the works of Coron *et al.* [14], Seurin [27], and Holenstein *et al.* [17], and the number of rounds required was finally fixed to 14 by Holenstein *et al.* [17].

Our Problem: How to Mix the Key into Feistel. In this paper, we consider the problem that *how to mix the key material into a Feistel construction by a popular approach to obtain a block cipher indifferentiable from an* ***ideal cipher***. Since an un-keyed Feistel construction is indifferentiable from a random permutation, a Feistel-based cipher indifferentiable from an ideal cipher can be trivially obtained through domain separation. However, such a result tells us nothing about how to concretely mix the keys into the state – in fact, none of the works mentioned before addressed this problem. To our knowledge, domain separation technique is seldom used in existing block cipher designs. Existing designs usually inserts keys via efficient group operations, e.g. xor and modular addition; therefore, this problem has practical meanings.

A natural candidate solution to this problem is the construction called *key-alternating Feistel cipher* (KAF for short) analyzed by Lampe *et al.* [19]. The KAF cipher they considered has the round keys xored before each round function, as depicted in Fig. 1 (left). Lampe *et al.* studied the indistinguishability of KAF in a setting where the underlying round functions are random and *public* (in contrast to the classical Luby-Rackoff setting) and the keys are fixed and secret; this is also the only provable security work on KAF.

However, due to the well known complementation property, there exist obstacles when trying to achieve an indifferentiability proof for KAF (detailed

[1] See Sect. 2 for the formal definitions of indifferentiability and the ideal cipher model.

discussions are deferred to the full version [16]). This motivates us to turn to another candidate construction, which has the round key xored into the halve of the state after the round functions. Due to the similarity between the two constructions, we denote the latter construction by KAF^* to follow the convention of Lampe *et al.* while making a distinction. For KAF^*, the $2n$-bit intermediate state s_i is split to two halves, i.e. $s_i = (x_{i+1}, x_i)$ where $i \in \{0, 1, \ldots, r\}$, and at the i-th round, the state value is updated according to

$$(x_i, x_{i-1}) \mapsto (x_{i-1} \oplus F_i(x_i) \oplus k_i, x_i),$$

as depicted in Fig. 1 (right). KAF^* can be seen as the basic structure of NSA's SIMON family of block ciphers.

Clearly, the proof for KAF^* with no cryptographically strong assumption about the key derivation functions is more attractive, since such key derivations are more relevant to practice than random oracle modeled ones. Whereas KAF^* with independent round keys cannot resist related-key attacks (the case is similar to Even-Mansour ciphers). Hence we consider KAF^* with an identical user-provided n-bit key applied at each round, and call such ciphers *single-key* KAF^* ($SKAF^*$ for short). The 21-round $SKAF^*$ is depicted in Fig. 2. With the discussions above, we focus on the question that *whether it is possible for* $SKAF^*$ *with sufficiently many rounds to be indifferentiable from ideal ciphers.*

Our Results. We show 21-round $SKAF^*$ to be indifferentiable from ideal ciphers, thus for the first time giving a solution to the problem *how to mix keys into Feistel in the open-key model.*

Theorem. *The 21-round key-alternating Feistel cipher* $SKAF^*_{21}$ *with all round functions* $\mathbf{F} = (F_1, \ldots, F_{21})$ *being 21 independent* n*-to-*n*-bit random functions and an identical (user-provided)* n*-bit key* k *applied at each round is indifferentiable from an ideal cipher with* $2n$*-bit blocks and* n*-bit keys.*

To our knowledge, this paper is also the first to study the indifferentiability/provable security of key-alternating Feistel ciphers – in particular, with no key derivation – in the open key model.

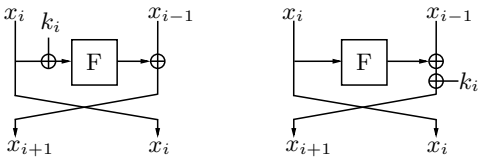

Fig. 1. Mixing the key into: (left) the input of the round function – KAF; (right) the halve of the state after the round function – KAF^*

From a practical point of view, our results suggest a possible choice to resist complementing attack and its extensions (see [7]) when designing Feistel

ciphers[2]. KAF with random oracle modeled key derivation functions may also have such resistance. However, practical key derivation algorithms are usually designed to be "lightweight" and moderately complex, and KAF with such moderately complex key derivations may still be broken in hash mode (the example is Camellia, in [7]). Hence we think our results have its own interest. Meanwhile, since publicly released in June 2013, the SIMON family of block ciphers [6] designed by NSA has attracted considerable attention due to its simple structure, high flexibility, and remarkable performance [4,8,1,28,11]. SIMON family is based on KAF^*. Our results may be seen as a first step towards understanding the underlying reasons.

Remark. We heavily borrow the techniques used by Holenstein *et al.* [17] and Lampe *et al.* [18] (see the next paragraph). We stress that our main constructions consist of the indifferentiability result for $SKAF^*$ and the analyzes of $SKAF^*$.

Overview of Techniques. We reuse and adapt the *simulation via chain-completion* technique introduced by Coron *et al.* [14], while the overall framework is very close to that used by Holenstein *et al.* [17] and Lampe *et al.* [18]. This framework consists of constructing a simulator which works by detecting and completing *partial chains* created by the queries of the distinguisher. To ensure consistency in the answers while avoiding exponentially many chain completions, each of the rounds in the construction is assigned a unique and specific role needed in the proof, including *chain detection*, *uniformness ensuring*, and *chain adaptation* (see Fig. 2). By this, the simulator first detects new partial chains when the associated values have "filled" the chain detection zone; then fills in the corresponding computation chain by both querying the ideal primitive and simulating the other necessary function values, until only the values of the round functions in the chain adaptation zone remain (possibly) undefined; and finally defines these values to complete the whole chain so that the answers of the ideal primitive are consistent with the function values simulated by the simulator.

Adaptations in this Work. To fit into the $SKAF^*$ context, the framework has to be adapted. Note that in the $SKAF^*$ context, each complete chain corresponds to a unique pair of input and output where the input consists of an n-bit key and a $2n$-bit block; hence the entropy of each chain is $3n$ bits, and it is necessary and sufficient to uniquely specify a chain by the queries to 3 round functions (recall that for un-keyed Feistel, the entropy of each chain is only $2n$ bits). Another consequence of this property is that in the $SKAF^*$ context, two different chains may collide at two successive rounds, i.e. for two different chains $(x_0, x_1, \ldots)$ and $(x'_0, x'_1, \ldots)$, it may hold that $x_j = x'_j \wedge x_{j+1} = x'_{j+1}$ for some j. As a comparison, consider the un-keyed Feistel context: in this context, for

[2] This idea is not new, as it has been used by XTEA (see [10]). However, this paper provides the first security proof.

two different chains, it is impossible to find j such that $x_j = x'_j \wedge x_{j+1} = x'_{j+1}$, otherwise we will have $x_i = x'_i$ for any i and the two chains are not different.

With these in mind, we introduce the following adaptations: first, we increase the number of rounds used for chain detection to 3, so that given the queries x_i, x_{i+1}, and x_{i+2} to these round functions, a chain can be uniquely specified with the associated key $k = x_i \oplus F_{i+1}(x_{i+1}) \oplus x_{i+2}$, after which it is possible to move forward and backward along the computation path (and do some "completion").

Second, we increase the number of rounds used to ensure randomness. Surrounding each adaptation zone with 2 *always-randomly-defined* buffer rounds is a key point of this framework. The buffer rounds are expected to protect the adaptation zone in the sense that the simulator does not define the values in the 2 buffer rounds while completing other chains. This idea works well in previous contexts. However, in the $SKAF^*$ context, if we continue working with 2 buffer rounds, then since two different chains are possible to collide at two successive rounds, such an expectation may be broken. More clearly, when a chain is to be adapted, the corresponding function values in the buffer rounds may have been defined (this can be shown by a simple operation sequence with only 5 queries; see Appendix A). In such a case, we find it not easy to achieve a proof. To get rid of this, we increase the number of buffer rounds to 4 – more clearly, 2 buffer rounds at each side of each adaptation zone (and in total 8 for the whole construction). We then prove that unless an improbable event happens, the simulator does not define the function values in the buffer rounds *exactly next to* the adaptation zones when completing other chains, and then all chains can be correctly adapted.

Another evidence for the necessity of increasing the number of buffer rounds is that the additional buffer rounds actually play an important role in the proof (see Lemma 2).

At last, to show the indistinguishability of the systems, we combine the *randomness mapping argument* [17] (RMA for short) and its *relaxed* version [2] (RRMA for short). This allows us to bypass the intermediate system composed of the idealized construction and the simulator.

Organization. Sect. 2 presents preliminaries. Sect. 3 contains the main theorem. Sect. 4 gives the simulator. Finally, Sect. 5 sketches the proof. Some additional notations will be introduced later, when necessary.

2 Preliminaries

The Ideal Cipher Model (ICM). The ICM is a widespread model in which all parties have access to a random primitive called ideal cipher $\mathbf{E} : \{0,1\}^n \times \{0,1\}^\kappa \to \{0,1\}^n$. $\mathbf{E}$ is taken randomly from the set of $(2^n!)^{2^\kappa}$ block ciphers with key space $\{0,1\}^\kappa$ and plaintext and ciphertext space $\{0,1\}^n$. ICM finds enormous applications, for instance, the analysis of blockcipher based hash functions (e.g. [13]).

Indifferentiability. The indifferentiability framework was introduced by Maurer, Renner, and Holenstein, at TCC 2004 [23]. It is applicable in settings where the underlying primitives and parameters are exposed to the adversary. Briefly speaking, for a construction $\mathcal{C}^{\mathcal{G}}$ from an idealized primitive $\mathcal{G}$ (hopefully simpler), if $\mathcal{C}^{\mathcal{G}}$ is indifferentiable from another ideal primitive $\mathcal{T}$, then $\mathcal{C}^{\mathcal{G}}$ can safely replace $\mathcal{T}$ in most "natural" settings[3]. A formal definition is recalled as follows.

Definition 1. *A primitive $\mathcal{C}^{\mathcal{G}}$ with oracle access to an ideal primitive $\mathcal{G}$ is said to be statistically and strongly (q, σ, ε)-indifferentiable from an ideal primitive $\mathcal{T}$ if there exists a simulator $\mathbf{S}^{\mathcal{T}}$ s.t. $\mathbf{S}$ makes at most σ queries to $\mathcal{T}$, and for any distinguisher D which issues at most q queries, it holds that*

$$\left| Pr[D^{\mathcal{C}^{\mathcal{G}},\mathcal{G}} = 1] - Pr[D^{\mathcal{T},\mathbf{S}^{\mathcal{T}}} = 1] \right| < \varepsilon$$

Since then, indifferentiability framework has been applied to various constructions, including variants of Merkle-Damgård [13], sponge construction, Feistel [14,17], and iteratated Even-Mansour ciphers [2,18].

3 Indifferentiability for 21-Round Single-Key KAF^*

The main theorem is presented as follows.

Theorem 1. *For any q, the 21-round single-key key-alternating Feistel cipher $SKAF^*_{21}$ with all round functions $\mathbf{F} = (F_1, \ldots, F_{21})$ being 21 independent n-to-n-bit random functions and an identical (user-provided) n-bit key k applied at each round is strongly and statistically (q, σ, ε)-indifferentiable from an ideal cipher $\mathbf{E}$ with $2n$-bit blocks and n-bit keys, where $\sigma = 2^{11} \cdot q^9$ and $\varepsilon \leq \frac{2^{19} \cdot q^{15}}{2^{2n}} + \frac{2^{222} \cdot q^{30}}{2^n} + \frac{2^{34} \cdot q^6}{2^{2n}} = O(\frac{q^{30}}{2^n})$.*

To show it, in the following sections we first give the simulator, then sketch the proof. The full formal proof is deferred to the full version [16].

4 The Simulator

To simplify the proof, we take a strategy introduced by Holenstein *et. al* [17], that is, making the randomness taken by the simulator $\mathbf{S}$, the cipher $\mathbf{E}$ (in the simulated world), and the random functions $\mathbf{F}$ (in the real world) *explicit* as random tapes. The simulator's random tape is an array of tables $\varphi = (\varphi_1, \ldots, \varphi_{21})$, where each φ_i maps entries $x \in \{0,1\}^n$ to uniform and independent values in $\{0,1\}^n$. The cipher's random tape is a table η which encodes an ideal cipher with $2n$-bit blocks and n-bit keys. More clearly, η is selected uniformly at random from all tables with the property of mapping entries $(\delta, k, z) \in \{+,-\} \times \{0,1\}^n \times \{0,1\}^{2n}$

[3] Restrictions on the indifferentiability composition theorem have been exhibited in [26,15]. However, indifferentiability has been sufficient in most "natural" settings (see [15]).

to uniform values $z' \in \{0,1\}^{2n}$ such that $\eta(+,k,z) = z'$ iff. $\eta(-,k,z') = z$. The random functions $\mathbf{F}$ have access to the array of tables $f = (f_1, \ldots, f_{21})$ where each f_i maps entries $x \in \{0,1\}^n$ to uniform and independent values in $\{0,1\}^n$. We denote the constructions/primitives which take randomness from the tapes φ, η, and f by $\mathbf{S}(\varphi)$, $\mathbf{E}(\eta)$, and $\mathbf{F}(f)$ respectively. Among the three, $\mathbf{E}(\eta)$ and $\mathbf{F}(f)$ simply relay the values in η and f. As argued by Andreeva *et al.* [2], such a strategy does not reduce the validity of the simulation, since access to such tapes can be efficiently simulated by uniformly sampling.

We now describe the simulator. $\mathbf{S}(\varphi)$ provides an interface $\mathbf{S}(\varphi).F(i,x)$ to the distinguisher for querying the simulated random function F_i on value x, where $i \in \{1,\ldots,21\}$ and $x \in \{0,1\}^n$. For each i, the simulator maintains a hash table G_i that has entries in the form of pairs (x,y), which denote pairs of inputs and outputs of $\mathbf{S}(\varphi).F(i,x)$. Denote the fact that x is a preimage in the table G_i by $x \in G_i$, and $G_i(x)$ the corresponding image when $x \in G_i$.

Receiving a query $\mathbf{S}(\varphi).F(i,x)$, $\mathbf{S}(\varphi)$ looks in G_i, returns $G_i(x)$ if $x \in G_i$. Otherwise $\mathbf{S}(\varphi)$ accesses the tap φ_i to draw the answer $\varphi_i(x)$ and adds the entry $(x, \varphi_i(x))$ to G_i, and then, if i belongs to the set $\{3, 10, 11, 12, 19\}$, the *chain detection* mechanism and subsequent *chain completion* mechanism of $\mathbf{S}(\varphi)$ will be triggered. These two mechanisms help in ensuring that the answers of the random functions simulated by $\mathbf{S}(\varphi)$ are consistent with the answers of the ideal cipher $\mathbf{E}(\eta)$. Depending on i, there are three case:

1. when $i = 3$, for each newly generated tuple $(x_1, x_2, x_3, x_{20}, x_{21}) \in G_1 \times G_2 \times G_3 \times G_{20} \times G_{21}$, the simulator computes $k := x_1 \oplus G_2(x_2) \oplus x_3$, $x_0 := x_2 \oplus G_1(x_1) \oplus k$, and $x_{22} := x_{20} \oplus G_{21}(x_{21}) \oplus k$. It then calls an inner procedure $\mathbf{S}(\varphi).Check((x_1,x_0),(x_{22},x_{21}),k)$, which checks whether $\mathbf{E}(\eta).Enc(k,(x_1,x_0)) = (x_{22},x_{21})$ (i.e. $\eta(+,k,(x_1,x_0)) = (x_{22},x_{21})$) holds, and returns true if so. Whenever this call returns true, the simulator enqueues a 5-tuple $(x_1, x_2, x_3, 1, 6)$ into a queue *ChainQueue*. In the 5-tuple, the 4-th value 1 informs $\mathbf{S}(\varphi)$ that the first value of the tuple is x_1, and the last value 6 informs $\mathbf{S}(\varphi)$ that when completing the chain $(x_1, x_2, x_3, 1)$, it should set entries in G_6 and G_7 to "adapt" the chain and ensure consistency.
2. when $i = 19$, the case is similar to the previous one by symmetry: for each newly generated tuple $(x_1, x_2, x_{19}, x_{20}, x_{21}) \in G_1 \times G_2 \times G_{19} \times G_{20} \times G_{21}$, the simulator computes $k := x_{19} \oplus G_{20}(x_{20}) \oplus x_{21}$, $x_0 := x_2 \oplus G_1(x_1) \oplus k$, $x_{22} := x_{20} \oplus G_{21}(x_{21}) \oplus k$, and $x_3 := x_1 \oplus G_2(x_2) \oplus k$, makes a call to $\mathbf{S}(\varphi).Check((x_1,x_0),(x_{22},x_{21}),k)$, and enqueues the 5-tuple $(x_1, x_2, x_3, 1, 15)$ into *ChainQueue* whenever this call returns true.
3. when $i \in \{10, 11, 12\}$, for each newly generated tuple $(x_{10}, x_{11}, x_{12}) \in G_{10} \times G_{11} \times G_{12}$, the simulator enqueues the 5-tuple $(x_{10}, x_{11}, x_{12}, 10, l)$ into the queue *ChainQueue*, where $l = 6$ if $i = 10$ or 11, and $l = 15$ if $i = 12$. The sketch of the whole strategy is illustrated in Fig. 2.

After having enqueued the newly generated tuples, $\mathbf{S}(\varphi)$ immediately takes the tuples out of *ChainQueue* and completes the associated partial chains. More clearly, $\mathbf{S}(\varphi)$ maintains a set *CompletedSet* for the chains it has completed. For each chain C dequeued from the queue, if $C \notin CompletedSet$ (i.e. C has

not been completed), $\mathbf{S}(\varphi)$ completes it, by evaluating in the corresponding $SKAF^*$ computation chain both forward and backward (defining the necessary but undefined $G_i(x_i)$ values), and querying $\mathbf{E}.Enc$ or $\mathbf{E}.Dec$ once to "wrap" around, until it reaches the value x_l (when moving forward) and x_{l+1} (when moving backward). Then $\mathbf{S}(\varphi)$ "adapts" the entries by defining $G_l(x_l) := x_{l-1} \oplus x_{l+1} \oplus k$ and $G_{l+1}(x_{l+1}) := x_l \oplus x_{l+2} \oplus k$ to make the entire computation chain consistent with the answers of $\mathbf{E}(\eta)$. This defining action may overwrite values in G_l or G_{l+1} if $x_l \in G_l$ or $x_{l+1} \in G_{l+1}$ before it happens, however we will show the probability to be negligible. $\mathbf{S}(\varphi)$ then adds $(x_1, x_2, x_3, 1)$ and $(x_{10}, x_{11}, x_{12}, 10)$ to $CompletedSet$, where the two chains correspond to C.

During the completion, the values in G_j newly defined by $\mathbf{S}(\varphi)$ also trigger the chain detection mechanism and chain completion mechanism when $j \in \{3, 10, 11, 12, 19\}$. $\mathbf{S}(\varphi)$ hence keeps dequeuing and completing until $ChainQueue$ is empty again. $\mathbf{S}(\varphi)$ finally returns $G_i(x)$ as the answer to the initial query.

Pseudocode of the Simulator. We present the pseudocode of the simulator $\mathbf{S}(\varphi)$ and a modified simulator $\widetilde{S}(\varphi)$ (will be introduced in Sect. 5). When a line has a boxed statement next to it, $\mathbf{S}(\varphi)$ uses the original statement, while $\widetilde{S}(\varphi)$ uses the boxed one.

1: **Simulator** $\mathbf{S}(\varphi)$: **Simulator** $\widetilde{S}(\varphi)$:
2: **Variables**
3: hash tables $\{G_i\} = (G_1, \ldots, G_{21})$, initially empty
4: queue $ChainQueue$, initially empty
5: set $CompletedSet$, initially empty
The procedure $F(i, x)$ provides an interface to the distinguisher.
6: **public procedure** $F(i, x)$
7: $y := F^{inner}(i, x)$
8: **while** $ChainQueue \neq \emptyset$ **do**
9: $(x_j, x_{j+1}, x_{j+2}, j, l) := ChainQueue.Dequeue()$
10: **if** $(x_j, x_{j+1}, x_{j+2}, j, l) \notin CompletedSet$ **then** // Complete the chain
11: $k := x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2}$
12: $(x_{l-4}, x_{l-3}, x_{l-2}, l-4) := EvalForward(x_j, x_{j+1}, x_{j+2}, j, l-4)$
13: $(x_{l+3}, x_{l+4}, x_{l+5}, l+3) := EvalBackward(x_j, x_{j+1}, x_{j+2}, j, l+3)$
14: $Adapt(x_{l-4}, x_{l-3}, x_{l-2}, x_{l+3}, x_{l+4}, x_{l+5}, l)$
15: $(x_1, x_2, x_3, 1) := EvalForward(x_j, x_{j+1}, x_{j+2}, j, 1)$
16: $(x_{10}, x_{11}, x_{12}, 10) := EvalForward(x_1, x_2, x_3, 1, 10)$
17: $CompletedSet := CompletedSet \cup \{(x_1, x_2, x_3, 1), (x_{10}, x_{11}, x_{12}, 10)\}$
18: **return** y
The procedure $Adapt$ randomly sets the "missed" values if necessary and adds entries to G_l and G_{l+1} to make the chain match the computation.
19: **private procedure** $Adapt(x_{l-4}, x_{l-3}, x_{l-2}, x_{l+3}, x_{l+4}, x_{l+5}, l)$
20: $k := x_{l-4} \oplus G_{l-3}(x_{l-3}) \oplus x_{l-2}$
21: $y_{l-2} := F^{inner}(l-2, x_{l-2})$
22: $x_{l-1} := x_{l-3} \oplus y_{l-2} \oplus k$
23: $y_{l-1} := F^{inner}(l-1, x_{l-1})$

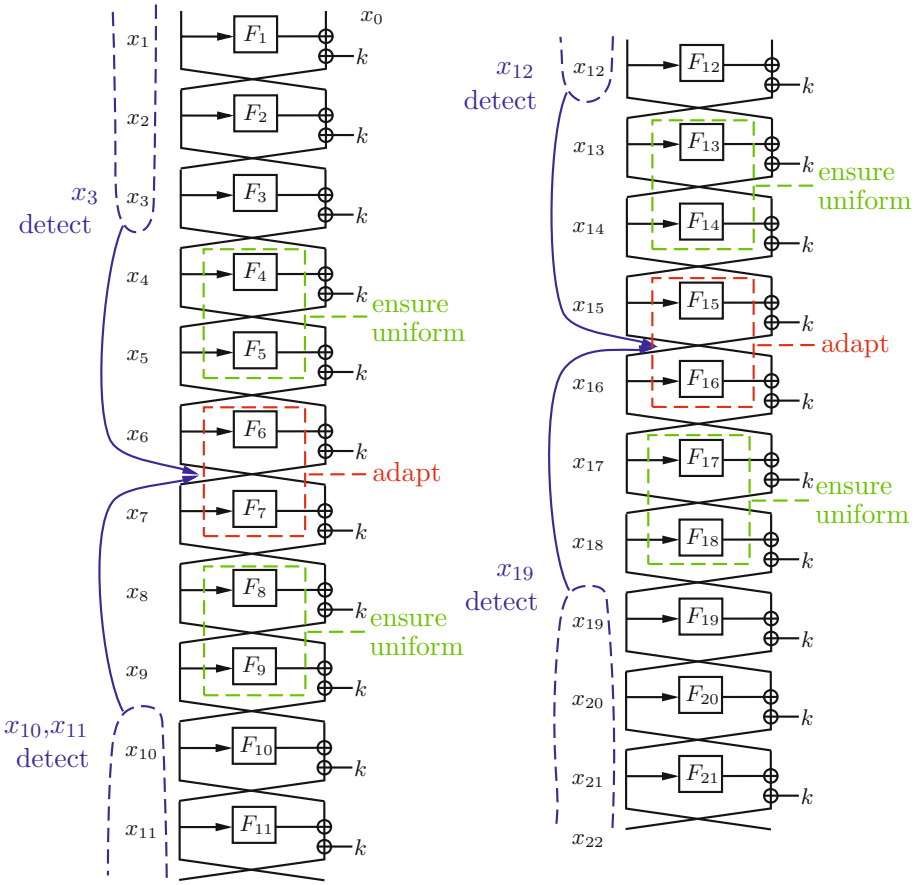

Fig. 2. The 21-round $SKAF^*$ cipher with the zones where the simulator detects chains and adapts them0

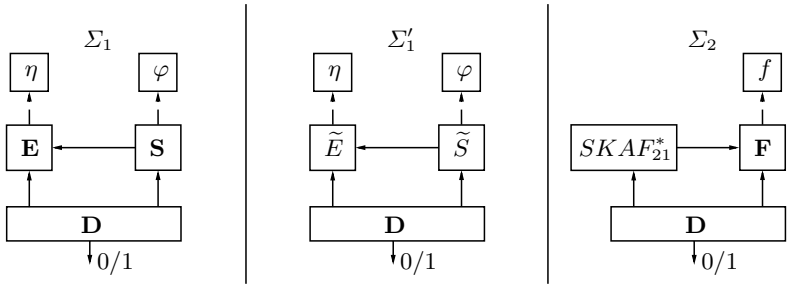

Fig. 3. Systems used in the proof in this paper

24: $x_l := x_{l-2} \oplus y_{l-1} \oplus k$
25: $y_{l+3} := F^{inner}(l+3, x_{l+3})$
26: $x_{l+2} := x_{l+4} \oplus y_{l+3} \oplus k$
27: $y_{l+2} := F^{inner}(l+2, x_{l+2})$
28: $x_{l+1} := x_{l+3} \oplus y_{l+2} \oplus k$
29: $ForceVal(x_l, x_{l-1} \oplus x_{l+1} \oplus k, l)$
30: $ForceVal(x_{l+1}, x_l \oplus x_{l+2} \oplus k, l+1)$
31: **private procedure** $ForceVal(x, y, l)$
32: $G_l(x) := y$ // May overwrite the entry $G_l(x)$

The procedure F^{inner} draws answers from the table G_i, or the tape φ_i if the answers have not been defined in G_i, and enqueue chains when necessary.

33: **private procedure** $F^{inner}(i, x)$
34: **if** $x \notin G_i$ **then**
35: $G_i(x) := \varphi_i(x)$
36: **if** $i \in \{3, 10, 11, 12, 19\}$ **then**
37: $EnqueueNewChains(i, x)$
38: **return** $G_i(x)$
39: **private procedure** $EnqueueNewChains(i, x)$
40: **if** $i = 3$ **then**
41: **for all** $(x_1, x_2, x_3, x_{20}, x_{21}) \in G_1 \times G_2 \times \{x\} \times G_{20} \times G_{21}$ **then**
42: $k := x_1 \oplus G_2(x_2) \oplus x_3$
43: $chk\_pa := ((x_1, G_1(x_1) \oplus x_2 \oplus k), (x_{20} \oplus G_{21}(x_{21}) \oplus k, x_{21}), k)$
44: $flag := Check(chk\_pa)$ $\boxed{flag := \widetilde{E}.Check(chk\_pa)}$
45: **if** $flag = true$ **then**
46: $ChainQueue.Enqueue(x_1, x_2, x_3, 1, 6)$
47: **else if** $i = 19$ **then**
48: **for all** $(x_1, x_2, x_{19}, x_{20}, x_{21}) \in G_1 \times G_2 \times \{x\} \times G_{20} \times G_{21}$ **do**
49: $k := x_{19} \oplus G_{20}(x_{20}) \oplus x_{21}$
50: $chk\_pa := ((x_1, G_1(x_1) \oplus x_2 \oplus k), (x_{20} \oplus G_{21}(x_{21}) \oplus k, x_{21}), k)$
51: $flag := Check(chk\_pa)$ $\boxed{flag := \widetilde{E}.Check(chk\_pa)}$
52: **if** $flag = true$ **then**
53: $x_3 := x_1 \oplus G_2(x_2) \oplus k$
54: $ChainQueue.Enqueue(x_1, x_2, x_3, 1, 15)$
55: **else if** $i = 10$ **then**
56: **for all** $(x_{10}, x_{11}, x_{12}) \in \{x\} \times G_{11} \times G_{12}$ **do**
57: $ChainQueue.Enqueue(x_{10}, x_{11}, x_{12}, 10, 6)$
58: **else if** $i = 11$ **then**
59: **for all** $(x_{10}, x_{11}, x_{12}) \in G_{10} \times \{x\} \times G_{12}$ **do**
60: $ChainQueue.Enqueue(x_{10}, x_{11}, x_{12}, 10, 6)$
61: **else if** $i = 12$ **then**
62: **for all** $(x_{10}, x_{11}, x_{12}) \in G_{10} \times G_{11} \times \{x\}$ **do**
63: $ChainQueue.Enqueue(x_{10}, x_{11}, x_{12}, 10, 15)$

The *Check* procedure queries $\mathbf{E}$ to verify whether the inputs are valid pairs of plaintext and ciphertext of $\mathbf{E}$. Note that $\widetilde{S}$ does not own *Check* procedure; instead $\widetilde{S}$ calls the *Check* procedure of a modified cipher $\widetilde{E}$.

64: **private procedure** $Check(x, y, k)$ // $\widetilde{S}$ does not own such a procedure
65: **return** $\mathbf{E}.Enc(k, x) = y$

The procedures *EvalForward* (and *EvalBackward*, resp.) takes a partial chain $(x_j, x_{j+1}, x_{j+2}, j)$ as input, and evaluate forward (and backward, resp.) in $SKAF^*$ until obtaining the tuple (x_l, x_{l+1}, x_{l+2}) of input values for G_l, G_{l+1}, and G_{l+2} for specified l.

66: **private procedure** $EvalForward(x_j, x_{j+1}, x_{j+2}, j, l)$
67: $k := x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2}$ // By construction $x_{j+1} \in G_{j+1}$ holds
68: **while** $j \neq l$ **do**
69: **if** $j = 20$ **then**
70: $(x_1, x_0) := \mathbf{E}.Dec(k, (x_{22}, x_{21}))$ $\boxed{(x_1, x_0) := \widetilde{E}.Dec(k, (x_{22}, x_{21}))}$
71: $x_2 := x_0 \oplus F^{inner}(1, x_1) \oplus k$
72: $j := 0$
73: **else**
74: $x_{j+3} := x_{j+1} \oplus F^{inner}(j + 2, x_{j+2}) \oplus k$
75: $j := j + 1$
76: **return** $(x_l, x_{l+1}, x_{l+2}, l)$
77: **private procedure** $EvalBackward(x_j, x_{j+1}, x_{j+2}, j, l)$
78: $k := x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2}$
79: **while** $j \neq l$ **do**
80: **if** $j = 0$ **then**
81: $(x_{22}, x_{21}) := \mathbf{E}.Enc(k, (x_1, x_0))$ $\boxed{(x_{22}, x_{21}) := \widetilde{E}.Enc(k, (x_1, x_0))}$
82: $x_{20} := x_{22} \oplus F^{inner}(21, x_{21}) \oplus k$
83: $j := 20$
84: **else**
85: $x_{j-1} := x_{j+1} \oplus F^{inner}(j, x_j) \oplus k$
86: $j := j - 1$
87: **return** $(x_l, x_{l+1}, x_{l+2}, l)$

5 Proof of the Indifferentiability: Sketch

Denote by $\Sigma_1(\mathbf{E}(\eta), \mathbf{S}(\varphi))$ the simulated system composed of the ideal cipher $\mathbf{E}$ with tape η and the simulator $\mathbf{S}$ with tape φ, and denote by $\Sigma_2(SKAF^*_{21}, \mathbf{F}(f))$ the real system composed of $SKAF^*_{21}$ and the random functions $\mathbf{F}(f)$. Then, for any fixed, deterministic, and computationally unbounded distinguisher $\mathbf{D}$, we show the following two to establish the indifferentiability:

(i) $\Sigma_1(\mathbf{E}(\eta), \mathbf{S}(\varphi))$ and $\Sigma_2(SKAF^*_{21}, \mathbf{F}(f))$ are indistinguishable.
(ii) With overwhelmingly large probability, $\mathbf{S}(\varphi)$ runs in polynomial time.

Note that the underlying ideas used in Sect. 5.2 and Sect. 5.3 are originally used by Coron *et al.* [14] (also used in [17,18]). As stressed in Introduction, the main

novelties of this part are in Sect. 5.4. To keep Sect. 5.4 clear and simple, we only present the sketch in the main body, while extracting the lemmas corresponding to the core step (the simulator overwrites with negligible probability) and listing them in Appendix B.

5.1 An Intermediate System Σ_1'

We use an intermediate system $\Sigma_1'(\widetilde{E}(\eta), \widetilde{S}(\varphi))$, which consists of a modified ideal cipher $\widetilde{E}(\eta)$ and a slightly modified simulator $\widetilde{S}(\varphi)$. $\widetilde{E}(\eta)$ maintains a table E, which contains entries of the form $((+,k,x),y)$ and $((-,k,y),x)$, and is initially empty. $\widetilde{E}(\eta)$ provides an additional interface $Check(x,y,k)$. Once being queried on $Enc(k,x)$ or $Dec(k,y)$, $\widetilde{E}(\eta)$ adds the corresponding entries in η to E and returns them as answers. Once being called on $Check(x,y,k)$, $\widetilde{E}(\eta)$ looks in the table E to check whether $E(+,k,x)=y$ and returns the answer. On the other hand, the differences between $\widetilde{S}(\varphi)$ and $\mathbf{S}(\varphi)$ consist of two aspects:

- the cipher queried by them: $\widetilde{S}(\varphi)$ queries $\widetilde{E}(\eta)$ while $\mathbf{S}(\varphi)$ queries $\mathbf{E}(\eta)$;
- the owner of the $Check$ procedure called by them: $\widetilde{S}(\varphi)$ calls $\widetilde{E}(\eta).Check$ while $\mathbf{S}(\varphi)$ calls $\mathbf{S}(\varphi).Check$;

Denote by $\widetilde{E}(\eta).E^+$ the set of entries in $\widetilde{E}(\eta).E$ of the form $((+,\cdot,\cdot),\cdot)$. The pseudocode of $\widetilde{E}(\eta)$ is deferred to the full version [16], while the pseudocode of $\widetilde{S}(\varphi)$ is presented along with $\mathbf{S}(\varphi)$, in Sect. 4, captured by the boxed statements. The three systems are depicted in Fig. 3. Σ_1' mostly helps in bounding the complexity of $\mathbf{S}(\varphi)$.

5.2 Bounding the Complexity of $\widetilde{S}(\varphi)$ in Σ_1'

The simulator $\widetilde{S}(\varphi)$ in Σ_1' runs in polynomial time: each time $\widetilde{S}(\varphi)$ dequeues a tuple of the form $(x_1,x_2,x_3,1,l)$ for which $(x_1,x_2,x_3,1)\notin CompletedSet$ must correspond to an entry in $\widetilde{E}(\eta).E^+$ previously added during a query issued by $\mathbf{D}$, since $(x_1,x_2,x_3,1,l)$ can be enqueued only when the corresponding call to $\widetilde{E}(\eta).Check$ returns true. Hence such dequeuing happens at most q times. Based on this, the size of G_i and $\widetilde{E}(\eta).E^+$ is upper bounded to $10q^3$, and the number of queries to $\widetilde{E}(\eta).Check$ issued by $\widetilde{S}(\varphi)$ is upper bounded to $2\cdot(10q^3)^5$.

5.3 Σ_1 to Σ_1'

To show this, we define a bad event to capture the difference between Σ_1 and Σ_1'. The core difference between the two lies in that the procedure $\mathbf{S}(\varphi).Check$ actually answers queries according to the content of the table/tape η, while the procedure $\widetilde{E}(\eta).Check$ answers according to a much smaller table $\widetilde{E}(\eta).E$: this forms the idea of the bad event. During an execution $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$, the bad event **BadCheck** happens if $\exists(x,y,k)$ s.t. all the following hold:

(i) $\mathbf{S}(\varphi)$ makes a call $\mathbf{S}(\varphi).Check(x, y, k)$.
(ii) $\eta(+, k, x) = y$.
(iii) Before the call in (i), neither $\mathbf{E}(\eta).Enc(k, x)$ nor $\mathbf{E}(\eta).Dec(k, y)$ has been issued (i.e. if the call is made in Σ_1', then $(+, k, x) \notin \widetilde{E}(\eta).E$ before the call).

For some fixed (η, φ), assume that $\widetilde{S}(\varphi)$ makes q_1' calls to $\widetilde{S}(\varphi).Check$ during $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$. Then if during $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$, **BadCheck** does not happen in the first q_1' calls to $\mathbf{E}(\eta).Check$ – the probability is at least $1-2q_1'/2^{2n}$, $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$ and $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ behave the same way, and $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))} = \mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$. Since $q_1' \leq 2 \cdot (10q^3)^5$, we have:

Lemma 1. *For any distinguisher* $\mathbf{D}$ *which issues at most* q *queries, we have:*

$$\left| Pr[\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))} = 1] - Pr[\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))} = 1] \right| \leq \frac{2^{19} \cdot q^{15}}{2^{2n}}.$$

Proof. See the full version [16]. □

This bound on advantage along with the bound on complexity of $\widetilde{S}(\varphi)$ show that with probability at least $1 - \frac{2^{19} \cdot q^{15}}{2^{2n}}$, the complexity of $\mathbf{S}(\varphi)$ is the same as that of $\widetilde{S}(\varphi)$, and can be upper bounded to $2^{11} \cdot q^9$ queries to $\mathbf{E}(\eta)$.

5.4 Σ_1' to Σ_2: The Relaxed Randomness Mapping Argument

We use an RRMA to fill in this part. First, we specify the domain of the randomness map; second, we complete the argument.

Specifying the Domain of the Map. The domain of the map should include overwhelmingly many Σ_1' executions, and these executions should have the same behaviors as the Σ_2 executions in the view of $\mathbf{D}$. Hence we figure out the difference between the two systems first. Consider Σ_1' and Σ_2. In the former, the answers to F-queries are simulated by $\widetilde{S}(\varphi)$, and when $\widetilde{S}(\varphi)$ is forced to overwrite some entries (in $\{G_i\}$), the consistency in the answers will be broken. On the other hand, such inconsistency never appears in Σ_2: this forms the difference. We will take the Σ_1' executions during which $\widetilde{S}$ does not overwrite any entry to specify the domain (later we will show that such Σ_1' executions are the same as the Σ_2 executions in the view of $\mathbf{D}$). For this, we first define an additional bad event **BadHit**, then show that **BadHit** happens with negligible probability, and finally show that during a Σ_1'-execution, if **BadHit** does not happen, then $\widetilde{S}$ does not overwrite, so that the domain we specify covers overwhelmingly many Σ_1' executions as expected. During an execution $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, the event **BadHit** happens if the n-bit value read from the tape φ – or either of the two n-bit halves of the value read from the tape η – equals the xor of 9 or less values in the history $\mathcal{H}$, where $\mathcal{H}$ is the set of all the n-bit values – or halves – extracted from the tables $\{G_i\}$ and $\widetilde{E}(\eta).E$ right before the tape accessing action happens. With the bound on the size of the tables, we calculate $Pr[\mathbf{BadHit}] \leq \frac{2^{88} \cdot q^{30}}{2^n}$.

We then show that during the *good* executions $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ (during which **BadHit** does not happen), $\widetilde{S}(\varphi)$ never overwrites any entry in $\{G_i\}$. As mentioned in Sect. 1, the reason is that right before any call to *Adapt*, the function values in the two buffer rounds exactly next to the adaptation zone must have not been defined. Then the two values will be set to uniformly random values, which implies that the probability for $\widetilde{S}(\varphi)$ to overwrite is negligible. To illustrate more clearly, we define the 4-tuple $(x_i, x_{i+1}, x_{i+2}, i)$ as *partial chain* for $i \in \{0, \ldots, 20\}$, and borrow the helper functions *next*, *prev*, val_l^+, val_l^-, and the notions *equivalent* partial chains, *table-defined* partial chains from [17,18]. The two helper functions *next* and *prev* take a partial chain C as input and return the partial chain obtained by moving respectively one step forward or backward in $SKAF_{21}^*$ according to the given tables $\widetilde{E}(\eta).E$ and $\{G_i\}$ (wrapping around through $\widetilde{E}(\eta).E$ if necessary), or empty value $\perp$ when some values are necessary for the computation but have not been defined in the tables. The two functions val_l^+ and val_l^- take a partial chain as input and evaluate forward and backward respectively (also according to the given $\widetilde{E}.E$ and $\{G_i\}$) until obtaining and returning the corresponding x_l, or returning $\perp$ when some necessary values have not been defined in the tables. The notions *equivalent* and *table-defined* partial chains are as follows: (i) two partial chains C and D are *equivalent*, if they belong to the same computation chain; (ii) a partial chain $C = (x_i, x_{i+1}, x_{i+2}, i)$ is *table-defined* if all the three values x_i, x_{i+1}, and x_{i+2} have been added to their corresponding tables.

Then, we have the following non-overwriting lemma.

Lemma 2. *In a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*, before any two successive calls to* $ForceVal(x_l, y_l, l)$ *and* $ForceVal(x_{l+1}, y_{l+1}, l+1)$*,* $x_l \notin G_l \wedge x_{l+1} \notin G_{l+1}$ *must hold.*

Proof. A formal proof – along with the lemmas that support the proof – are presented in Appendix B; here we only sketch it. Consider any such two calls $ForceVal(x_l, y_l, l)$ and $ForceVal(x_{l+1}, y_{l+1}, l+1)$, and suppose that they are triggered by a chain C. Note that $C \notin CompletedSet$ when C is dequeued, otherwise the two calls will not happen. Then the sketch consists of four stages:

First, denote by $Path_C$ the whole computation path that C belongs to. Then before C is enqueued, $val_{l-2}^+(C) = \perp \wedge val_{l+3}^-(C) = \perp$ must hold. Otherwise the values of $Path_C$ must have "filled" another chain detection zone, after which $Path_C$ would be completed, and C would have been added to $CompletedSet$, a contradiction.

Second, since being enqueued, C must have been equivalent to a table-defined chain. Then, during the completion of some other chain D (D is not equivalent to C and is completed after C being enqueued), the subsequent calls to $ForceVal$ cannot affect $val_i^\delta(C)$ for any valid i and $\delta \in \{+, -\}$. The underlying reason is strongly relevant to the number of buffer rounds we arrange. Briefly speaking, for previous calls to $ForceVal$ (triggered by D) to change $val_i^\delta(C)$, C and D must agree on either of the two rounds l' and $l'+1$ where D is supposed to be adapted. However since we arrange two buffer rounds at each side of the adaptation zone,

we find that two inequivalent partial chains (C and D) either: (i) cannot collide at three consecutive rounds, and as a result, $val^{+}_{l'-1}(C) \neq val^{+}_{l'-1}(D)$ ($val^{-}_{l'+2}(C) \neq val^{-}_{l'+2}(D)$, resp.); or: (ii) cannot collide at round $l'-1$ ($l'+2$, resp.) to avoid the bad event **BadHit**. By this, C and D cannot agree on any of the two rounds l' and $l'+1$, and $val^{\delta}_{i}(C)$ can only be changed by tape accessing and table entry setting actions, and to avoid **BadHit**, $val^{+}_{l-2}(C) \notin G_{l-2} \wedge val^{-}_{l+3}(C) \notin G_{l+3}$ immediately holds after such actions make them two non-empty.

Third, by carefully analyzing all possibilities, we show that any chain completed between the point when C is enqueued and the point when C is dequeued cannot add $val^{+}_{l-1}(C)$ to G_{l-1} (even if $val^{+}_{l-1}(C)$ has previously been made non-empty during the completion of some other chains). Similarly, $val^{-}_{l+2}(C)$ cannot be added to G_{l+2} during this period. Hence $val^{+}_{l-1}(C) \notin G_{l-1} \wedge val^{-}_{l+2}(C) \notin G_{l+2}$ keeps holding till C being dequeued.

Finally, after C is dequeued, $G_{l-1}(val^{+}_{l-1}(C))$ and $G_{l+2}(val^{-}_{l+2}(C))$ will be defined to values from φ tape, and to avoid **BadHit**, $x_l = val^{+}_{l}(C) \notin G_l \wedge x_{l+1} = val^{-}_{l+1}(C) \notin G_{l+1}$ must hold after these assignments. □

Completing the RRMA. Fix a distinguisher $\mathbf{D}$. Consider a distinguisher $\overline{D}$ which runs $\mathbf{D}$ and then completes all the chains for each query to $\widetilde{E}(\eta)$ made by $\mathbf{D}$. During $\overline{D}^{\Sigma'_1(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, many entries in the tapes (η, φ) may not be accessed, and those that are really accessed compose *footprints*. Clearly with respect to $\overline{D}$, there is a bijection between the footprint set and the Σ'_1 execution set.

Then, consider the Σ'_1 executions $\overline{D}^{\Sigma'_1(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ during which $\widetilde{S}(\varphi)$ does not overwrite any entry. By Lemma 2, the probability for such Σ'_1 executions to occur is at least $1 - \frac{2^{222} \cdot q^{30}}{2^n}$. Taking the set of all possible footprints of such Σ'_1 executions as the domain, we define $\tau(\alpha) = f = (f_1, \ldots, f_{21})$ as the exact copies of the tables $(G_1, \ldots, G_{21})$ standing at the end of the execution: $\tau(\alpha) \equiv \{G_i\}$. The Σ'_1 and Σ_2 executions linked by τ have the same behaviors in the view of $\overline{D}$ because the answers in them two are consistent with $\tau(\alpha)$ and $\{G_i\}$; and the probabilities for the tapes (η, φ) and f to respectively agree with the preimage and the image are close – for $22q$-query $\overline{D}$, the ratio of the two probabilities lies in the interval $[1 - \frac{(10(22q)^3)^2}{2^{2n}}, 1]$. By these, with a *nearly* standard process, we upper bound the advantage of distinguishing Σ'_1 and Σ_2 to $\frac{2^{222} \cdot q^{30}}{2^n} + \frac{2^{34} \cdot q^6}{2^{2n}}$.

Lemma 3. *For any distinguisher* $\mathbf{D}$ *which issues at most* q *queries, we have:*

$$\left| Pr[\mathbf{D}^{\Sigma'_1} = 1] - Pr[\mathbf{D}^{\Sigma_2} = 1] \right| \leq \frac{2^{222} \cdot q^{30}}{2^n} + \frac{2^{34} \cdot q^6}{2^{2n}}$$

Proof. See the full version [16]. □

Acknowledgements. We are deeply grateful to the anonymous referees of TCC 2015 for their useful comments. We are also particularly grateful to Jianghua Zhong. This paper would have not been possible without her help.

This work is partially supported by National Key Basic Research Project of China (2011CB302400), National Science Foundation of China (61379139) and the "Strategic Priority Research Program" of the Chinese Academy of Sciences, Grant No. XDA06010701.

References

1. Abed, F., List, E., Lucks, S., Wenzel, J.: Differential cryptanalysis of round-reduced simon and speck. In: Fast Software Encryption 2014. LNCS. Springer, Heidelberg (2014) (to appear)
2. Andreeva, E., Bogdanov, A., Dodis, Y., Mennink, B., Steinberger, J.P.: On the indifferentiability of key-alternating ciphers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 531–550. Springer, Heidelberg (2013)
3. Andreeva, E., Bogdanov, A., Mennink, B.: Towards understanding the known-key security of block ciphers. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 348–366. Springer, Heidelberg (2014)
4. Aysum, A., Gulcan, E., Schaumont, P.: Simon says, break the area records for symmetric key block ciphers on fpgas. Tech. rep., Cryptology ePrint Archive, Report 2014/237 (2014), `http://eprint.iacr.org`
5. Barbosa, M., Farshim, P.: The related-key analysis of feistel constructions. In: Fast Software Encryption 2014. LNCS. Springer, Heidelberg (2014) (to appear)
6. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck families of lightweight block ciphers
7. Biryukov, A., Nikolić, I.: Complementing feistel ciphers. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 3–18. Springer, Heidelberg (2014)
8. Biryukov, A., Roy, A., Velichkov, V.: Differential analysis of block ciphers simon and speck. In: Fast Software Encryption 2014. LNCS. Springer, Heidelberg (2014) (to appear)
9. Black, J.A.: The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 328–340. Springer, Heidelberg (2006)
10. Bouillaguet, C., Dunkelman, O., Leurent, G., Fouque, P.-A.: Another look at complementation properties. In: Hong, S., Iwata, T. (eds.) FSE 2010. LNCS, vol. 6147, pp. 347–364. Springer, Heidelberg (2010)
11. Boura, C., Naya-Plasencia, M., Suder, V.: Scrutinizing and improving impossible differential attacks: Applications to CLEFIA, Camellia, LBlock and SIMON. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, PART I. LNCS, vol. 8873, pp. 179–199. Springer, Heidelberg (2014)
12. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM 51(4), 557–594 (2004)
13. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-damgård revisited: How to construct a hash function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005)
14. Coron, J.-S., Patarin, J., Seurin, Y.: The random oracle model and the ideal cipher model are equivalent. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 1–20. Springer, Heidelberg (2008)
15. Demay, G., Gaži, P., Hirt, M., Maurer, U.: Resource-restricted indifferentiability. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 664–683. Springer, Heidelberg (2013)

16. Guo, C., Lin, D.: On the indifferentiability of key-alternating feistel ciphers with no key derivation. Cryptology ePrint Archive, Report 2014/786 (2014), http://eprint.iacr.org/
17. Holenstein, T., Künzler, R., Tessaro, S.: The equivalence of the random oracle model and the ideal cipher model, revisited. In: Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing, STOC 2011, pp. 89–98. ACM, New York (2011)
18. Lampe, R., Seurin, Y.: How to construct an ideal cipher from a small set of public permutations. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 444–463. Springer, Heidelberg (2013)
19. Lampe, R., Seurin, Y.: Security analysis of key-alternating feistel ciphers. In: Fast Software Encryption 2014. LNCS. Springer, Heidelberg (2014) (to appear)
20. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal on Computing 17(2), 373–386 (1988)
21. Mandal, A., Patarin, J., Seurin, Y.: On the public indifferentiability and correlation intractability of the 6-round feistel construction. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 285–302. Springer, Heidelberg (2012)
22. Maurer, U., Pietrzak, K.: The security of many-round luby-rackoff pseudorandom permutations. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 544–561. Springer, Heidelberg (2003)
23. Maurer, U.M., Renner, R.S., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 21–39. Springer, Heidelberg (2004)
24. Patarin, J.: Pseudorandom permutations based on the D.E.S. scheme. In: Charpin, P., Cohen, G. (eds.) EUROCODE 1990. LNCS, vol. 514, pp. 193–204. Springer, Heidelberg (1991)
25. Patarin, J.: Security of random feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 106–122. Springer, Heidelberg (2004)
26. Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with composition: Limitations of the indifferentiability framework. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 487–506. Springer, Heidelberg (2011)
27. Seurin, Y.: Primitives et protocoles cryptographiques àsécurité prouvée. Ph.D. thesis, PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, France (2009)
28. Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (Related-key) differential characteristic search: Application to SIMON, PRESENT, lBlock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, PART I. LNCS, vol. 8873, pp. 158–178. Springer, Heidelberg (2014)

A Surrounding Each Adaptation Zone with Two Buffer Rounds – the Broken Expectations

If we increase the number of rounds used for chain detection to 3, while continue surrounding each adaptation zone with two buffer Rounds – exactly same as done in the previous works [17,18] – then we are working on $3+1+2+1+3+1+2+1+3=17$ rounds ($SKAF^{*}_{17}$). For the modified simulator, the buffer rounds are round 4, 7, 11, and 14, while the first adaptation zone consists of round 5

and 6, the second consists of round 12 and 13. Then the following operation sequence shows that when a chain is to be adapted, the function values in the buffer rounds next to the adaption zone may have been defined:

(i) arbitrarily chooses x_3, x_2, and $x_2^{'}$;
(ii) issues queries $G_2(x_2)$ and $G_2(x_2^{'})$ to the simulator;
(iii) arbitrarily chooses k and calculate $k^{'} := k \oplus x_2 \oplus x_2^{'}$;
(iv) calculates $x_1 := x_3 \oplus G_2(x_2) \oplus k$ and $x_1^{'} := x_3 \oplus G_2(x_2^{'}) \oplus k^{'}$;
(v) issues queries $G_1(x_1)$ and $G_1(x_1^{'})$;
(vi) issues queries $G_3(x_3)$;

The last query $G_3(x_3)$ enqueues two chains $(x_1, x_2, x_3, 1)$ and $(x_1^{'}, x_2^{'}, x_3, 1)$, and whatever value is assigned to $G_3(x_3)$, for the two chains we have $x_4 = x_2 \oplus G_3(x_3) \oplus k = x_2^{'} \oplus G_3(x_3) \oplus k^{'} = x_4^{'}$. When the later one is dequeued, we have $x_4 \in G_4$; this breaks the expectation that *the simulator does not define the values in the buffer rounds while completing other chains*. The underlying reason for this lies in the fact that in the $SKAF^*$ context, it is possible to make two different chains collide at two successive rounds (as already discussed in Introduction). The operation sequence mentioned before indeed takes advantage of this property.

However, at current time, we are not clear whether 17-round single-key $SKAF_{17}^*$ can achieve indifferentiability or not.

B The Formal Proof for $\widetilde{S}$ Not Overwrites

To give the formal proof, we introduce two notions *key-defined* and *key-undefined* chains and a helper function k, as follows: a partial chain $C = (x_i, x_{i+1}, x_{i+2}, i)$ is called *key-defined* if $x_{i+1} \in G_{i+1}$, otherwise is called *key-undefined*; and $k(C)$ returns the associated key when C is key-defined, while returning $\perp$ otherwise. Moreover, we borrow two additional notions *safe call to Adapt*, and *non-overwriting call to ForceVal* from [17,18]: (i) a call to $Adapt(x_{l-4}, x_{l-3}, x_{l-2}, x_{l+3}, x_{l+4}, x_{l+5}, l)$ is *safe* if the following holds before the call:

$$(((x_{l-2} \notin G_{l-2}) \vee (x_{l-2} \in G_{l-2} \wedge x_{l-3} \oplus G_{l-2}(x_{l-2}) \oplus k(B) \notin G_{l-1}))$$
$$\wedge((x_{l+3} \notin G_{l+3}) \vee (x_{l+3} \in G_{l+3} \wedge x_{l+4} \oplus G_{l+3}(x_{l+3}) \oplus k(D) \notin G_{l+2}))),$$

where $B = (x_{l-4}, x_{l-3}, x_{l-2}, l-4)$ and $D = (x_{l+3}, x_{l+4}, x_{l+5}, l+3)$; (ii) a call to $ForceVal(x, y, l)$ is *non-overwriting* if $x \notin G_l$ before the call.

Then, we have Lemma 11, which claims that $\widetilde{S}(\varphi)$ does not overwrite in good Σ_1' executions. Before presenting this main lemma, we list some properties of the good Σ_1' executions, as follows. They consist of the foundation of Lemma 11. To save space while highlighting the features of $SKAF^*$, we summarize the properties that are *almost the same* as those owned by un-keyed Feistel [17] and single-key iterated Even-Mansour [18] as Lemma 5.

First, in the good executions, each random tape accessing and the subsequent entry setting action can only extend the key-defined chains one round. Compared to the previous results in [17,18], Lemma 4 only focuses on *key-defined* chains.

Lemma 4. *The following hold in a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*:*

(i) For any key-defined partial chain C*, if* $next(C) = \perp$ *(*$prev(C) = \perp$*, resp.) before a random tape accessing and subsequent entry setting action on either* $\widetilde{E}(\eta).E$ *or* $\{G_i\}$*, then if* C *is table-defined after the action, it holds that* $next^2(C) = \perp$ *(*$prev^2(C) = \perp$*, resp.).*

(ii) For any key-defined partial chain C *and each* $\delta \in \{+,-\}$*, a random tape accessing and entry setting action* $G_j(x_j) := \varphi_j(x_j)$ *can only change at most one of the values* $val_i^\delta(C)$*; and if such change happens, then:*
- *the value is changed from* $\perp$ *to some non-empty values.*
- *if* $\delta = +$*,* $i = j + 1$*; if* $\delta = -$*,* $i = j - 1$*.*
- $val_j^\delta(C) = x_j$ *before the assignment.*
- *after the action, if* C *is table-defined, then* $val_i^\delta(C) \notin G_i$*.*

Proof. See the full version [16]. □

Lemma 5. *Informally speaking, during a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*, we have:*

(i) the relation $\equiv$ *between partial chains is an equivalence relation;*

(ii) the relation $\equiv$ *between table-defined chains is invariant before and after the random tape accessing and subsequent entry setting action;*

(iii) if a chain C *is dequeued such that* $C \notin CompletedSet$*, then when* C *was enqueued, no chain equivalent to* C *had been enqueued.*

Proof. See *Lemma 9*, *Lemma 10*, and *Lemma 13* in the full version [16]. □

The following lemma claims that two inequivalent chains cannot collide at two consecutive rounds when they are extended by the random tape accessing and entry setting actions.

Lemma 6. *Fix a point in a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ *and suppose all calls to* $ForceVal$ *to be non-overwriting up to this point. Assume that a random tape accessing and entry setting action* $G_i(x_i) := \varphi_i(x_i)$ *happens right after this point, then for any two key-defined partial chains* C *and* D*, any* $l \in \{3, \ldots, 19\}$*, and any* $\delta \in \{+,-\}$*, the following four cannot be simultaneously fulfilled:*

(i) before the action, C *is not equivalent to* D*;*

(ii) before the action, $val_l^\delta(C) = \perp$ *or* $val_l^\delta(D) = \perp$*;*

(iii) after the action, C *and* D *are table-defined;*

(iv) after the action, $(val_l^\delta(C) = val_l^\delta(D) \neq \perp) \wedge (val_{l-1}^\delta(C) \oplus k(C) = val_{l-1}^\delta(D) \oplus k(D))$ *when* $\delta = +$*, or* $(val_l^\delta(C) = val_l^\delta(D) \neq \perp) \wedge (val_{l+1}^\delta(C) \oplus k(C) = val_{l+1}^\delta(D) \oplus k(D))$ *when* $\delta = -$*;*

Proof. Briefly speaking, once statement (ii), (iii), and (iv) are fulfilled, then either $C \equiv D$, or **BadHit** happens. See [16] for the formal proof. □

If all the previous calls to $ForveVal$ were non-overwriting, then the calls to $ForceVal$ triggered by safe calls to *Adapt* do not affect the values in previously defined chains, nor the equivalence relation. As mentioned in Introduction, this property is one of the key points of the proof, and is similar to those exhibited in [17] and [18]; the difference is brought in by the increased buffer rounds.

Lemma 7. *Consider a safe call* $Adapt(x_{l-4}, x_{l-3}, x_{l-2}, x_{l+3}, x_{l+4}, x_{l+5}, l)$ *in a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*, and suppose all the previous calls to Adapt to be safe, then:*

(i) Right before the subsequent call to $F^{inner}(l-1, x_{l-1})$*,* $x_{l-1} \notin G_{l-1}$*; right before the subsequent call to* $F^{inner}(l+2, x_{l+2})$*,* $x_{l+2} \notin G_{l+2}$*;*
(ii) The subsequent calls to $ForceVal$ *are non-overwriting.*
(iii) If a chain C *is table-defined before this call to Adapt and is not equivalent to the chain which is being completed, then for any* $i \in \{1, \ldots, 21\}$*,* $val_i^+(C)$ *and* $val_i^-(C)$ *are invariant before and after both calls to* $ForceVal$*.*
(iv) The relation $\equiv$ *between table-defined chains is invariant before and after the subsequent calls to* $ForceVal$*.*

Proof. See the full version [16]. □

Lemma 8. *Consider a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*. Let* C *be a chain which is dequeued and to be adapted at position* l *s.t.* $C \notin CompletedSet$*. Then the subsequent call to Adapt is safe, if the following holds when* C *is dequeued:*

$$(((val_{l-2}^+(C) \notin G_{l-2}) \vee (val_{l-2}^+(C) \in G_{l-2} \wedge val_{l-1}^+(C) \notin G_{l-1}))$$
$$\wedge((val_{l+3}^-(C) \notin G_{l+3}) \vee (val_{l+3}^-(C) \in G_{l+3} \wedge val_{l+2}^-(C) \notin G_{l+2}))).$$

Proof. See the full version [16]. □

For the following discussions, we introduce a tuple set $KUDCS$[4], as the set of 5-tuples $(x_{10}, x_{11}, x_{12}, 10, 6)$ which is enqueued by a call to $F^{inner}(11, x_{11})$. The tuples in this set are special in the sense that before the call to F^{inner} which leads to they being enqueued, the partial chains correspond to them were *key-undefined*.

Then, the following two lemmas show that the assumptions of Lemma 8 hold in a good execution. Lemma 9 shows them to hold before the chains are enqueued, while Lemma 11 shows them to hold till the chains are dequeued (so that all calls to $ForceVal$ are non-overwriting). Lemma 10 is a helper lemma for Lemma 11.

Lemma 9. *Consider a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*. Let* C *be a partial chain which is enqueued at some time and to be adapted at position* l*. Suppose that no chain equivalent to* C *was enqueued before* C *is enqueued. Then:*

(i) $val_{l-2}^+(C) = \perp$ *and* $val_{l+3}^-(C) = \perp$ *before the call to* $F^{inner}(i, x)$ *which led to* C *being enqueued.*
(ii) right after C *is enqueued,* $val_{l-2}^+(C) \notin G_{l-2} \wedge val_{l+3}^-(C) \notin G_{l+3}$*.*

Proof. See the full version [16]. □

Lemma 10. *Consider a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*. Let* $C = (x_{10}, x_{11}, x_{12}, 10, 6) \in KUDCS$ *be a partial chain which is enqueued at some time such that no chain equivalent to* C *was enqueued before* C *is enqueued. Then for any chain* D *which is dequeued before* C *is dequeued, the following two hold;*

[4] The term is short for *key-undefined chain set*.

(i) it cannot be $val_4^+(C) \neq \bot \wedge val_4^+(C) = val_4^+(D) \wedge val_3^+(C) \oplus k(C) = val_3^+(D) \oplus k(D)$*;*
(ii) it cannot be $val_9^-(C) \neq \bot \wedge val_9^-(C) = val_9^-(D) \wedge val_{10}^-(C) \oplus k(C) = val_{10}^-(D) \oplus k(D)$*;*

Proof. By the assumption, D must have been enqueued before C is enqueued. Consider proposition (i). After the call to $F^{inner}(11, x_{11})$ which led to C being enqueued, we have:

(i) $val_4^+(C) = \bot$ (follows from Lemma 9);
(ii) C is table-defined, and D is equivalent to some table-defined chain D_{td}, since they have been enqueued (hence C and D_{td} are also key-defined).

After this point in the execution, since C has been table-define, $val_4^+(C)$ can only be changed to non-empty by the tape accessing and entry setting actions (by Lemma 7 (iii)) on $\{G_i\}$ (by Lemma 4 (ii)). Then proposition (i) is established by Lemma 6 (note that $val_i^\delta(D) = val_i^\delta(D_{td})$).

Consider proposition (ii). After the call to $F^{inner}(11, x_{11})$, we have:

(i) $val_9^-(C) \neq \bot \wedge val_9^-(C) \notin G_9$ (also follows from Lemma 9);
(ii) C and D_{td} ($D \equiv D_{td}$) are table-defined (and key-defined);

Depending on $val_9^-(D)$, we distinguish the following cases. First, if $val_9^-(D) \neq \bot$ before the call to $F^{inner}(11, x_{11})$, then D must have been enqueued before this call. By this, for some sufficiently large j, we have $(x'_{10}, x'_{11}, x'_{12}, 10) = prev^j(D)$ where all the three values have been in corresponding tables and $x'_{11} \neq x_{11}$. Then after the call, $val_9^-(C) = val_9^-(D)$ is not possible (and will never be possible in future) since it implies **BadHit**.

Second, if $val_9^-(D) = \bot$ before and after the call to $F^{inner}(11, x_{11})$, then similarly to the argument for proposition (i), $val_9^-(C) \neq \bot \wedge val_9^-(C) = val_9^-(D) \wedge val_{10}^-(C) \oplus k(C) = val_{10}^-(D) \oplus k(D)$ cannot be simultaneously fulfilled.

Finally, if $val_9^-(D) = \bot$ before the call to $F^{inner}(11, x_{11})$ while $val_9^-(D) \neq \bot$ after it, then the only possible case is $D = (x'_{10}, x_{11}, x'_{12}, 10)$ and D is also enqueued by the call to $F^{inner}(11, x_{11})$. In this case, assume that $val_9^-(C) \neq \bot \wedge val_9^-(C) = val_9^-(D) \wedge val_{10}^-(C) \oplus k(C) = val_{10}^-(D) \oplus k(D)$ simultaneously hold; then it necessarily be $x_{12} = x'_{12}$ and $G_{10}(x_{10}) \oplus x_{10} = G_{10}(x'_{10}) \oplus x'_{10}$. By construction, $G_{10}(x_{10})$ and $G_{10}(x'_{10})$ are defined to be $\varphi_{10}(x_{10})$ and $\varphi_{10}(x'_{10})$ respectively (since the 10-th round is not in the adaptation zone), hence the one defined later implies **BadHit**.

Having excluded all possibilities, we establish proposition (ii). □

Lemma 11. *In a good execution* $\mathbf{D}^{\Sigma'_1(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*, all calls to Adapt are safe, and all calls to* $ForceVal$ *are non-overwriting.*

Proof. Suppose that the lemma does not hold, and let C be the first chain during the completion of which the call to *Adapt* is not safe. Clearly $C \notin CompletedSet$ when C is dequeued, and since all calls to *Adapt* before C is dequeued were safe, by Lemma 5 (iii) we know when C was enqueued, no chain equivalent

to C had been enqueued. Hence, Lemma 9 implies that $val^+_{l-2}(C) \notin G_{l-2} \wedge val^+_{l+3}(C) \notin G_{l+3}$ immediately holds after C was enqueued. We show that when C is dequeued, $val^+_{l-1}(C) \notin G_{l-1} \wedge val^+_{l+2}(C) \notin G_{l+2}$; this implies that the subsequent call to *Adapt* is safe (by Lemma 8), so that the calls to $ForceVal$ are non-overwriting (by Lemma 7 (ii)). *Wlog* consider $val^+_{l-2}(C)$ and $val^+_{l-1}(C)$. If $val^+_{l-2}(C) = \perp$ after C was enqueued, we show that $val^+_{l-2}(C) = x_{l-2} \notin G_{l-2}$ immediately holds after $val^+_{l-2}(C) \neq \perp$ holds. Consider the last table entry setting action before $val^+_{l-2}(C) \neq \perp$ holds. Recall that C has been equivalent to a table-defined chain C_{td} since being enqueued; then by Lemma 7 (iii), $val^+_{l-2}(C) = val^+_{l-2}(C_{td})$ cannot be changed by previous calls to $ForceVal$. Hence it was changed by a tape accessing and entry setting action, and we have $val^+_{l-2}(C) = x_{l-2} \notin G_{l-2}$ after this action (Lemma 4 (ii)).

Now assume $val^+_{l-1}(C) \in G_{l-1}$ when C is dequeued. Then during the period between the point C was enqueued and the point C is dequeued, the following two actions must have been induced by the completion of some other chains D:

(i) $G_{l-2}(val^+_{l-2}(C))(= G_{l-2}(x_{l-2}))$ was defined;
(ii) after action (i), $G_{l-1}(val^+_{l-1}(C))$ was defined;

We show it to be impossible to show that $val^+_{l-1}(C) \notin G_{l-1}$ holds when C is dequeued. If the two happen, then for (either of) them two to be defined during the completion of D, we must have $val^+_{l-2}(D) = val^+_{l-2}(C)$ **or** $val^+_{l-1}(D) = val^+_{l-1}(C)$. We then show that for a chain D which is completed in this period,

- during the completion of D, if $val^+_{l-2}(C) = val^+_{l-2}(D)$, then $val^+_{l-1}(C) \neq val^+_{l-1}(D)$ (hence $G_{l-1}(val^+_{l-1}(C))$ cannot be defined).
- during the completion of D, $G_{l-1}(val^+_{l-1}(C))$ can be defined only if $val^+_{l-2}(C) = val^+_{l-2}(D)$ $(val^+_{l-1}(C) = val^+_{l-1}(D) \Rightarrow val^+_{l-2}(C) = val^+_{l-2}(D))$.

Gathering the two claims yields that $G_{l-1}(val^+_{l-1}(C))$ cannot be defined during this period and the call to *Adapt* will be safe.

For the first claim, assume otherwise, i.e. $val^+_{l-2}(D) = val^+_{l-2}(C)$, and right after $G_{l-2}(val^+_{l-2}(D))$ was defined, $val^+_{l-1}(D) = val^+_{l-1}(C)$ holds. This means that before $G_{l-2}(val^+_{l-2}(D))$ was defined, the following two hold:

(i) $val^+_{l-2}(D) = val^+_{l-2}(C) \neq \perp$
(ii) $val^+_{l-3}(D) \oplus k(D) = val^+_{l-3}(C) \oplus k(C)$

Consider the last table entry setting action before the above two hold. After this action, we have $val^+_{l-2}(D) \neq \perp$ and $val^+_{l-2}(C) \neq \perp$; then after this action, C must have been enqueued (because by Lemma 9 (i), before C was enqueued, $val^+_{l-2}(C)$ should be $\perp$), and D has been enqueued even earlier, hence C and D are equivalent to some table-defined chains C_{td} and D_{td} respectively. Then, if $C \in KUDCS$, a contradiction is directly reached by Lemma 10; if $C \notin KUDCS$, for the action, we exclude possibility for each case:

(i) This cannot have been a tape accessing and table entry setting action on $\{G_i\}$. To illustrate this, assume otherwise. Then this action must be the one or posterior to the one which leads to C being enqueued, and the following five hold simultaneously, which contradicts Lemma 6:
  - before the action, both C_{td} and D_{td} are key-defined.
  - before the action, C_{td} is not equivalent to D_{td};
  - before the action, $val^+_{l-2}(C_{td}) = \perp$ or $val^+_{l-2}(D_{td}) = \perp$;
  - after the action, C_{td} and D_{td} are table-defined;
  - after the action, $val^+_{l-2}(D_{td}) = val^+_{l-2}(C_{td}) \neq \perp$ and $val^+_{l-3}(D_{td}) + k(D_{td}) = val^+_{l-3}(C_{td}) + k(C_{td})$;

(ii) This cannot have been an entry setting action on E, since such actions cannot change $val^+_{l-2}(C_{td})$ nor $val^+_{l-2}(D_{td})$ (by Lemma 4 (ii));

(iii) This cannot have been because of a previous call to $ForceVal$. For this, assume otherwise; as already discussed before, after this call to $ForceVal$, C and D are enqueued and equivalent to some table-defined chains C_{td} and D_{td} respectively. Then it must be either of the following two cases:
  (a) C has been enqueued before this call to $ForceVal$. Then by Lemma 7 (iii), none of the previous calls to $ForceVal$ affects $val^+_i(D) = val^+_i(D_{td})$ and $val^+_i(C) = val^+_i(C_{td})$, a contradiction.
  (b) C is enqueued by this call to $ForceVal$. This is impossible.

Hence the first claim holds.

For the second claim, assume otherwise, then we know that before the entry setting action on $G_{l-1}(val^+_{l-1}(C))$, the following two hold:

(i) $val^+_{l-2}(C) \in G_{l-2}$, $val^+_{l-2}(D) \in G_{l-2}$, and $val^+_{l-2}(C) \neq val^+_{l-2}(D)$
(ii) $val^+_{l-1}(C) = val^+_{l-1}(D) \notin G_{l-1}$

Consider the last table entry setting action before the above two hold. By Lemma 9 (ii), $val^+_{l-2}(C) \notin G_{l-2}$ immediately holds after C is enqueued; hence this action must happen after C is enqueued, and C, D (enqueued earlier that C) must have been equivalent to some table-defined chains C_{td} and D_{td} respectively, as discussed before. Then, since none of the previous calls to $ForceVal$ affects $val^+_i(D) = val^+_i(D_{td})$ and $val^+_i(C) = val^+_i(C_{td})$ (by Lemma 7 (iii)), the last action before the above two hold must be a tape accessing and entry setting action. Moreover, since $val^+_{l-2}(C_{td}) \notin G_{l-2}$ and C_{td} is table-defined (and $val^+_{l-2}(D_{td}) \notin G_{l-2}$ and D_{td} is table-defined) immediately hold after C (D, resp.) is enqueued, and then this action changed $val^+_{l-1}(C_{td})(= val^+_{l-1}(C))$ and $val^+_{l-1}(D_{td})(= val^+_{l-1}(D))$ from $\perp$ to non-empty values, this action must have been a defining action on either $G_{l-2}(val^+_{l-2}(C_{td}))$ or $G_{l-2}(val^+_{l-2}(D_{td}))$ (by Lemma 4 (ii)). However neither is possible: *wlog* assume the action to be $G_{l-2}(val^+_{l-2}(C_{td})) := \varphi_{l-2}(val^+_{l-2}(C_{td}))$, then after this action, the following holds (by $val^+_{l-1}(C_{td}) = val^+_{l-1}(D_{td}) \notin G_{l-1}$):

$$\begin{aligned} &val^+_{l-3}(C_{td}) \oplus \varphi_{l-2}(val^+_{l-2}(C_{td})) \oplus k(C_{td}) \\ =&val^+_{l-3}(D_{td}) \oplus G_{l-2}(val^+_{l-2}(D_{td})) \oplus k(D_{td}) \end{aligned}$$

Suppose $C_{td} = (c_i, c_{i+1}, c_{i+2}, i)$ and $D_{td} = (d_j, d_{j+1}, d_{j+2}, j)$, then we have

$$\varphi_{l-2}(val^+_{l-2}(C_{td})) = val^+_{l-3}(C_{td}) \oplus c_i \oplus G_{i+1}(c_{i+1}) \oplus c_{i+2} \\ \oplus val^+_{l-3}(D_{td}) \oplus G_{l-2}(val^+_{l-2}(D_{td})) \oplus d_j \oplus G_{j+1}(d_{j+1}) \oplus d_{j+2}$$

which implies an occurrence of **BadHit**. Therefore the claim that $G_{l-1}(val^+_{l-1}(C))$ $(= G_{l-1}(val^+_{l-1}(C_{td})))$ can be defined only if $val^+_{l-2}(C) = val^+_{l-2}(D)$ holds.

Having excluded all possibilities we show $val^+_{l-1}(C) \notin G_{l-1}$ to hold when C is dequeued. The reasoning for $val^+_{l+1}(C) \notin G_{l+1}$ is similar by symmetry. Hence the subsequent call to *Adapt* will be safe; and by Lemma 7 (ii), the subsequent calls to $ForceVal$ will be non-overwriting. □