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
[2] University of Chinese Academy of Sciences
{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 construct hash functions and message authentication codes. These applications require the security in the *open key model*, where the adversary

* A preliminary version is to appear at TCC 2015.

knows 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. It can be used to evaluate the "closeness" of a block cipher construction to an *ideal cipher*[3]. Despite the uninstantiability of idealized primitives [12,25,9], 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 [22], Feistel constructions have been extensively studied. Most of the provable security works fall in the Luby-Rackoff framework [22], 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. [26,24,27]), provable security in the open key model (Mandal *et al.* [23] 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 [31], 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.* [21]. The KAF cipher they considered has the round keys xored before each round function, as depicted in Fig. 1 (right). Lampe *et al.* studied the indistinguishability of KAF in a setting where the underlying round functions are random and public (in contrast to the Luby-Rackoff setting) and the round 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

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

discussions are deferred to the full version). 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 assumptions 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 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 the first to study how to properly mix the keys into Feistel in the open key model. It 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.

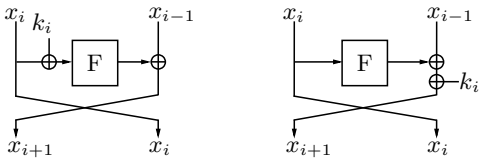

Fig. 1. Mixing the key into: (left) the input of the round function – KAF; (right) the halve of the state after the round function – KAF^*.

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[4]. 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 attacked in hash mode (the example is Camellia, in [7]). Hence we think our results have its own advantage. 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 [4,8,1,32,11]. 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.* [20] (see the next paragraph). We stress that our main constructions consist of the indifferentiability result for KAF^* and the analyzes of KAF^*.

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.* [20]. 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 path 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 path 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 of the ideal cipher $\mathbf{E}$ which has n-bit keys and $2n$-bit blocks; therefore 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 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.

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

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 an 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.[5]

Another evidence for the necessity of increasing the number of buffer rounds is that they actually play an important role in the proof (see Lemma 17).

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 the formal definitions of the ideal cipher model and the indifferentiability notion. Sect. 3 makes discussions on KAF. Sect. 4 contains the main theorem. Sect. 5 presents the simulator. Finally, Sect. 6 gives 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 $\mathbf{E} : \{0,1\}^n \times \{0,1\}^\kappa \to \{0,1\}^n$, which is taken randomly from the set of $(2^n!)^{2^k}$ 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]).

[5] An alternative interpretation of this *dual-buffer strategy* is that we use an *outer buffer* and an *inner buffer*, and we expect the outer buffer to protect the inner buffer, and the latter further protect the adaptation zone.

The Indifferentiability Framework The indifferentiability framework was introduced by Maurer, Renner, and Holenstein, at TCC 2004 [25]. 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[6] . 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 [2,20].

3 KAF Ciphers, and Obstacles in Its Indifferentiability Proof

Consider the Feistel ciphers of which the pseudorandom round functions are of the form $F_i(x \oplus k_i)$, where k_i is the round key and F_i is a random function. More clearly, at the i-th round, the state is updated according to

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

where x_i and x_{i-1} are respectively the left and right n-bit halves of the state, and k_i is an n-bit round key (Fig. 1 (left)). This construction was named *key-alternating Feistel cipher* by Lampe and Seurin, since it can also be seen as a key-alternating cipher with two-round Feistel-based permutations. Throughout this paper, we refer to this construction by KAF. KAF seems to be the "most natural" solution to the problem *how to mix key into Feistel*. As to existing designs, many notable block ciphers can be seen as the variant of KAF:

- DES: although it has an expansion function before the key xoring;
- GOST: although it inserts keys via modular addition instead of xor;

Possibly because it is actually used in block cipher constructions and it is "natural", KAF has received considerable attention. Previous works focus on known-key attacks [19,30,29], complementing (related-key) attacks [7], and generic attacks [18,33,16], and the provable security of KAF in the single secret key setting [21].

[6] Restrictions on the indifferentiability composition theorem have been exhibited in [28,15]. However, indifferentiability has been sufficient in most "natural" settings (see [15]).

Although works well in the single secret key setting, there exist obstacles when trying to achieve an indifferentiability proof for KAF: first, due to the complementation property (e.g. DES)

$$E(\overline{x}, \overline{k}) = \overline{E(x, k)},$$

KAF ciphers without key derivation functions are vulnerable to related-key distinguishing attack, thus not indifferentiable from ideal ciphers.

Second, when appealing to key derivation functions modeled as random oracles, there still exist obstacles. According to the works of Biryukov *et al.* [7], KAF with as many as two *random-oracle-derived* keys alternatively applied at each round is still vulnerable to a *relaxed complementing attack*: the attacker asks the key derivation oracle for two arbitrary keys $(k_1^1, k_2^1) := KD(K^1)$ and $(k_1^2, k_2^2) := KD(K^2)$ and let $\delta_1 = k_1^1 \oplus k_1^2$, $\delta_2 = k_2^1 \oplus k_2^2$, then for plaintext pairs of the form $((x_0, x_1), (x_0 \oplus \delta_2, x_1 \oplus \delta_1))$, with probability 1 we have $\mathrm{KAF}_r((x_0, x_1), K^1) \oplus \mathrm{KAF}_r((x_0 \oplus \delta_2, x_1 \oplus \delta_1), K^2) = (\delta_1, \delta_2)$ when the number of rounds r is odd ((δ_2, δ_1) when r is even).

We *conjecture* that indifferentiability may be achieved on KAF with key derivation of form $(k_1, k_2, k_3) = KD(K)$ and the three keys cyclicly applied at each round; or, on KAF with key derivation of form $(k_1, k_2) = KD(K)$ and the two keys applied in the order k_1, k_2, k_2, k_1 repeatedly. Whereas: first, the proof still seems difficult; second, such results impose strong constraints on the key derivation; third, such a key derivation model seems to be intricate, unnatural, and unreasonable. All the discussions above motivate us to turn to KAF^*.

4 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*

$$\sigma = 2^{11} \cdot q^9 \text{ and } \varepsilon \leq \frac{2^{19} \cdot q^{15}}{2^{2n}} + \frac{2^{222} \cdot q^{30}}{2^n} + \frac{2^{34} \cdot q^6}{2^{2n}} = O(\frac{q^{30}}{2^n}).$$

To prove Theorem 1 we firstly describe a simulator $\mathbf{S}$ which mimics the behaviors of the random functions $\mathbf{F}$, then bound the complexity of $\mathbf{S}$ and prove the indistinguishability of the simulated world $\Sigma_1(\mathbf{E}, \mathbf{S})$ and the real world $\Sigma_2(SKAF_{21}^*, \mathbf{F})$.

5 The Simulator

We first provide a high-level description of $\mathbf{S}$, then present the pseudocode to illustrate it more clearly.

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 $(\delta, k, z) \in \{+,-\} \times \{0,1\}^n \times \{0,1\}^{2n}$ 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; for completeness we provide implementations for them, in Sect. 5.2. As argued by Andreeva *et al.* [2], such a strategy does not reduce the validity of the simulating, since access to such tapes can be efficiently simulated by uniformly sampling.

5.1 High-level Description of 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 path 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.

5.2 Formal Description of the Simulator

A formal description of the simulator $\mathbf{S}(\varphi)$ in pseudocode is presented as follows. A slightly different simulator $\widetilde{S}(\varphi)$ will be introduced later (see Sect. 6.1). For this, 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

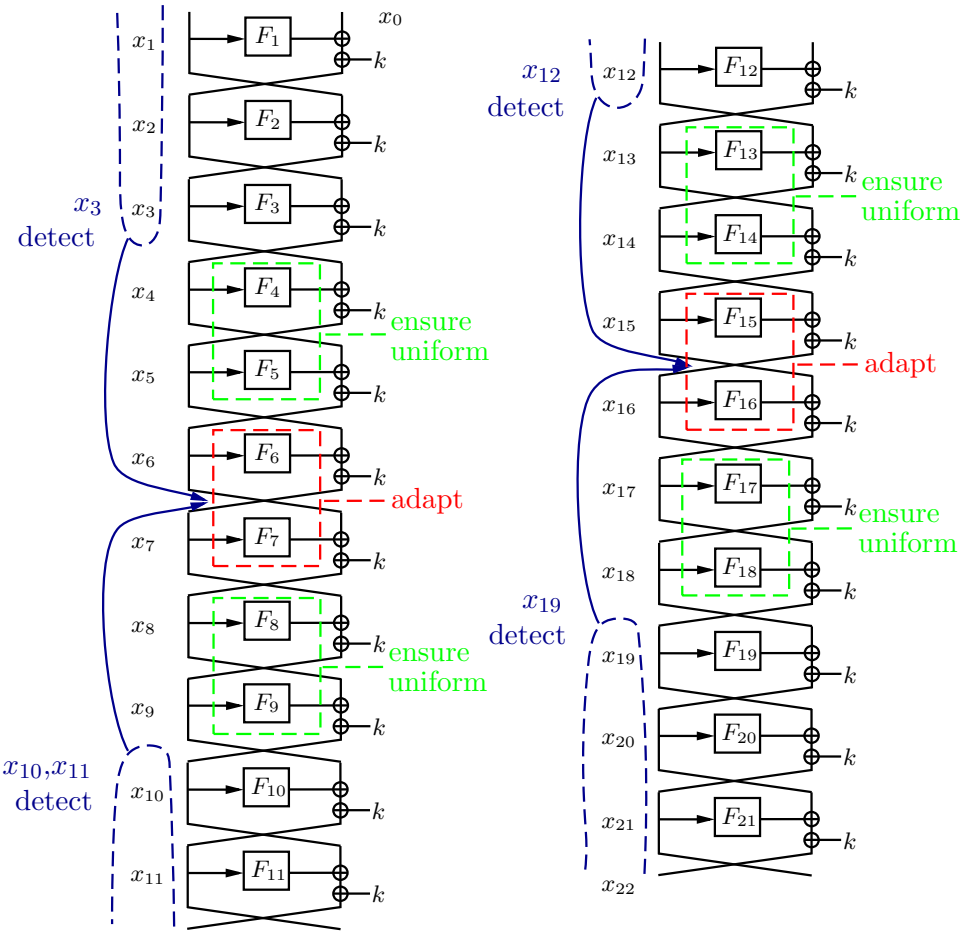

Fig. 2. The 21-round $SKAF^*$ cipher with the zones where the simulator detects chains and adapts them.

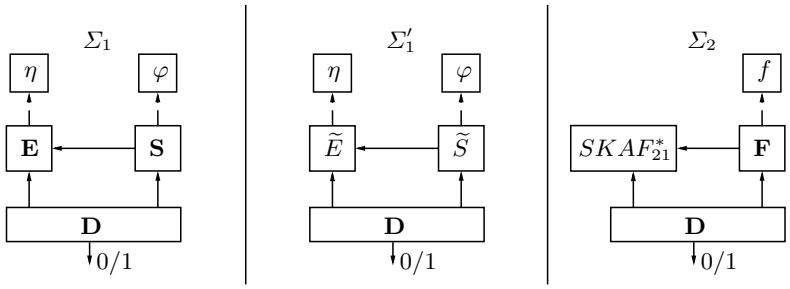

Fig. 3. Systems used in the proof in this paper

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* adapts the values by randomly setting the necessary but "missed" entries and then adding 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: $y_{l-1} := F^{inner}(l-1, x_{l-1})$
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** $G_i(x)$

The procedure *EnqueueNewChains* enqueues newly generated partial chains.

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 **E** to verify whether the inputs are valid pairs of plaintext and ciphertext of **E**. Note that $\widetilde{S}$ does not own *Check* procedure; instead $\widetilde{S}$ calls the *Check* procedure of a modified cipher $\widetilde{E}$, as described in the boxed statements in the code section before.

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** $(x_l, x_{l+1}, x_{l+2}, l)$

As mentioned before, $\mathbf{E}(\eta)$ and $\mathbf{F}(f)$ simply relay the values in η and f; but for completeness, we provide the codes of them.

1: **Ideal cipher** $\mathbf{E}(\eta)$:
2: **public procedure** $Enc(k, x)$
3: **return** $\eta(+, k, x)$
4: **public procedure** $Dec(k, x)$
5: **return** $\eta(-, k, x)$

1: **Random functions** $\mathbf{F}(f)$:
2: **public procedure** $F(i, x)$
3: **return** $f_i(x)$

6 Proof of the Indifferentiability

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.

6.1 Intermediate System Σ_1', and Proof Sketch

For further simplicity, we introduce 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 to keep track of the past queries, which contains entries of the form $((+, k, x), y)$ and $((-, k, y), x)$. $\widetilde{E}(\eta)$ 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 of 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. More clearly, $\widetilde{E}(\eta)$ is implemented as follows:

1: **Modified ideal cipher** $\widetilde{E}(\eta)$:
2: **Variables**
3: hash table E, initially empty
4: **end variables**
5: **public procedure** $Enc(k, x)$
6: **if** $(+, k, x) \notin E$ **then**
7: $y := \eta(+, k, x)$
8: $E(+, k, x) := y$
9: $E(-, k, y) := x$
10: **end if**
11: **return** $E(+, k, x)$
12: **end procedure**

13: **public procedure** $Dec(k, y)$
14: **if** $(-, k, y) \notin E$ **then**
15: $x := \eta(-, k, y)$
16: $E(-, k, y) := x$
17: $E(+, k, x) := y$
18: **end if**
19: **return** $E(-, k, y)$
20: **end procedure**
21: **public procedure** $Check(x, y, k)$
22: **if** $(+, k, x) \in E$ **then**
23: **return** $E(+, k, x) = y$
24: **else**
25: **return false**
26: **end if**
27: **end procedure**

By construction and the fact that η encodes an ideal cipher, it can be easily seen that $\widetilde{E}(\eta).E$ always defines a partial cipher before or after any call to Enc or Dec, i.e. for all $k, x, y \in \{0,1\}^n$, $(+, k, x) \in \widetilde{E}(\eta).E$ and $\widetilde{E}(\eta).E(+, k, x) = y$ iff. $(-, k, y) \in \widetilde{E}(\eta).E$ and $\widetilde{E}(\eta).E(-, k, y) = x$. We use $|\widetilde{E}(\eta).E^+|$ and $|\widetilde{E}(\eta).E^-|$ to denote the number of entries in $\widetilde{E}(\eta).E$ of the form $((+, \cdot, \cdot), \cdot)$ and $((-, \cdot, \cdot), \cdot)$ respectively. By above, $|\widetilde{E}(\eta).E| = 2 \cdot |\widetilde{E}(\eta).E^+| = 2 \cdot |\widetilde{E}(\eta).E^-|$ holds at any point when every call to Enc or Dec has been answered.

On the other hand, the differences between $\widetilde{S}(\varphi)$ and $\mathbf{S}(\varphi)$ (in Σ_1) are captured by the boxed statements in the pseudocode in Sect. 5.2. They mainly consist of two aspects:

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

The three systems are depicted in Fig. 3. For simplicity we introduce additional notations $\Sigma_1(\eta, \varphi)$, $\Sigma_1'(\eta, \varphi)$, and $\Sigma_2(f)$, among which $\Sigma_1(\eta, \varphi)$ is short for $\Sigma_1(\mathbf{E}(\eta), \mathbf{S}(\varphi))$, $\Sigma_1'(\eta, \varphi)$ is short for $\Sigma_1^{'}(\widetilde{E}(\eta), \widetilde{S}(\varphi))$, and $\Sigma_2(f)$ is short for $\Sigma_2(SKAF_{21}^*, \mathbf{F}(f))$. Furthermore, for $\alpha = (\eta, \varphi)$, $\Sigma_1'(\alpha)$ is short for $\Sigma_1'(\eta, \varphi)$.

Then the proof of indifferentiability is divided into two stages:

- First, we specify bad events in the $Check$ procedure in Σ_1, which capture the essential differences between Σ_1 and $\Sigma_1^{'}$. Then for any distinguisher which issues at most q queries, we upper bound the probability of the bad events to $\frac{2^{19} \cdot q^{15}}{2^{2n}}$ (in the proof of Lemma 5); and upper bound the advantage of distinguishing Σ_1 and $\Sigma_1^{'}$ to $\frac{2^{19} \cdot q^{15}}{2^{2n}}$ (Lemma 5) and the complexity of $\mathbf{S}$ in Σ_1 to no more than $2 \cdot (10q^3)^3 \leq 2^{11} \cdot q^9$ queries (Lemma 6).
- Second, we specify bad events relevant to $ForceVal$ procedure overwriting entries, bound the probability of such events, and finally use a *relaxed randomness mapping argument* to upper bound the advantage of distinguishing $\Sigma_1^{'}$ and Σ_2 to $\frac{2^{2\hat{2}2} \cdot q^{30}}{2^n} + \frac{2^{34} \cdot q^6}{2^{2n}}$ (Lemma 25).

Gathering these yields Theorem 1.

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

In this section we show that the simulator $\widetilde{S}(\varphi)$ in Σ_1' runs in polynomial time. The technique used in this section (and Sect. 6.3) is originally introduced by Coron *et al.* [14] (also used by Holenstein *et al.* [17] and Lampe *et al.* [20]).

Lemma 1. *During any execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*, after the* q*-th query made by* $\mathbf{D}$ *is answered,* $\widetilde{S}(\varphi)$ *dequeues at most* q *times a tuple of the form* $(x_1, x_2, x_3, 1, l)$ *for which* $(x_1, x_2, x_3, 1) \notin \mathit{CompletedSet}$*.*

Proof. We will show that each such dequeuing action corresponds to a distinct call to $\widetilde{E}(\eta).\mathit{Enc}(k, x)$ or $\widetilde{E}(\eta).\mathit{Dec}(k, y)$ previously made by $\mathbf{D}$.

Consider such a dequeuing action and let $(x_1, x_2, x_3, 1, l)$ be the tuple dequeued for which $(x_1, x_2, x_3, 1) \notin \mathit{CompletedSet}$. By construction, $(x_1, x_2, x_3, 1, l)$ can be enqueued only when $(x_1, x_2, x_3, x_{20}, x_{21})$ or $(x_1, x_2, x_{19}, x_{20}, x_{21})$ is detected and the call $\widetilde{E}(\eta).\mathit{Check}((x_1, x_0), (x_{22}, x_{21}), k)$ returns true, and the latter happens only when the entry $((+, k, (x_1, x_0)), (x_{22}, x_{21}))$ has been in $\widetilde{E}(\eta).E$. Hence each such dequeuing corresponds to an entry in $\widetilde{E}(\eta).E$.

On the other hand, if two chains $C = (x_1, x_2, x_3, 1)$ and $C' = (x_1', x_2', x_3', 1)$ dequeued correspond to the same call to $\widetilde{E}(\eta).\mathit{Check}$, then we must have $x_i = x_i'$ for $i = 1, 2, 3$, and $C = C'$. In this case, C will be added to $\mathit{CompletedSet}$ since its first completion, and it will not be $C \notin \mathit{CompletedSet}$ when C is dequeued again. Hence each such dequeuing corresponds to a unique entry $((+, k, x), y)$ in $\widetilde{E}(\eta).E$. This entry must have been added during a query issued by $\mathbf{D}$, since $\widetilde{S}(\varphi)$ makes such queries only when it is completing a chain, and after this completion, the chain $(x_1, x_2, x_3, 1)$ will be added into $\mathit{CompletedSet}$, and it cannot be $(x_1, x_2, x_3, 1) \notin \mathit{CompletedSet}$ when it is dequeued again.

Hence, each such dequeuing action corresponds to a call to $\widetilde{E}(\eta).\mathit{Enc}$ or $\widetilde{E}(\eta).\mathit{Dec}$ made by $\mathbf{D}$, and cannot occur more than q times. □

Lemma 2. *During any execution* $\mathbf{D}^{\Sigma_1(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*, after the* q*-th query made by* $\mathbf{D}$ *is answered:*

- *for* $i \in \{1, 2, \ldots, 21\}$ *and* $\delta \in \{+, -\}$, $|G_i| \leq 10q^3$*, and* $|\widetilde{E}.E^{\delta}| \leq 10q^3$*;*
- $\widetilde{S}$ *issues at most* $2 \cdot (10q^3)^5$ *queries to* $\widetilde{E}.\mathit{Check}$*;*

Proof. The proof is similar to Lemma 2 in [20], while the results are different.

First, $|G_{10}|$, $|G_{11}|$, and $|G_{12}|$ are at most $2q$: entries can only be added to these three tables when $\mathbf{D}$ issues a query to $F(i, x)$ with $i = 10$, 11, or 12, or when $\widetilde{S}$ completes a chain $(x_1, x_2, x_3, 1, l)$. The former occurs at most q times, while the latter occurs at most q times by Lemma 1, hence the bound is $2q$, and $\widetilde{S}$ completes at most $|G_{10}| \cdot |G_{11}| \cdot |G_{12}| \leq 8q^3$ chains of form $(x_{10}, x_{11}, x_{12}, 10, l)$.

Second, for any $i \in \{1, \ldots, 21\}$, $|G_i|$ can only be enlarged by at most 1 when: the distinguisher calls $\widetilde{S}.F(i, x)$; a chain $(x_1, x_2, x_3, 1, l)$ is completed; or a chain $(x_{10}, x_{11}, x_{12}, 10, l)$ is completed. The first case occurs at most q times,

the second at most q times, while the last at most $8q^3$ times. Hence in total the bound is $2q + 8q^3 \leq 10q^3$.

Then, by construction, each query to either $\widetilde{E}.Enc$ or $\widetilde{E}.Dec$ increases both $|\widetilde{E}.E^+|$ and $|\widetilde{E}.E^-|$ by at most 1. Such queries may be issued by $\mathbf{D}$ or $\widetilde{S}$. The former is at most q, while the latter only happens during completion of a chain, thus at most $q + 8q^3$. Hence the bound is $q + q + (8q^3) \leq 10q^3$. Finally, the number of queries to $\widetilde{E}.Check$ made by $\widetilde{S}(\varphi)$ is bounded by $|G_1| \cdot |G_2| \cdot |G_3| \cdot |G_{20}| \cdot |G_{21}| + |G_1| \cdot |G_2| \cdot |G_{19}| \cdot |G_{20}| \cdot |G_{21}| \leq 2 \cdot (10q^3)^5$. □

6.3 Indistinguishability of Σ_1 and Σ_1'

The proof of indistinguishability of Σ_1 and Σ_1' is presented in this section. It consists of specifying the bad events in Σ_1, showing the two systems to have exactly same behaviors given that the bad events do not happen, and upper bounding the complexity of $\mathbf{S}$ in Σ_1.

Bad Event BadCheck Since we have made the randomness taken by Σ_1 explicit, the only essential difference between Σ_1 and Σ_1' lies in the $Check$ procedure: in Σ_1', the return value of a call $\widetilde{E}(\eta).Check(x, y, k)$ depends on the content of the table $\widetilde{E}(\eta).E$, while in Σ_1 the return value of $\mathbf{S}(\varphi).Check(x, y, k)$ actually depends on a much larger table η. For this, we define a bad event **BadCheck**: consider a pair of random tapes (η, φ), **BadCheck** happens during the execution $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$ if $\exists(x, y, k)$ s.t. all the following hold:

(i) $\mathbf{S}(\varphi)$ makes a call $Check(x, y, k)$;
(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.

Note that such a call $Check(x, y, k)$ returns true if being made in $\Sigma_1(\mathbf{E}(\eta), \mathbf{S}(\varphi))$, while returns false if being made in $\Sigma_1'(\widetilde{E}(\eta), \widetilde{S}(\varphi))$; this is the main idea of **BadCheck**. We now bound the probability that **BadCheck** happens in a *fixed number* of calls to $\mathbf{S}(\varphi).Check$.

Lemma 3. *Fix a point in* $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$. *Suppose up to this point,* $\mathbf{S}(\varphi).Check$ *is called* q_1' *times, while* $\mathbf{E}(\eta).Enc$ *and* $\mathbf{E}(\eta).Dec$ *are queried* q_2' *times in total. Then the probability that* **BadCheck** *happens before this point is upper bounded to* $\frac{2q_1'}{2^{2n}}$.

Proof. Consider a call $\mathbf{S}(\varphi).Check(x, y, k)$. Since neither $\mathbf{E}(\eta).Enc(k, x)$ nor $\mathbf{E}(\eta).Dec(k, y)$ has been issued before this call, y is a $2n$-bit "fresh" value. Since η encodes a random permutation for each k, we have $Pr[\eta(+, k, x) = y] \leq \frac{1}{2^{2n} - q_2'}$. And since the number of queries to $Check$ is at most q_1', in total the probability is $\frac{q_1'}{2^{2n} - q_2'}$. Assuming $q_2' < \frac{2^{2n}}{2}$, we have $Pr[\mathbf{BadCheck}] \leq \frac{2 \cdot q_1'}{2^{2n}}$. □

Σ_1 with No BadCheck in the First $2 \cdot (10q^3)^5$ Calls to *Check*: Indistinguishable from Σ_1' Consider $\Sigma_1(\mathbf{E}(\eta), \mathbf{S}(\varphi))$. Instead of the *Check* procedure in $\mathbf{S}(\varphi)$, we can imagine that $\mathbf{E}(\eta)$ has a procedure which is implemented exactly as $\mathbf{S}(\varphi)$, and at line 44 and line 51, $\mathbf{S}(\varphi)$ calls $\mathbf{E}(\eta).Check$ instead of $\mathbf{S}(\varphi).Check$. This "imagined" $\mathbf{S}_{img}(\varphi)$ (can be obtained by excluding the implementation of *Check* from $\mathbf{S}(\varphi)$ and letting it call $\mathbf{E}(\eta).Check$) and $\widetilde{S}(\varphi)$ are the same except that they query different ciphers. By this, for the two systems $(\mathbf{D}, \mathbf{S}_{img}(\varphi))$ and $(\mathbf{D}, \widetilde{S}(\varphi))$, if all the return values of the procedures *Enc*, *Dec*, F, and *Check* equal correspondingly, then they two will have the same behaviors. To formally described these, we use the notion *transcript*. For a Σ_1-execution $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$, the *transcript* of $\mathbf{D} \cup \mathbf{S}$ is a sequence composed of all the following query answer pairs generated during the execution:

(i) all the *Dec*, *Check*, and φ queries issued by $\mathbf{D}$ and $\mathbf{S}(\varphi)$, and the corresponding answers (the φ queries are issued to the tape φ);
(ii) all the *Enc* queries issued **outside** the *Check* procedure by $\mathbf{D}$ and $\mathbf{S}(\varphi)$, and the corresponding answers;

On the other hand, for a Σ_1'-execution $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, the *transcript* of $\mathbf{D} \cup \widetilde{S}$ is a sequence composed of all the queries to *Enc*, *Dec*, *Check*, and φ generated by $\mathbf{D}$ and $\widetilde{S}(\varphi)$, and the corresponding answers.

Then, the following lemma claims that the transcripts of $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$ and $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ are equal if in $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$, **BadCheck** does not happen in a sufficiently long period. This lemma is the core of this section.

Lemma 4. *Consider two executions $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$ and $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$. Assume that $\widetilde{E}(\eta).Check$ receives N calls during $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$. Then if* **BadCheck** *does not happen in the first N calls to $\mathbf{S}(\varphi).Check$ during $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$, the transcript of $\mathbf{D} \cup \mathbf{S}$ in $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$ is the same as the transcript of $\mathbf{D} \cup \widetilde{S}$ in $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, and $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))} = \mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$.*

Proof. We proceed by induction on the sequence of queries of $\mathbf{D} \cup \mathbf{S}$ ($\mathbf{D} \cup \widetilde{S}$, resp.) to $(\mathbf{E}(\eta), \varphi)$ ($(\widetilde{E}(\eta), \varphi)$, resp.). Assume that the sequence of queries and answers is the same in the two systems up to some point in the executions, and consider the next query issued by $\mathbf{D} \cup \mathbf{S}$ or $\mathbf{D} \cup \widetilde{S}$. According to the discussions before (about $\widetilde{S}$ and $\mathbf{S}_{img}$), this query is the same in both systems. We now argue the corresponding answers are the same. For this, we distinguish the following cases depending on the query:

- if the query is to *Enc*, *Dec*, or φ, then the answers are clearly the same since the same tape tuple (η, φ) is used in the two systems.
- if the query is to *Check*, then by the assumptions that $\widetilde{E}(\eta).Check$ receives N calls during $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ and **BadCheck** does not happen in the first N calls to $\mathbf{S}(\varphi).Check$ during $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$, the answers are the same.

These show that the transcripts in the two systems are the same. Moreover, these also show that the transcripts of queries and answers of $\mathbf{D}$ are the same

in both the systems. Since $\mathbf{D}$ is deterministic, $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))} = \mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ holds. □

We are now ready to bound the advantage of distinguishing Σ_1 and Σ_1':

Lemma 5. *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. Consider a pair (η, φ). Assume that during $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, $\widetilde{E}(\eta).Check$ is called q_1' times. Then by Lemma 3, during $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$, the probability that **BadCheck** occurs in the first q_1' calls to $\mathbf{S}(\varphi).Check$ is at most $\frac{2q_1'}{2^{2n}}$. By Lemma 2, $q_1' \leq 2 \cdot (10q^3)^5$, hence

$$Pr_{(\eta,\varphi)}[\textbf{BadCheck } occurs\ during\ \mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}] \leq \frac{2 \cdot 2 \cdot (10q^3)^5}{2^{2n}} \leq \frac{2^{19} \cdot q^{15}}{2^{2n}}$$

and by Lemma 4 we know if **BadCheck** does not happen in all these calls, $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))} = \mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$. Hence

$$\begin{aligned}
&\left|Pr[\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))} = 1] - Pr[\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))} = 1]\right| \\
\leq & Pr_{(\eta,\varphi)}[\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))} \neq \mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}] \\
\leq & Pr_{(\eta,\varphi)}[\textbf{BadCheck } occurs\ during\ \mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}] \leq \frac{2^{19} \cdot q^{15}}{2^{2n}}
\end{aligned}$$

as claimed. □

Bounding the Complexity of S The discussions above enable us to upper bound the complexity of $\mathbf{S}$ in $\Sigma_1(\mathbf{E}(\eta), \mathbf{S}(\varphi))$ for most cases.

Lemma 6. *For any distinguisher* $\mathbf{D}$ *which issues no more than* q *queries, with probability no less than* $1 - \frac{2^{19} \cdot q^{15}}{2^{2n}}$, $\mathbf{S}(\varphi)$ *issues no more than* $2^{11} \cdot q^9$ *queries to* $\mathbf{E}(\eta)$ *during execution* $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$.

Proof. $\mathbf{S}(\varphi)$ issues at most $|G_1| \cdot |G_2| \cdot |G_3| + |G_{19}| \cdot |G_{20}| \cdot |G_{21}|$ queries to $\mathbf{E}(\eta)$. By Lemma 4, with probability at least $1 - \frac{2^{19} \cdot q^{15}}{2^{2n}}$, the transcript of $\mathbf{D} \cup \mathbf{S}$ in $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$ is the same as the transcript of $\mathbf{D} \cup \widetilde{S}$ in $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$; therefore, with probability at least $1 - \frac{2^{19} \cdot q^{15}}{2^{2n}}$, the bounds on the size of the tables (Lemma 2) holds in $\mathbf{D}^{\Sigma_1(\mathbf{E}(\eta),\mathbf{S}(\varphi))}$. Therefore the bound is $2 \cdot (10q^3)^3 \leq 2^{11} \cdot q^9$. □

6.4 Indistinguishability of Σ_1' and Σ_2

In this section we use a relaxed randomness mapping argument to upper bound the advantage of distinguishing Σ_1' and Σ_2. We recall the principle first.

The Relaxed Randomness Mapping Argument: Principle Since $\mathbf{D}$ is deterministic, each tuple of random tapes (η, φ) uniquely determines a $\Sigma_1^{'}$-execution. However during $\mathbf{D}^{\Sigma_1^{'}(\eta,\varphi)}$, certain entries in (η, φ) may not be accessed and will not affect the execution. The entries of (η, φ) that are accessed during $\mathbf{D}^{\Sigma_1^{'}(\eta,\varphi)}$ compose *footprint* (see Sect. 6.4 for a more formal definition). Hence there is a bijection between the possible value of the footprint and the transcript of $\mathbf{D}^{\Sigma_1^{'}(\eta,\varphi)}$. Then the core idea of upper bounding $|Pr[\mathbf{D}^{\Sigma_2} = 1] - Pr[\mathbf{D}^{\Sigma_1^{'}} = 1]|$ by relaxed randomness mapping argument is exhibiting a bijection τ between some of the footprints of (η, φ) and f such that (i) τ maps $\Sigma_1^{'}$ executions to Σ_2 executions that look exactly same in the view of $\mathbf{D}$; (ii) τ maps $\Sigma_1^{'}$ executions to Σ_2 executions of nearly equal probability; (iii) the domain of τ represents most of the probability mass of all the possible footprints of $\mathbf{D}^{\Sigma_1^{'}(\eta,\varphi)}$.

Our novelties In the previous proof of Andreeva *et al.* [2], the map linked the intermediate system G_2 and G_3, where G_2 was composed of the target idealized primitive and the simulator, and G_3 was composed of the proved construction and the simulator. Moreover, both the preimages and the images of the map are footprints of the two systems respectively (in [2] the two systems were G_2 and G_3). After bounding the advantage of distinguishing G_2 and G_3, Andreeva *et al.* used a quite standard step to further transit to the real system G_4. Whereas we notice that it is not necessary to keep G_3 (the system composed of the proved construction and the simulator): if we define the image of the map as the exact copies of the tables generated by G_2 (the system composed of the target idealized primitive and the simulator) and serve such partial random tapes to the underlying building block in the real system G_4, then the map can directly link G_2 and G_4. As mentioned in Introduction, such a trick allows us to bypass an intermediate system (such a system was also used in [17] and [20]) and achieve the proof within only three systems.

Following the above, we first specify the domain of the map, then define the map and complete the proof.

Specifying the Domain: Bad Event BadHit To specify the domain of the map, we shall be aware of which tapes (η, φ) are able to induce executions same as those induced by f (from the viewpoint of $\mathbf{D}$). 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. Whereas such inconsistency never appears in Σ_2: this forms the difference.

Consider the Σ_1' executions during which $\widetilde{S}$ does not overwrite any entry. The footprints of the random tapes inducing such "non-overwriting" executions will be taken as the domain of the map (later we will show that such "non-overwriting" Σ_1' executions behave the same as the Σ_2 executions in the view of $\mathbf{D}$; see Lemma 20). Following the RRMA methodology, we shall show that most of the probability mass of all the possible Σ_1' executions are such "non-overwriting" executions; for this, we first define a 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.

The Bad Event **BadHit***, and the Probability* To define **BadHit**, we take the methodology introduced by Lampe and Seurin [20]. In $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, for each random tape accessing action, we define the history $\mathcal{H}$ as the set of all n-bit strings extracted from the entries in the table $\widetilde{E}(\eta).E$ and $\{G_i\}$ just before this action. More clearly:

- In table $\widetilde{E}(\eta).E$, for any entry $((\cdot,k,x),y)$, $\mathcal{H}$ includes five n-bit values k, x_L, x_R, y_L, y_R, where x_L and x_R (y_L and y_R) are the left and right n-bit halves of x (y, resp.).
- For each $i \in \{1,\ldots,21\}$, for any entry (x,y) in G_i, $\mathcal{H}$ includes two n-bit values x and y.
- The parameters passed to the calls which triggers the random tape accessing actions are also included, i.e. for calls to $F^{inner}(i,x)$, x is included in $\mathcal{H}$ immediately after the call is made, while for calls to $\widetilde{E}.Enc(k,m)$ (and $Dec(k,m)$), k, m_L, and m_R are all included in $\mathcal{H}$ immediately after the call is made, where m_L and m_R are the left and right n-bit halves of m.

Then we define the bad event **BadHit**:

Definition 2. *The bad event* **BadHit** *happens if when the simulator read an entry from the random tape, either of the following two happens:*

- *If the entry $x = (x_L, x_R)$ is a $2n$-bit value read from η, then either x_L or x_R equals the bitwise xor of 9 or less values in the history $\mathcal{H}$;*
- *If the entry y is an n-bit value read from φ, then y equals the bitwise xor of 9 or less values in $\mathcal{H}$.*

Based on the upper bounds on the size of the tables in $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, we upper bound the probability of **BadHit**.

Lemma 7. *For any distinguisher* $\mathbf{D}$ *which issues at most q queries, the probability (over the random choice of η and φ) that event* **BadHit** *happens in* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ *can be upper bounded as*

$$Pr[\mathbf{BadHit}] \leq \frac{2^{88} \cdot q^{30}}{2^n}$$

Proof. According to Lemma 2, $|\widetilde{E}(\eta).E^+| \leq 10q^3$, and $|G_i| \leq 10q^3$. By the definition of $\mathcal{H}$, each entry of $\widetilde{E}(\eta).E^+$ adds 5 values to $\mathcal{H}$[7], while each entry of G_i adds 2 values. Hence $|\mathcal{H}| \leq 5 \cdot 10q^3 + 2 \cdot 21 \cdot 10q^3 = 470q^3$. Then:

- for each query to F^{inner} which triggers φ_i tape accessing action, since values in φ_i are random, the probability that **BadHit** occurs is at most $\frac{(470q^3)^9}{2^n}$;

[7] The values added by $\widetilde{E}(\eta).E^-$ are the same as $\widetilde{E}(\eta).E^+$.

– for each query to $\widetilde{E}(\eta).Enc$ or $\widetilde{E}(\eta).Dec$ which triggers η tape accessing action, since η encodes an ideal cipher, the probability that **BadHit** occurs is at most $Pr[y_L \xleftarrow{\$} \{0,1\}^n : \textbf{BadHit}\ occurs\ on\ y_L] + Pr[y_R \xleftarrow{\$} \{0,1\}^n : \textbf{BadHit}\ occurs\ on\ y_R]$, and is upper bounded to $\frac{2\cdot(470q^3)^9}{2^n}$;

Then the probability that **BadHit** happens in no more than $21 \cdot 10q^3$ φ tape accessing actions is upper bounded to $\frac{(21\cdot 10q^3)\cdot(470q^3)^9}{2^n}$, while that in no more than $10q^3$ η tape accessing is upper bounded to $\frac{(10q^3)\cdot 2\cdot(470q^3)^9}{2^n}$. In total it is

$$\begin{aligned} Pr[\textbf{BadHit}] &\leq \frac{21 \cdot 10q^3 \cdot (470q^3)^9}{2^n} + \frac{10q^3 \cdot 2 \cdot (470q^3)^9}{2^n} \\ &\leq \frac{2^{88} \cdot q^{30}}{2^n} \end{aligned}$$

as claimed. □

The executions $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ during which **BadHit** does not happen are called *good executions*. The next section shows that during good executions, $\widetilde{S}(\varphi)$ never overwrites entries.

Good Executions: No Overwriting We first introduce necessary notions, then show a series of lemmas which finally establishes the claim of non-overwriting.

Necessary Notions and Functions Most of the notions used in this section are borrowed from [17] (some of them are redefined to fit into the $SKAF_{21}^*$ context). For $SKAF_{21}^*$, the partial chain is defined as a 4-tuple $(x_i, x_{i+1}, x_{i+2}, i)$ where $x_i, x_{i+1}, x_{i+2} \in \{0,1\}^n$ and $i \in \{0, \ldots, 20\}$. The associated key is $k = x_i \oplus G_{i+1}(x_{i+1}) \oplus x_{i+2}$ if $x_{i+1} \in G_{i+1}$. Further denote $C[1] = x_i$, $C[2] = x_{i+1}$, $C[3] = x_{i+2}$, $C[4] = i$. Given hash tables $G_1, \ldots, G_{21}$ and $\widetilde{E}(\eta).E$ at some point in the execution $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, we (re)define helper functions *next* and *prev* which take a partial chain C as input and return the partial chain obtained by moving respectively one step forward or backward in $SKAF_{21}^*$, or empty value $\perp$ when some necessary values (in tables G_i or $\widetilde{E}(\eta).E$) have not been defined. We also borrow (and redefine) the helper functions val_l^+ and val_l^- to help probe in the computation path; and a function k which returns the associated key value if the value can be calculated, or $\perp$ otherwise.

```
1: function next(x_i, x_{i+1}, x_{i+2}, i)
2:     if x_{i+1} ∉ G_{i+1} then
3:         return ⊥
4:     end if
5:     k := x_i ⊕ G_{i+1}(x_{i+1}) ⊕ x_{i+2}
6:     if i < 20 then
7:         if x_{i+2} ∉ G_{i+2} then
8:             return ⊥
9:         end if
```

```
        x_{i+3} := x_{i+1} ⊕ G_{i+2}(x_{i+2}) ⊕ k
        return (x_{i+1}, x_{i+2}, x_{i+3}, i + 1)
    else
        if (−, k, (x_{22}, x_{21})) ∉ E then
            return ⊥
        end if
        (x_1, x_0) := E(−, k, (x_{22}, x_{21}))
        if x_1 ∉ G_1 then
            return ⊥
        else
            x_2 := x_0 ⊕ G_1(x_1) ⊕ k
            return (x_0, x_1, x_2, 0)
        end if
    end if
end function
function prev(x_i, x_{i+1}, x_{i+2}, i)
    if x_{i+1} ∉ G_{i+1} then
        return ⊥
    end if
    k := x_i ⊕ G_{i+1}(x_{i+1}) ⊕ x_{i+2}
    if i > 0 then
        if x_i ∉ G_i then
            return ⊥
        end if
        x_{i−1} := x_{i+1} ⊕ G_i(x_i) ⊕ k
        return (x_{i−1}, x_i, x_{i+1}, i − 1)
    else
        if (+, k, (x_1, x_0)) ∉ E then
            return ⊥
        end if
        (x_{22}, x_{21}) := E(+, k, (x_1, x_0))
        if x_{21} ∉ G_{21} then
            return ⊥
        else
            x_{20} := x_{22} ⊕ G_{21}(x_{21}) ⊕ k
            return (x_{20}, x_{21}, x_{22}, 20)
        end if
    end if
end function
function val_l^+(C)
    if l ≥ 2 then
        while (C ≠ ⊥) ∧ (C[4] ∉ {l − 2, l − 1, l}) do
            C := next(C)
        end while
        if C = ⊥ then
```

```
            return ⊥
        else
            return C[(l − C[4] + 1)]
        end if
    else
        // l = 0 or 1
        while (C ≠ ⊥) ∧ (C[4] ≠ 20) do
            C := next(C)
        end while
        if C = ⊥ then
            return ⊥
        else if (−, k(C), (C[3], C[2])) ∉ E then
            return ⊥
        else
            (v[1], v[0]) := E(−, k(C), (C[3], C[2]))
            return v[l]
        end if
    end if
end function
function val_l^−(C)
    if l ≤ 20 then
        while (C ≠ ⊥) ∧ (C[4] ∉ {l − 2, l − 1, l}) do
            C := prev(C)
        end while
        if C = ⊥ then
            return ⊥
        else
            return C[(l − C[4] + 1)]
        end if
    else
        // l = 21 or 22
        while (C ≠ ⊥) ∧ (C[4] ≠ 0) do
            C := prev(C)
        end while
        if C = ⊥ then
            return ⊥
        else if (+, k(C), (C[2], C[1])) ∉ E then
            return ⊥
        else
            (v[22], v[21]) := E(+, k(C), (C[2], C[1]))
            return v[l]
        end if
    end if
end function
function k(x_j, x_{j+1}, x_{j+2}, j)
```

2: **if** $x_{j+1} \notin G_{j+1}$ **then**
3: **return** $\perp$
4: **else**
5: **return** $x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2}$
6: **end if**
7: **end function**

Notations $prev^j$ and $next^j$ are used to denote the j-th functional power of $prev$ and $next$ respectively. To make the functional powers well-defined, we define $prev(\perp) = \perp$ and $next(\perp) = \perp$[8]. Then we define *equivalent* and *table-defined* partial chains (the two notions are actually borrowed from [17,20]):

Definition 3. *Two partial chains C and D are* equivalent *if $C = D$, or for some $1 \leq j \leq 20$, $C = next^j(D)$ or $C = prev^j(D)$. Denote it by $C \equiv D$.*

Definition 4. *For the hash tables $G_1, \ldots, G_{21}$, and $\widetilde{E}(\eta).E$ at some point in the execution of $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, a partial chain $C = (x_i, x_{i+1}, x_{i+2}, i)$ is said to be table-defined if $next(C) \neq \perp$ and $prev(C) \neq \perp$.*

Note that when $i \in \{1, \ldots, 19\}$, $C = (x_i, x_{i+1}, x_{i+2}, i)$ being table-defined implies $x_i \in G_i$, $x_{i+1} \in G_{i+1}$, and $x_{i+2} \in G_{i+2}$, i.e. all the three values involved in C have been in the history $\mathcal{H}$ of the execution. When $i = 0$ ($= 20$, resp.), it means that (x_1, x_0) ((x_{22}, x_{21}), resp.) has been in table E, which also implies that all the three values involved in C have been in $\mathcal{H}$.

We call a call to *Adapt safe*, if during the *Adapt* procedure, the function values in *at least one of the two* buffer rounds $l-2$ and $l-1$ ($l+2$ and $l+3$, resp.) has not been defined and can be freely set to fresh random values. This is more involved than the analogue in [20]. For simplicity, we will call such a condition *safe Adapt call condition* in the following sections.

Definition 5. *A call to $Adapt(x_{l-4}, x_{l-3}, x_{l-2}, x_{l+3}, x_{l+4}, x_{l+5}, l)$ is called* safe *if the following holds before the call:* [9]

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

where $B = (x_{l-4}, x_{l-3}, x_{l-2}, l-4)$ and $D = (x_{l+3}, x_{l+4}, x_{l+5}, l+3)$.

Definition 6. *A call to $ForceVal(x, y, l)$ is called* non-overwriting *if $x \notin G_l$ before the call.*

Two notions *key-defined* and *key-undefined* are introduced to reveal whether the associated key of a partial chain can be calculated from the tables.

Definition 7. *A partial chain $C = (x_j, x_{j+1}, x_{j+2}, j)$ is called* key-defined *if $x_{j+1} \in G_{j+1}$; otherwise is called* key-undefined.

[8] This makes the definition well-defined. However this actually has no additional influence on the subsequent proof.

[9] By construction, when *Adapt* is called, $k(B) \neq \perp$ and $k(D) \neq \perp$ must hold.

Note that:

– For a partial chain C, $k(C) \neq \perp$ if and only if C is key-defined.
– Each table-defined chain is also key-defined.

Good Executions: Properties In this section we show some properties that are helpful for the proof of our main goal, i.e. $\widetilde{S}(\varphi)$ never overwrites entries. Most of the properties are very close to those of un-keyed Feistel construction (exhibited in [17]). We list the main differences as follows:

(i) Our Lemma 8 focus on key-defined chains, which is a new notion in this work;
(ii) Our Lemma 11 concerns with two round values instead of only one in the previous works. This is due to the n bits increment in the entropy of the chain;
(iii) Due to the increment in the number of buffer rounds, Lemma 12 and Lemma 14 are slightly different from their previous analogues;
(iv) to tackle the chains which were key-undefined before being enqueued, we add a lemma: Lemma 16.
(v) at last, the idea to achieve the proof of Lemma 17 is also affected by the increment in the number of buffer rounds.

As to the question *why almost all the analyzes focus on key-defined chains*, the reasons are two-fold:

(i) The core of the analyzes is on the chains that are enqueued and completed by the simulator. Most of these chains have been key-defined right before the calls to F^{inner} which trigger them to be enqueued. The only exception is the case when a chain $(x_{10}, x_{11}, x_{12}, 10)$ is enqueued by a call to $F^{inner}(11, x_{11})$; such chains are captured by the set $KUDCS$ (see page 32), and separately analyzed in Lemma 16.
(ii) For a chain $C = (x_i, x_{i+1}, x_{i+2}, i)$, the three values $k(C)$, $val^{-}_{i-1}(C)$, and $val^{+}_{i+3}(C)$ all depend on $G_{i+1}(x_{i+1})$. The absence of $G_{i+1}(x_{i+1})$ limits the influence of C to a very restricted extent (see Lemma 26 in the Appendix).

After highlighting the main differences, we present the lemmas themselves. First, in the good executions, the key-defined chains behave quite similar to the partial chains analyzed by Holenstein *et al.*: the random tape accessing and the subsequent entry setting actions can only extend them one round each time.

Lemma 8. *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 $next(C) = \perp$ 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 $next^2(C) = \perp$.
(ii) For any key-defined partial chain C, if $prev(C) = \perp$ 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 $prev^2(C) = \perp$.

(iii) 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. We prove the propositions one-by-one.

(i) Suppose $C = (x_l, x_{l+1}, x_{l+2}, l)$, and assume otherwise, i.e. $next(next(C)) \neq \perp$. Depending on l, we distinguish two cases:
- $l < 20$: in this case, before the action, from $next(C) = \perp$ and C was key-defined we know $x_{l+2} \notin G_{l+2}$. Since C turns to be table-defined after the action, the action must be $G_{l+2}(x_{l+2}) := \varphi_{l+2}(x_{l+2})$. Then if $next(next(C)) \neq \perp$ holds after the action, it must fall in one of the following two cases:
  (a) when $l \leq 19$, we have

$$\begin{aligned} x_{l+3} &= x_{l+1} \oplus \varphi_{l+2}(x_{l+2}) \oplus k(C) \\ &= x_{l+1} \oplus \varphi_{l+2}(x_{l+2}) \oplus x_l \oplus G_{l+1}(x_{l+1}) \oplus x_{l+2} \in G_{l+3}, \end{aligned}$$

  which means the value read from φ equals xor of 5 values in $\mathcal{H}$: $\varphi_{l+2}(x_{l+2}) = x_{l+1} \oplus x_l \oplus G_{l+1}(x_{l+1}) \oplus x_{l+2} \oplus x_{l+3}$.
  (b) when $l = 19$, we have $(-, k(C), (x_{22}, x_{21})) \in E$ (and $x_{22} \in \mathcal{H}$) for $x_{22} = x_{20} \oplus G_{21}(x_{21}) \oplus k(C)$, which (also) means the value read from φ equals xor of 5 values in $\mathcal{H}$: $\varphi_{21}(x_{21}) = x_{20} \oplus x_{19} \oplus G_{20}(x_{20}) \oplus x_{21} \oplus x_{22}$.

  Both of the two contradict the assumption that **BadHit** does not occur.
- $l = 20$ ($C = (x_{20}, x_{21}, x_{22}, 20)$): in this case, the assumption that before the action $next(C) = \perp$ while after the action C turns to be table-defined and $next(next(C)) \neq \perp$ means, before the action, exactly one of the following two holds:
  (a) $(-, k(C), (x_{22}, x_{21})) \notin E$
  (b) $(-, k(C), (x_{22}, x_{21})) \in E \wedge x_1 \notin G_1$ for $(x_1, x_0) = E(-, k(C), (x_{22}, x_{21}))$

  while after the action, all the following three hold:
  - $(-, k(C), (x_{22}, x_{21})) \in E$ (by C being table-defined)
  - for $(x_1, x_0) = E(-, k(C), (x_{22}, x_{21}))$, $x_1 \in G_1$ (by C being table-defined and $next(next(C)) \neq \perp$)
  - $x_2 \in G_2$ for $x_2 = x_0 \oplus G_1(x_1) \oplus k(C)$ (by $next(next(C)) \neq \perp$)

  We exclude the possibility for each case:
  (a) the first case: $(-, k(C), (x_{22}, x_{21})) \notin E$ before the action, and the action is $E(-, k(C), (x_{22}, x_{21})) := \eta(-, k(C), (x_{22}, x_{21}))$. In such case, since $x_1 \in G_1$ after the action, the value m read from η satisfies $m_L = x_1$ which has been in $\mathcal{H}$, a contradiction.

(b) the second case: $x_1 \notin G_1$ before the action, and the action is $G_1(x_1) := \varphi_1(x_1)$. In such case, after the action we have $x_2 = x_0 \oplus \varphi_1(x_1) \oplus k(C) \in G_2$, hence $\varphi_1(x_1) = x_0 \oplus (x_{20} \oplus G_{21}(x_{21}) \oplus x_{22}) \oplus x_2$, a contradiction.

(ii) The proof is similar to (i) by symmetry.

(iii) By construction, the action $G_j(x_j) := \varphi_j(x_j)$ never overwrites entries in the tables. Hence $val_i^+(C)$ or $val_i^-(C)$ can only change from $\perp$ to non-empty values. Let $y_j = \varphi_j(x_j)$, and *wlog* consider the case when $val_i^+(C)$ changes. According to the implementation of function val_i^+, before the action $G_j(x_j) := y_j$, $val_i^+(C) = \perp$ might be due to either of the following two:

(a) for some $1 \leq l \leq 21$, $val_l^+(C) \neq \perp \wedge val_l^+(C) \notin G_l$, and calculating $val_i^+(C)$ requires $val_l^+(C) \in G_l$;

(b) $val_{22}^+(C) \neq \perp \wedge val_{21}^+(C) \neq \perp \wedge (-, k(C), (val_{22}^+(C), val_{21}^+(C))) \notin E$;

In the second case, $G_j(x_j) := y_j$ will not affect $val_i^+(C)$ since after the action, $(-, k(C), (val_{22}^+(C), val_{21}^+(C))) \notin E$ still holds. In the first case we argue $j = l = i - 1 \wedge val_j^+(C) = x_j$ to hold:

(a) if $j \neq l \vee (j = l \wedge val_j^+(C) \neq x_j)$, then $G_j(x_j) := y_j$ will not affect $val_i^+(C)$ since after the action, $val_l^+(C) \notin G_l$ still holds;

(b) if $j = l < i - 2$, then $val_{j+2}^+(C) = x_j \oplus G_{j+1}(val_{j+1}^+(C)) \oplus (val_{j-1}^+(C) \oplus y_j \oplus val_{j+1}^+(C))$ must be either $\perp$ (when $val_{j+1}^+(C) \notin G_{j+1}$) or non-empty value which has not been in G_{j+2} (to avoid **BadHit**). Since $j + 2 < i$, this implies $val_i^+(C) = \perp$ after the action – $val_i^+(C)$ does not change;

(c) if $j = l = i - 2$, for $val_i^+(C) = val_{i-2}^+(C) \oplus G_{i-1}(val_{i-1}^+(C)) \oplus k(C) \neq \perp$ to hold after the action, $val_{i-1}^+(C) \in G_{i-1}$ must hold before the action. In such case, if $C = (val_{i-3}^+(C), val_{i-2}^+(C), val_{i-1}^+(C), i - 3)$, then C is key-undefined before the action, contradicting the assumption; let $C^* = (val_{i-4}^+(C), val_{i-3}^+(C), val_{i-2}^+(C), i - 4)$, if $C = prev^s(C^*)$ for some $s \geq 0$, then $next(C^*) = \perp$ before the action while after the action, either $next(C^*) = \perp$ (when $val_{i-3}^+(C) \notin G_{i-3}$) which contradicts the assumption that $val_i^+(C)$ changes, or $next^2(C^*) \neq \perp$ (when $val_{i-3}^+(C) \in G_{i-3}/C^*$ is key-defined) which contradicts proposition (i).

We then show the uniqueness. Let $C' = (val_{i-3}^+(C), val_{i-2}^+(C), val_{i-1}^+(C), i - 3)$. By discussions above we know C' must have been key-defined before the action. *Wlog*, for some $i' \geq i + 1$, suppose $val_{i'}^+(C)$ also changes from $\perp$ to non-empty values. Then $next(C') = \perp$ before $G_j(x_j) := \varphi_j(x_j)$ while $next^2(C') \neq \perp$ after $G_j(x_j) := \varphi_j(x_j)$, contradicting proposition (i).

Finally, if $val_i^+(C) \in G_i$ after the action, then $next(C') = \perp$ before the action while $next^2(C') \neq \perp$ after the action, contradicting proposition (i).

Hence we establish the claim for $val_i^+(C)$. The reasoning for $val_i^-(C)$ is similar by symmetry. □

The next lemma presents basic properties of the equivalence relation $\equiv$.

Lemma 9. *During a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta), \widetilde{S}(\varphi))}$*, at any point such that all the previous calls to ForceVal were non-overwriting, the following hold:*

(i) *For any two partial chains C and D, $next(C) = D \Leftrightarrow prev(D) = C$.*
(ii) *The relation $\equiv$ between partial chains is an equivalence relation.*
(iii) *If two table-defined partial chains C and D are equivalent at this point, then there exists a sequence of table-defined chains $C_1, \ldots, C_r (r \geq 1)$ s.t.*
- *$C = C_1$ and $D = C_r$, or $C = C_r$ and $D = C_1$.*
- *$C_i = next(C_{i-1})$ and $C_{i-1} = prev(C_i)$.*

Proof. (i) By construction, only $ForceVal$ can overwrite entries. Since all are assumed to be non-overwriting, both evaluating $SKAF^*$ one step forward or backward and evaluating $E.Enc$ or $E.Dec$ are bijective and (i) holds.
(ii) Due to (i), $\equiv$ is symmetric, and by definition it is reflexive and transitive.
(iii) By definition we know $D = next^j(C)$ or $D = prev^j(C)$ for some j. In the former case the chain sequence is $C_1 = C$, $C_2 = next(C)$, ..., $C_i = next^{i-1}(C)$, ..., $C_r = D$ where $r = j + 1$; in the latter case it is similar by symmetry. Clearly, all the chains are table-defined. □

Then we show the invariance of the equivalence relation for chains before and after the tape accessing and entry setting action.

Lemma 10. *Consider a good execution $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$. Let C and D be two table-defined chains at some point in $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ such that all the previous calls to $ForceVal$ were non-overwriting. Assume a tape accessing and entry setting action happens after this point, then the equivalence of C and D is invariant before and after this action.*

Proof. If $C \equiv D$ before the action, then the chain sequence linked C and D is invariant before and after the action since no entry is overwritten, and $C \equiv D$ after the action. On the other hand if sequence $C_1, \ldots, C_r$ links C and D after the action while $next(C_j) = prev(C_{j+1}) = \perp$ before the action, then D being table-defined before the action implies $j + 1 < r$. In this case, $next(C_j) = \perp$ before the action while $next(C_j) \neq \perp \wedge next^2(C_j) \neq \perp$ after the action, which contradicts Lemma 8 (i). Hence $C \equiv D$ before the action. □

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

Lemma 11. *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. Towards a contradiction assume all the four statements to hold after an action $G_i(x_i) := \varphi_i(x_i)$ for two partial chains C and D, some $l \in \{1, \ldots, 21\}$, and $\delta = +$. Let $y_i = \varphi_i(x_i)$, and *wlog* assume $val_l^+(C) = \perp$ before the action. Then since the two chains are key-defined before the action and $val_l^+(C)$ is changed from $\perp$ to non-empty values by the action, by Lemma 8 (iii), $i = l - 1$, and $val_i^+(C) = x_i$ must hold before the action. We distinguish two cases:

- $val_{i+1}^+(D) = \perp$ before the action. Then $val_i^+(D) = x_i$ must hold before the action. Let $(x_{i-2}, x_{i-1}, x_i, i-2) = next^{j_1}(C)$ and $(x'_{i-2}, x'_{i-1}, x_i, i-2) = next^{j_2}(D)$ for sufficiently large j_1 and j_2. If $val_{i+1}^+(C) = val_{i+1}^+(D) = x_{i+1} \neq \perp$ holds after the action, then after the action we have $x_{i-1} \oplus G_i(x_i) \oplus k(C) = x'_{i-1} \oplus G_i(x_i) \oplus k(D)$, which implies $x_{i-1} \oplus k(C) = x'_{i-1} \oplus k(D)$ to hold before the action. Note that $x_{i-1} = x'_{i-1}$ cannot hold, since otherwise $(x_{i-1}, x_i, x_{i+1}, i-1) = (x'_{i-1}, x_i, x_{i+1}, i-1)$ and $C \equiv D$ which contradicts the assumption that C is not equivalent to D before the action. Therefore $k(C) = x_{i-1} \oplus G_i(x_i) \oplus x_{i+1} \neq x'_{i-1} \oplus G_i(x_i) \oplus x_{i+1} = k(D)$, and

$$val_{l-1}^+(C) \oplus k(C) = x_i \oplus k(C) \neq x_i \oplus k(D) = val_{l-1}^+(D) \oplus k(D).$$

  This implies that the four statements cannot simultaneously hold.
- $val_{i+1}^+(D) \neq \perp$ before the action. Then $val_i^+(C) \neq val_i^+(D)$ must hold, otherwise $val_{i+1}^+(C) \neq \perp$. Let $(x_{i-2}, x_{i-1}, x_i, i-2) = next^{j_1}(C)$ and $(x'_{i-2}, x'_{i-1}, x_i, i-2) = next^{j_2}(D)$ for sufficiently large j_1 and j_2. Since D is table-defined after the entry setting action on $G_i(x_i)$, D must have been table-defined before the action. Further since $val_l^+(C) = val_l^+(D) \neq \perp$ after the action, we have $x_{i-1} \oplus y_i \oplus k(C) = x_{i-1} \oplus \varphi_i(x_i) \oplus k(C) = x'_{i-1} \oplus G_i(x'_i) \oplus k(D)$, i.e.

$$\varphi_i(x_i) = x_{i-1} \oplus x_{i-2} \oplus G_{i-1}(x_{i-1}) \oplus x_i \oplus x'_{i-1} \oplus G_i(x'_i) \oplus x'_{i-2} \oplus G_{i-1}(x'_{i-1}) \oplus x_i,$$

  BadHit happens.

The reasoning for $\delta = -$ is similar by symmetry. □

The following lemma claims that 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.

Lemma 12. *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) *If two chains C and D are table-defined before this call to Adapt, then the equivalence of C and D is invariant before and after the subsequent calls to $ForceVal$.*

Proof. Denote the chain being completed by $C_{complete}$. Then, for proposition (i), consider x_{l-1}. Since the call to *Adapt* is safe, before the *Adapt* either $x_{l-2} \notin G_{l-2}$ or $x_{l-2} \in G_{l-2} \wedge x_{l-1} \notin G_{l-1}$ holds. If the latter holds, then clearly proposition (i) holds. If the former holds, then action $G_{l-2}(x_{l-2}) := \varphi_{l-2}(x_{l-2})$ must happen during the call to $F^{inner}(l-2, x_{l-2})$. Before this action, $x_{l-4} \in G_{l-4}$, $x_{l-3} \in G_{l-3}$, and $next(x_{l-4}, x_{l-3}, x_{l-2}, l-4) = \perp$, hence by Lemma 8 (i), after this action we have $next^2(x_{l-4}, x_{l-3}, x_{l-2}, l-4) = \perp$ and $x_{l-1} \notin G_{l-1}$. Utilizing Lemma 8 (ii), and by symmetry we achieve the proof of $x_{l+2} \notin G_{l+2}$.

Gathering proposition (i) and Lemma 8 (i), (ii) yields $x_l \notin G_l$ and $x_{l+1} \notin G_{l+1}$ before the call to $ForceVal$. Therefore proposition (ii) holds.

For proposition (iii), consider a chain $C = (x_j, x_{j+1}, x_{j+2}, j)$ which is table-defined before the call to *Adapt*. Let $B = (x_{l-4}, x_{l-3}, x_{l-2}, l-4)$ and $D = (x_{l+3}, x_{l+4}, x_{l+5}, l+3)$. Then $B \equiv D \equiv C_{complete}$ while C cannot be equivalent to them by assumption. Suppose $val_i^+(C)$ is changed by the subsequent calls to $ForceVal$. Then $val_l^+(C) = x_l$ or $val_{l+1}^+(C) = x_{l+1}$ must hold before the calls to $ForceVal$, for the value of $val_i^+(C)$ to change.

We first assume $val_l^+(C) = x_l$ right before the two calls to $ForceVal$. For each of the following cases, we exclude the possibility:

(i) Before the call to *Adapt*, $val_l^+(C) \neq \perp$. Then $val_l^+(C)$ can be written as xor of five values extracted from the history: $val_l^+(C) = val_{l-2}^+(C) \oplus G_{l-1}(val_{l-1}^+(C)) \oplus x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2}$. By proposition (i), $x_{l-1} \notin G_{l-1}$ before $y_{l-1} := F^{inner}(l-1, x_{l-1})$ is executed, therefore the value x_l calculated by $x_l := x_{l-2} \oplus \varphi_{l-1}(x_{l-1}) \oplus k(B)$ cannot equal $val_l^+(C)$ unless $\varphi_{l-1}(x_{l-1}) = (val_{l-2}^+(C) \oplus G_{l-1}(val_{l-1}^+(C)) \oplus x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2}) \oplus (x_{l-2} \oplus x_{l-4} \oplus G_{l-3}(x_{l-3}) \oplus x_{l-2})$ and **BadHit** happens.

(ii) Before the call to *Adapt*, $val_l^+(C) = \perp$. We further distinguish two cases depending on $val_{l-2}^+(C)$ before the call to *Adapt*:

(a) $val_{l-2}^+(C) \in G_{l-2}$ while $val_{l-1}^+(C) \notin G_{l-1}$: then $val_{l-1}^+(C) = x_{l-1}$ must hold when $y_{l-1} := F^{inner}(l-1, x_{l-1})$ is executed, otherwise when $ForceVal$ is called, $val_l^+(C) = \perp \neq x_l$. Then, before the call to *Adapt*:

- if $x_{l-2} \notin G_{l-2}$, then $val_{l-1}^+(C) = x_{l-1}$ cannot hold when $y_{l-1} := F^{inner}(l-1, x_{l-1})$ is executed unless $\varphi_{l-2}(x_{l-2}) = (val_{l-3}^+(C) \oplus G_{l-2}(val_{l-2}^+(C)) \oplus x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2}) \oplus (x_{l-3} \oplus x_{l-4} \oplus G_{l-3}(x_{l-3}) \oplus x_{l-2})$ and **BadHit** happens.
- if $x_{l-2} \in G_{l-2}$ while $x_{l-1} \notin G_{l-1}$, the call to *Adapt* must set $G_{l-1}(val_{l-1}^+(C))$ besides $G_{l-1}(x_{l-1})$ to make $val_l^+(C) = x_l \neq \perp$ before the call to $ForceVal$. This means, before the call to *Adapt*:
  i. $val_{l-1}^+(C) = x_{l-1}$ holds;

ii. $val_l^+(C) = val_l^+(B)$ immediately holds after $G_{l-1}(x_{l-1})$ is defined;

These imply the following two to hold simultaneously:

i. $val_{l-3}^+(C) \oplus G_{l-2}(val_{l-2}^+(C)) \oplus (val_{l-2}^+(C) \oplus G_{l-3}(val_{l-3}^+(C)) \oplus val_{l-4}^+(C)) = x_{l-3} \oplus G_{l-2}(x_{l-2}) \oplus (x_{l-4} \oplus G_{l-3}(x_{l-3}) \oplus x_{l-2})$;

ii. $x_{l-2} \oplus (x_{l-4} \oplus G_{l-3}(x_{l-3}) \oplus x_{l-2}) = val_{l-2}^+(C) \oplus (val_{l-2}^+(C) \oplus G_{l-3}(val_{l-3}^+(C)) \oplus val_{l-4}^+(C))$;

Therefore all the following 6 values have been in $\{G_i\}$ before the call to *Adapt*: $val_{l-4}^+(C)$, $val_{l-3}^+(C)$, $val_{l-2}^+(C)$ (by C being table-defined before the call), x_{l-4}, x_{l-3}, and x_{l-2}. By construction we know all the entries in G_{l-4}, G_{l-3}, and G_{l-2} are defined to random values drew from the tapes (none of the 3 rounds $l-4$, $l-3$ and $l-2$ is in adaptation zone), consequently among them six, the last one added to $\{G_i\}$ implies **BadHit**, and it is impossible for this case to occur.

(b) $val_{l-2}^+(C) \notin G_{l-2}$ (which means $val_{l-1}^+(C) = \perp$): in such case the call to *Adapt* must define $G_{l-2}(val_{l-2}^+(C))$ and $G_{l-1}(val_{l-1}^+(C))$ besides $G_{l-2}(x_{l-2})$ and $G_{l-1}(x_{l-1})$ to lead to $val_l^+(C) \neq \perp$. Then $val_l^+(C) = val_l^+(B) = x_l$ is impossible since C is not equivalent to B.

We then assume $val_{l+1}^+(C) = x_{l+1}$ just before the second call to $ForceVal$. By the discussions above, we have: before the call to *Adapt*, $val_l^+(C) \in G_l$ must hold, otherwise $val_l^+(C) \notin G_l$ is kept till the second call to $ForceVal$, which implies $val_{l+1}^+(C) = \perp \neq x_{l+1}$, a contradiction. Then $val_{l+1}^+(C)$ can be written as xor of five values in the history: $val_{l+1}^+(C) = val_{l-1}^+(C) \oplus G_l(val_l^+(C)) \oplus x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2}$; and, by proposition (i) we know $val_{l+1}^+(C) = x_{l+1}$ cannot hold before the second call to $ForceVal$, otherwise the call $F^{inner}(l+2, x_{l+2})$ triggers **BadHit**, i.e. $\varphi_{l+2}(x_{l+2}) = val_{l-1}^+(C) \oplus G_l(val_l^+(C)) \oplus x_j \oplus G_{j+1}(x_{j+1}) \oplus x_{j+2} \oplus x_{l+3} \oplus x_{l+2} \oplus G_{l+3}(x_{l+3}) \oplus x_{l+4}$.

The reasoning for $val_i^-(C)$ is similar. These establish proposition (iii).

For proposition (iv), let C and D be two chains which are table-defined before the *Adapt*. If $C \equiv D \equiv C_{completed}$ before the call to $ForceVal$, then clearly $C \equiv D \equiv C_{completed}$ after the call since no entry is overwritten; and the proposition that if $C \equiv D \equiv C_{completed}$ after the call to $ForceVal$ then $C \equiv D \equiv C_{completed}$ before the call is also trivial. If C and D are not equivalent to $C_{completed}$, then by proposition (iii) the values $val_i^\delta(C)$ and $val_i^\delta(D)$ are invariant before and after the calls to $ForceVal$, $C \equiv D$ before $ForceVal$ implies $C \equiv D$ after $ForceVal$ meanwhile $C \equiv D$ after $ForceVal$ only if $C \equiv D$ before $ForceVal$. □

Given that no entry is overwritten, a chain dequeued form the queue will not be completed if some chain equivalent to it has been completed.

Lemma 13. *Assume that at a fixed point in a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*, a chain* C *is dequeued such that* $C \notin CompletedSet$ *and all calls to Adapt were safe up to the point* C *is dequeued. Then when* C *was enqueued, no chain equivalent to* C *has been enqueued.*

Proof. Towards a contradiction assume a chain $D \equiv C$ has been enqueued before C was enqueued. Since **BadHit** is assumed to be absent, and all the previous calls to $ForceVal$ are assumed to be safe, by Lemma 10 and Lemma 12 (iv), $D \equiv C$ till C is dequeued. Hence when C is dequeued, D must have been dequeued and completed, and the completion of D must have added C to $CompletedSet$. This contradicts the assumption $C \notin CompletedSet$ when C is dequeued. □

Lemma 14. *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. The aim is to show the *safe Adapt call condition* to hold right before the call $Adapt(x_{l-4}, x_{l-3}, x_{l-2}, x_{l+3}, x_{l+4}, x_{l+5}, l)$. *Wlog* we show this for "x_{l-2} side". By construction, $C = (x_i, x_{i+1}, x_{i+2}, i)$ must be key-defined since being enqueued. Then, when C is enqueued,

(i) if $val^+_{l-2}(C) = \perp$, then by Lemma 8 (iii), $val^+_{l-2}(C)$ can only change from $\perp$ to some non-empty values during the random tape accessing and entry setting action on G_{l-3} which occurs in the procedure $EvalForward(C, l-2)$, and by Lemma 8 (i), $val^+_{l-2}(C) \notin G_{l-2}$ immediately holds after this action;
(ii) if $val^+_{l-2}(C) \neq \perp \wedge val^+_{l-2}(C) \notin G_{l-2}$ (i.e. $val^+_{l-1}(C) = \perp$), then $x_{l-2} = val^+_{l-2}(C) \notin G_{l-2}$ keeps holding till *Adapt* is called;
(iii) if $val^+_{l-2}(C) \in G_{l-2} \wedge val^+_{l-1}(C) \notin G_{l-1}$, then $x_{l-1} = val^+_{l-1}(C) \notin G_{l-1}$ keeps holding till *Adapt* is called;

By discussions above, the safe *Adapt* call condition holds before *Adapt*. □

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

Then, the following lemmas show that the assumptions of Lemma 14 hold in a good execution: Lemma 15 shows them to hold before the chains are enqueued, Lemma 17 shows them to hold till the chains are dequeued, while Lemma 16 is a helper lemma for Lemma 17.

Lemma 15. *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.*

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

(ii) right after C is enqueued, $val^+_{l-2}(C) \notin G_{l-2} \wedge val^-_{l+3}(C) \notin G_{l+3}$.

Proof. The two propositions will be argued simultaneously. First, consider the case $l = 6$. For the following three cases:

(i) $C = (x_1, x_2, x_3, 1, 6)$ is enqueued by a call to $F^{inner}(3, x_3)$: clearly before this call, $val^+_4(C) = \perp$. Assume $val^-_9(C) \neq \perp$, then $val^-_{10}(C) \in G_{10}$, $val^-_{11}(C) \in G_{11}$, $val^-_{12}(C) \in G_{12}$, and $(val^-_{10}(C), val^-_{11}(C), val^-_{12}(C), 10)$ would be a chain which is equivalent to C and has been enqueued before C is enqueued. This contradicts the assumption.
After the action $G_3(x_3) := \varphi_3(x_3)$ triggered by $F^{inner}(3, x_3)$, by Lemma 8 (iii) we have $val^+_4(C) \notin G_4$. Moreover $val^-_9(C) = \perp \notin G_9$ keeps holding.

(ii) $C = (x_{10}, x_{11}, x_{12}, 10, 6)$ is enqueued by a call to $F^{inner}(10, x_{10})$: clearly $val^-_9(C) = \perp$ before it. Assume $val^+_4(C) \neq \perp$, then let $(x_0, x_1, x_2, x_3, x_{19}, x_{20}, x_{21}, x_{22}) = (val^+_0(C), val^+_1(C), val^+_2(C), val^+_3(C), val^-_{19}(C), val^-_{20}(C), val^-_{21}(C), val^-_{22}(C))$. All these 8 values must be non-empty values, since otherwise $val^+_4(C) = \perp$. Then at some point in the execution, all the 7 values $G_1(x_1), G_2(x_2), G_3(x_3), G_{19}(x_{19}), G_{20}(x_{20}), G_{21}(x_{21})$, and $E(-, k, (x_{22}, x_{21}))$ have been added to corresponding tables. Among them, consider the last added one. It must have been $G_3(x_3) := \varphi_3(x_3)$ or $G_{19}(x_{19}) := \varphi_{19}(x_{19})$ because:
   (a) it cannot be an action happened on table E, since otherwise $prev(x_0, x_1, x_2, 0) = \perp$ before the action while $prev^2(x_0, x_1, x_2, 0) \neq \perp$ after the action (contradicting Lemma 8 (ii)).
   (b) it cannot be $G_1(x_1) := \varphi_1(x_1)$ or $G_2(x_2) := \varphi_2(x_2)$ since otherwise $val^+_4(x_{19}, x_{20}, x_{21}, 19)$ changes (contradicting Lemma 8 (iii)).
   (c) it cannot be $G_{20}(x_{20}) := \varphi_{20}(x_{20})$ or $G_{21}(x_{21}) := \varphi_{21}(x_{21})$ since otherwise $val^-_{18}(x_1, x_2, x_3, 1)$ changes (contradicting Lemma 8 (iii)).

   Hence the last action (on round 3 or 19) will trigger the chain detection and completion process and add C to $CompletedSet$, which contradicts the assumption.
   In this case, after C is enqueued, $val^-_9(C) \notin G_9$ immediately holds by Lemma 8 (iii), while $val^+_4(C) = \perp \notin G_4$ keeps holding.

(iii) $C = (x_{10}, x_{11}, x_{12}, 10, 6)$ is enqueued by a call to $F^{inner}(11, x_{11})$: in this case, before C is enqueued, C is *key-undefined*, and $val^-_9(C) = \perp$ and $val^+_{13}(C) = \perp$ hold. Furthermore, $val^-_9(C) \notin G_9$ and $val^+_{13}(C) \notin G_{13}$ immediately hold after C is enqueued, otherwise **BadHit** happens e.g. if $val^-_9(C) = x_9 \in G_9$ then $\varphi_{11}(x_{11}) = x_9 \oplus x_{11} \oplus G_{10}(x_{10}) \oplus x_{10} \oplus x_{12}$. $val^+_{13}(C) \notin G_{13}$ further implies $val^+_4(C) = \perp \notin G_4$.

For $l = 15$, it is similar by symmetry, except for excluding case (iii). □

Lemma 16. *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;*

(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 $val_9^-(C) \neq \bot \wedge val_9^-(C) = val_9^-(D) \wedge val_{10}^-(C) \oplus k(C) = val_{10}^-(D) \oplus k(D)$*;*

In other words, C and D cannot collide at both round 4 and round 5; C and D cannot collide at both round 8 and round 9.

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 15);
(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 reading and entry setting actions (by Lemma 12 (iii)) on $\{G_i\}$ (by Lemma 8 (iii)). Then proposition (i) is established by Lemma 11 (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 15);
(ii) C and D_{td} ($D \equiv D_{td}$) are table-defined/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[11]. 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 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 was 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 17. *In a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta), \widetilde{S}(\varphi))}$*, all calls to Adapt are safe.*

[11] When $D = (x'_{10}, x'_{11}, x'_{12}, 10)$, the claim follows from Lemma 15 (i); when $D = (x'_1, x'_2, x'_3, 1)$, the claim follows from the assumption that D is enqueued earlier than C.

Proof. Suppose that the lemma does not hold, and let C be the first chain for which it fails. Clearly $C \notin CompletedSet$ when C is dequeued, and since all calls to *Adapt* before C is dequeued were safe, by Lemma 13 we know when C was enqueued, no chain equivalent to C had been enqueued. Hence, Lemma 15 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 the conclusion by Lemma 14. *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 12 (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 8 (iii)).

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 $val^+_{l-1}(C) \notin G_{l-1}$ to hold 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 15 (i), before C is enqueued, $val^+_{l-2}(C)$ shall 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 16; if $C \notin KUDCS$, for the action, we exclude possibility for each case:

(i) This cannot be 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 four hold simultaneously, which contradicts Lemma 11:
  - before the action, both C_{td} and D_{td} are key-defined. C_{td} is key-defined because $C \notin KUDCS$. D_{td} is key-defined because D is assumed to be enqueued before C is enqueued, and: (a) if D is enqueued before the call to F^{inner} which triggers C to be enqueued, then D_{td} clearly has been key-defined before the action we focus on; (b) if D and C are enqueued by the same call to F^{inner}, then it cannot be $D \in KUDCS$, hence D_{td} is key-defined.
  - before the action, C_{td} is not equivalent to D_{td};
  - before the action, $val^+_{l-2}(C_{td}) = \bot$ or $val^+_{l-2}(D_{td}) = \bot$, and this action affects one (or both) of them.
  - after the action, C_{td} and D_{td} are table-defined;
  - after the action, $val^+_{l-2}(D_{td}) = val^+_{l-2}(C_{td}) \neq \bot$ and $val^+_{l-3}(D_{td}) + k(D_{td}) = val^+_{l-3}(C_{td}) + k(C_{td})$;

(ii) This cannot be an entry setting action on E, since such actions cannot change $val^+_{l-2}(C_{td})$ nor $val^+_{l-2}(D_{td})$ (by Lemma 8 (iii));

(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 12 (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 15 (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 12 (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 $\bot$ 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 8 (iii)). However neither is possible: *wlog* assume it 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

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

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. □

Lemma 18. *In a good execution* $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$*, all calls to* $ForceVal$ *are non-overwriting.*

Proof. Gathering Lemma 17 and Lemma 12 (ii) yields this lemma. □

Randomness Mapping Argument: Defining the Map Since we have finished the argument that the domain of the map covers overwhelmingly many Σ_1' executions, we are now ready to define the map itself. We introduce some necessary notions first.

Basic Notions For any distinguisher $\mathbf{D}$, we define a distinguisher $\overline{D}$ which runs $\mathbf{D}$ and then emulates a call to $EvalForward(x_R, x_L, x_R \oplus F(1, x_L) \oplus k, 0, 20)$ ($EvalBackward(y_L \oplus F(21, y_R) \oplus k, y_R, y_L, 20, 0)$, resp.) for all queries $\widetilde{E}.Enc(k, (x_L, x_R))$ ($\widetilde{E}.Dec(k, (y_L, y_R))$, resp.) made by $\mathbf{D}$ during the execution, and outputs the output of $\mathbf{D}$. Clearly when $\mathbf{D}$ makes no more than q queries, $\overline{D}$ makes no more than $22q$ queries, and has exactly the same advantage as $\mathbf{D}$ in distinguishing Σ_1' and Σ_2. We call $\overline{D}$ the distinguisher which completes all chains corresponding to $\mathbf{D}$. Then, with respect to a fixed $\mathbf{D}$ and the corresponding $\overline{D}$, we introduce the following notions. Some of them are similar to those in [2].

- a tuple of random tapes $(\eta, \varphi) = (\eta, (\varphi_1, \ldots, \varphi_{21}))$ for Σ_1' is called a Σ_1'-*tuple*;
- a tuple of random tapes $f = (f_1, \ldots, f_{21})$ for Σ_2 is called Σ_2-*tuple*;
- R_1' (R_2, resp.) is the set of all Σ_1'-*tuples* (Σ_2-*tuples*, resp.);
- a Σ_1'-*tuple* is called *good* if during the execution $\overline{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, $\widetilde{S}(\varphi)$ does not overwrite any entry[12];

[12] Note that **BadHit** does happen during $\overline{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ is only a necessary condition for (η, φ) to be not good.

- $R_1^{good} \subseteq R_1^{'}$ is the set of all good $\Sigma_1^{'}$-*tuples* with respect to $\overline{D}$;
- *partial* $\Sigma_1^{'}$-*tuple*: random tapes obtained by arbitrarily setting some entries $\varphi_i(x)$ or entry pairs $(\eta(+,k,x), \eta(-,k,\eta(+,k,x)))$ to $\perp$ in a $\Sigma_1^{'}$-*tuple* (η,φ), while keeping the property $\eta(\delta,k,z) = z^{'} \neq \perp$ iff. $\eta(\overline{\delta},k,z^{'}) = z \neq \perp$;
- $R_1^{partial}$: the set of all partial $\Sigma_1^{'}$-tuples;
- *partial* Σ_2-*tuple*: obtained by arbitrarily setting some entries $f_i(x)$ to $\perp$ in a Σ_2-*tuple* f. $R_2^{partial}$ is the set of all partial Σ_2-tuples;
- For $f \in R_2^{partial}$, denote the number of pairs (i,x) s.t. $f_i(x) \neq \perp$ by $|f|$;
- *footprint* of a random $\Sigma_1^{'}$-tuple (η,φ): the partial tuple obtained by
  (i) for any $i \in \{1,\ldots,21\}$ and any $x \in \{0,1\}^n$, setting $\varphi_i(x)$ to $\perp$, if $\varphi_i(x)$ is not accessed during $\overline{D}^{\Sigma_1^{'}(\widetilde{E}(\eta),\mathbf{S}(\varphi))}$;
  (ii) for any $z \in \{0,1\}^{2n}$ and any $k \in \{0,1\}^n$, setting both $\eta(+,k,z)$ and $\eta(-,k,\eta(+,k,z))$ to $\perp$, if neither $\eta(+,k,z)$ nor $\eta(-,k,\eta(+,k,z))$ is accessed during $\overline{D}^{\Sigma_1^{'}(\widetilde{E}(\eta),\mathbf{S}(\varphi))}$;
- the set of $\Sigma_1^{'}$-footprints is denoted by R_1^{foot}, and the set of footprints of good tuples (η,φ) is denoted by R_1^{good-f};
- $FootPrint((\eta,\varphi),\overline{D})$ denotes the footprint of (η,φ) with respect to $\overline{D}$;

For $f = (f_1,\ldots,f_{21}) \in R_2$ and $u = (f_1',\ldots,f_{21}') \in R_2^{partial}$, denote by $f \cong u$ the fact that for any $i \in \{1,\ldots,21\}$ and any $x \in \{0,1\}^n$, if $f_i'(x) \neq \perp$, then $f_i(x) = f_i'(x)$. Briefly speaking, $f \cong u$ means that f agrees with u on all the non-empty entries.

Defining the Map The map is

$$\tau : R_1^{good-f} \to R_2^{partial}.$$

Let $\alpha := (\eta,\varphi) \in R_1^{good-f}$. We define $\tau(\alpha) = f = (f_1,\ldots,f_{21})$ according to the tables $(\widetilde{E}(\eta).E, G_1,\ldots,G_{21})$ standing at the end of the execution of $\overline{D}^{\Sigma_1^{'}(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$: for all $i \in \{1,\ldots,21\}$ and $x \in G_i$, $f_i(x) := G_i(x)$; for all $i \in \{1,\ldots,21\}$ and $x \notin G_i$, $f_i(x) := \perp$. Since α is a footprint, $\overline{D}^{\Sigma_1^{'}(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ is well-defined, and τ is also well-defined. Denote the range of τ by $\tau(R_1^{good-f})$.

Completing the RRMA With respect to $\overline{D}$, we have the following lemmas.

Lemma 19. *Consider $(\eta,\varphi) \in R_1^{good}$, and suppose that $\widetilde{E}(\eta).Enc(k,x)$, resp. $\widetilde{E}(\eta).Dec(k,y)$ is queried during $\overline{D}^{\Sigma_1^{'}(\eta,\varphi)}$. Then, at the end of the execution $\overline{D}^{\Sigma_1^{'}(\eta,\varphi)}$, it holds that $\widetilde{E}(\eta).E(+,k,x) = (val_{22}^{+}(x_R,x_L,x',0), val_{21}^{+}(x_R,x_L,x',0))$, resp. $\widetilde{E}(\eta).E(-,k,y) = (val_1^{-}(y',y_R,y_L,20), val_0^{-}(y',y_R,y_L,20))$, where $x' = x_R \oplus G_1(x_L) \oplus k$ and $y' = y_L \oplus G_{21}(y_R) \oplus k$.*

Proof. $\widetilde{S}(\varphi)$ queries $\widetilde{E}(\eta)$ only when it is completing a chain, hence if the query $\widetilde{E}(\eta).Enc(k,x)$ is made by $\widetilde{S}(\varphi)$, then the equality holds right after the completion of the chain, and will keep holding till the end since $(\eta,\varphi) \in R_1^{good}$ and no entry will be overwritten during $\overline{D}^{\Sigma_1'(\eta,\varphi)}$. On the other hand, if the query is issued by $\overline{D}$, then since $\overline{D}$ completes all chains, it must emulate a call to $EvalForward(x_R, x_L, x', 0, 20)$ where $x' = x_R \oplus F(1, x_L) \oplus k$, and at some point it must query $F(10, val_{10}^+(x_R, x_L, x', 0))$, $F(11, val_{11}^+(x_R, x_L, x', 0))$ and $F(12, val_{12}^+(x_R, x_L, x', 0))$, among which the last one will trigger the chain completion. After the completion, the equality holds, and will also keep holding till the end. The case of $\widetilde{E}(\eta).Dec(k,y)$ is similar by symmetry. □

Lemma 20. *For any $\alpha \in R_1^{good-f}$, the transcript of the sequence of queries and answers of $\overline{D}$ in $\overline{D}^{\Sigma_1'(\alpha)}$ is the same as the transcript of the sequence of queries and answers of $\overline{D}$ in $\overline{D}^{\Sigma_2(\tau(\alpha))}$, and $\overline{D}^{\Sigma_1'(\alpha)} = \overline{D}^{\Sigma_2(\tau(\alpha))}$.*

Proof. Let $\alpha = (\eta, \varphi) \in R_1^{good-f}$ and let $\beta = \tau(\alpha) = (f_1, \ldots, f_{21}) \in R_2^{partial}$. By definition of τ, $f_1, \ldots, f_{21}$ are copies of the tables $G_1, \ldots, G_{21}$ standing at the **end** of the execution $\overline{D}^{\Sigma_1'(\alpha)}$. To show the transcripts to be the same, we proceed by induction on the sequence of queries of $\overline{D}$ to $(\widetilde{E}(\eta), \widetilde{S}(\varphi))$ or $(SKAF_{21}^*, \mathbf{F}(\beta))$. Assume that the sequence of queries and answers is the same in the two systems up to some point in the executions, and consider the next query issued by $\overline{D}$. Since $\overline{D}$ is deterministic, this query is the same in both systems. We now argue the $\overline{D}$ receives the same answer. For the following cases:

- If the query is to $F(i,x)$, then since $\alpha \in R_1^{good-f}$, $\widetilde{S}(\varphi).F(i,x) = G_i(x) = f_i(x)$ holds, i.e. the answers received by $\overline{D}$ are the same in both the systems;
- If the query is to $Enc(k,(x_L,x_R))$, then since $\alpha \in R_1^{good-f}$, with respect to $\{G_i\}$, $\widetilde{E}(\eta).Enc(k,x) = (val_{22}^+(x_R,x_L,x',0), val_{21}^+(x_R,x_L,x',0))$ follows from Lemma 19 (where $x' = x_R \oplus G_1(x_L) \oplus k$). Since all the values in $\{G_i\}$ are transferred to β by τ, $SKAF_{21}^*.Enc(k,x) = (val_{22}^+(x_R,x_L,x'',0), val_{21}^+(x_R,x_L,x'',0)) = \widetilde{E}.Enc(k,x)$ holds where $x'' = x_R \oplus f_1(x_L) \oplus k = x'$; hence in this case, the answers are the same;
- The case when the procedure field is Dec is similar to Enc by symmetry;

Therefore the transcript of the sequence of queries and answers of $\overline{D}$ in the two systems are the same, and $\overline{D}^{\Sigma_1'(\alpha)} = \overline{D}^{\Sigma_2(\tau(\alpha))}$ since $\overline{D}$ is deterministic. □

Lemma 20 also shows that for any $v \in \tau(R_1^{good-f})$, $\overline{D}^{\Sigma_2(v)}$ is well-defined, i.e. during $\overline{D}^{\Sigma_2(v)}$, $\mathbf{F}(v)$ will not access the empty entries in v.

Lemma 21. *$\tau : R_1^{good-f} \to R_2^{partial}$ is one-to-one.*

Proof. Towards a contradiction, assume that there exist $\alpha, \alpha' \in R_1^{good-f}$ s.t. $\alpha \neq \alpha'$ while $\tau(\alpha) = \tau(\alpha') = (f_1, \ldots, f_{21})$. Let $\alpha = (\eta, \varphi)$, $\alpha' = (\eta', \varphi')$, $\{G_i\}$

be the tables standing at the end of $\overline{D}^{\Sigma_1^{'}(\alpha)}$, and $\{G_i^{'}\}$ be those standing at the end of $\overline{D}^{\Sigma_1^{'}(\alpha^{'})}$. By the definition of τ, $\tau(\alpha) = (f_1, \ldots, f_{21})$ copies the tables of $\widetilde{S}$; hence $\{G_i\}$ and $\{G_i^{'}\}$ are exactly the same, by $\tau(\alpha^{'}) = \tau(\alpha)$. Denote this fact by $\{G_i\} \equiv \{G_i^{'}\}$.

We argue the transcripts (see Sect. 6.3 for the formal definition) of queries and answers of $\overline{D} \cup \widetilde{S}$ in $\Sigma_1^{'}(\alpha)$ and $\Sigma_1^{'}(\alpha')$ to be the same, which implies that the zones of α and $\alpha^{'}$ accessed during the executions are exactly the same, and $\alpha = \alpha^{'}$ by the definition of footprint. Similarly to Lemma 4, assume that during the two executions, the transcripts attained so far are the same. Since both $\overline{D}$ and $\widetilde{S}$ are deterministic, the next query is the same. Consider this query:

- If the query is to φ, *Enc*, or *Dec*, following the same line as the proof of Lemma 20 and by $\{G_i\} \equiv \{G_i^{'}\}$, the answers obtained are equal.
- When the query is to *Check*, the answers depend on the tables $\widetilde{E}(\eta).E$ and $\widetilde{E}(\eta^{'}).E$. Since the transcripts attained so far are assumed to be the same, the two tables $\widetilde{E}(\eta).E$ and $\widetilde{E}(\eta^{'}).E$ have exactly the same contents at this point; this implies the return values of the two corresponding check calls to be equal, i.e. the answers obtained are equal.

These conclude the proof. □

The following lemma links the randomness brought in by η tape accessing actions in $\Sigma_1^{'}$ to that brought in by certain f tape accessing actions in Σ_2.

Lemma 22. *Consider* $(\eta, \varphi) \in R_1^{good-f}$. *Then during the execution* $\overline{D}^{\Sigma_1^{'}(\widetilde{E}(\eta), \widetilde{S}(\varphi))}$, *the number of* η *tape accessing action equals the number of calls to Adapt.*

Proof. During such an execution $\overline{D}^{\Sigma_1^{'}(\widetilde{E}(\eta), \widetilde{S}(\varphi))}$:

- Each call to *Adapt* corresponds to a distinct η tape accessing action, since $\widetilde{E}(\eta)$ must be queried by $\widetilde{S}(\varphi)$ either during the procedure *EvalForward* or during *EvalBackward*. These actions must be distinct since otherwise different calls to *Adapt* correspond to a same plaintext-ciphertext-key tuple of $\widetilde{E}(\eta)$, which implies some entries to be overwritten.
- Each η tape accessing action corresponds to a call to *Adapt*: if the action is triggered by query (to $\widetilde{E}(\eta)$) issued by $\widetilde{S}(\varphi)$, then clearly $\widetilde{S}(\varphi)$ is completing a chain and will call *Adapt* soon; if the action is triggered by query issued by $\overline{D}$, then $\overline{D}$ will query the corresponding $SKAF_{21}^*$ computation path, and will reach the point when the 3 corresponding values x_{10}, x_{11}, and x_{12} are all in the G tables. After this point a chain equivalent to $(x_{10}, x_{11}, x_{12}, 10)$ must have been enqueued and has been adapted accordingly. □

Then we bound the probability ratio between partial tuples linked by τ.

Lemma 23. *Consider a fixed q'-query distinguisher $\overline{D}$. Let $u \in R_1^{good-f}$. Then for any randomly chosen tapes (η, φ) and f the following holds:*

$$1 - \frac{(10{q'}^3)^2}{2^{2n}} \leq \frac{Pr[f \cong \tau(u)]}{Pr[FootPrint((\eta,\varphi),\overline{D}) = u]} \leq 1$$

Proof. Clearly $FootPrint((\eta,\varphi),\overline{D}) = u$ holds if $\overline{D}$ gets the same values for each tape accessing action in $\overline{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$ and $\overline{D}^{\Sigma_1'(u)}$. Let $u = (\eta^*, \varphi^*)$, and suppose that φ^* tape was accessed i times while η^* tape was accessed j times during $\overline{D}^{\Sigma_1'(u)}$. Since $u \in R_1^{good-f}$, by Lemma 22 each η^* tape accessing action corresponds to one call to *Adapt*/two n-bit "adapted" values in $\{G_i\}$. Hence at the end of $\overline{D}^{\Sigma_1'(u)}$, $\{G_i\}$ contains $i + 2j$ entries; this implies $|\tau(u)| = i + 2j$, and $Pr[f \cong \tau(u)] = (2^{-n})^{i+2j}$.

On the other hand, in $\overline{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, for each η tape accessing action, we have $Pr[\eta(\delta,x,k) = \eta^*(\delta,x,k)] \in [\frac{1}{2^{2n}-j}, \frac{1}{2^{2n}}]$ while for each φ tape accessing action we have $Pr[\varphi(x) = \varphi^*(x)] = 2^{-n}$. Hence $Pr[f \cong \tau(u)] \leq Pr[FootPrint((\eta,\varphi),\overline{D}) = u]$, and we have the following lower bound:

$$\frac{Pr[f \cong \tau(u)]}{Pr[FootPrint((\eta,\varphi),\overline{D}) = u]} \geq \frac{(2^{-n})^{i+2j}}{(2^{-n})^i \cdot (\frac{1}{2^{2n}-j})^j} = (\frac{2^{2n}-j}{2^{2n}})^j$$

By Lemma 2 we have $j \leq 10{q'}^3$, hence the claim holds. □

Lemma 24. $\sum_{v \in R_2 \wedge \overline{D}^{\Sigma_2(v)}=1} Pr_f[f = v] = \sum_{u \in \tau(R_1^{good-f}) \wedge \overline{D}^{\Sigma_2(u)}=1} Pr_f[f \cong u]$.

Proof. To show the claim, we show that for each $f \in R_2$:

(i) $\exists v \in \tau(R_1^{good-f})$ such that $f \cong v$;
(ii) there exists at most one $v \in \tau(R_1^{good-f})$ such that $f \cong v$;

For claim (i), consider the tape η which defines the input and output table of the cipher $SKAF_{21}^{*\mathbf{F}(f)}$. Then the preimage of v can be generated during the execution $\overline{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(f))}$ (clearly $(\eta, f) \in R_1^{good}$, since the values defined by η and f are consistent).

For claim (ii), assume otherwise, i.e. $\exists u, u'$ s.t. $u \neq u'$, $f \cong u$ and $f \cong u'$. Let $f = (f_1^*, \ldots, f_{21}^*)$, $u = (f_1, \ldots, f_{21})$, $u' = (f_1', \ldots, f_{21}')$, and let the preimages of u and u' be $\alpha = (\eta, \varphi)$ and $\alpha' = (\eta', \varphi')$ respectively. Consider the two executions $\overline{D}^{\Sigma_1'(\alpha)}$ and $\overline{D}^{\Sigma_1'(\alpha')}$, and let $\{G_i\}$ and $\{G_i'\}$ be the tables standing at the end of them respectively. We show the two executions to be the same: for any query $F(i,x)$, the answers in them two are equal because, if the query $F(i,x)$ is made, then $x \in G_i$ which implies $f_i(x), f_i'(x) \neq \perp$, and $\widetilde{S}(\varphi).F(i,x) = G_i(x) = f_i(x) = f_i^*(x) = f_i'(x) = G_i'(x) = \widetilde{S}(\varphi').F(i,x)$. Following the same line as in the proof

of Lemma 21 we see all the queries $\overline{D} \cup \widetilde{S}$ in the two executions get exactly the same answers and the transcripts of queries and answers of $\overline{D} \cup \widetilde{S}$ in $\Sigma_1'(\alpha)$ and $\Sigma_1'(\alpha')$ are the same, so that $\alpha = \alpha'$, and $u = u'$ follows from τ being one-to-one (Lemma 21), a contradiction. Then claim (ii) holds, and we further have:

$$\begin{aligned}\sum_{v \in R_2 \wedge \overline{D}^{\Sigma_2(v)}=1} Pr_f[f = v] &= \sum_{v \in R_2 \wedge \overline{D}^{\Sigma_2(v)}=1 \wedge \exists u \in \tau(R_1^{good-f}) \ s.t. \ v \cong u} Pr_f[f = v] \\ &= \sum_{u \in \tau(R_1^{good-f}) \wedge \overline{D}^{\Sigma_2(u)}=1} \left[\sum_{v \in R_2 \wedge v \cong u} Pr_f[f = v] \right] \\ &= \sum_{u \in \tau(R_1^{good-f}) \wedge \overline{D}^{\Sigma_2(u)}=1} Pr_f[f \cong u]\end{aligned}$$

as claimed. □

Now we are ready to bound the advantage of distinguishing $\Sigma_1^{'}$ and Σ_2:

Lemma 25. *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. Let $\overline{D}$ be the distinguisher which completes all chains corresponding to $\mathbf{D}$. Then $\overline{D}$ makes at most $22q$ queries, and by Lemma 7, we have[13]

$$\begin{aligned}Pr[\overline{D}^{\Sigma_1^{'}} = 1] &\leq Pr_{(\eta,\varphi)}[(\eta, \varphi) \in R_1^{good} \wedge \overline{D}^{\Sigma_1{}'(\eta,\varphi)} = 1] + Pr[(\eta, \varphi) \notin R_1^{good}] \\ &\leq \sum_{u \in R_1^{good-f} \wedge \overline{D}^{\Sigma_1^{'}(u)}=1} Pr_{(\eta,\varphi)}[FootPrint((\eta, \varphi), \overline{D}) = u] + \frac{2^{88} \cdot (22q)^{30}}{2^n}\end{aligned}$$

By Lemma 24 we have

$$Pr[\overline{D}^{\Sigma_2} = 1] = \sum_{v \in \tau(R_2^{good-f}) \wedge \overline{D}^{\Sigma_2^{'}(v)}=1} Pr_f[f \cong v]$$

Then, by Lemma 21 (τ is one-to-one) and Lemma 23, we have

$$\begin{aligned}&(1 - \frac{(10(22q)^3)^2}{2^{2n}}) \cdot Pr[(\eta, \varphi) \in R_1^{good} \wedge \overline{D}^{\Sigma_1{}'} = 1] \\ \leq &Pr[\overline{D}^{\Sigma_2} = 1] \leq Pr[(\eta, \varphi) \in R_1^{good} \wedge \overline{D}^{\Sigma_1{}'} = 1]\end{aligned}$$

Thus

$$\left| Pr[\mathbf{D}^{\Sigma_1^{'}} = 1] - Pr[\mathbf{D}^{\Sigma_2} = 1] \right| \leq \frac{2^{88} \cdot (22q)^{30}}{2^n} + \frac{100(22q)^6}{2^{2n}}.$$

Since $\overline{D}$ has exactly the same advantage as $\mathbf{D}$, we reach the conclusion. □

[13] $Pr[(\eta, \varphi) \notin R_1^{good}] \leq Pr[\textbf{BadHit}$ *happens during* $\overline{D}^{\Sigma_1^{'}(\eta,\varphi)}] \leq \frac{2^{88} \cdot (22q)^{30}}{2^n}$

7 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 cannot be published 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. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2014, to appear)
2. Andreeva, E., Bogdanov, A., Dodis, Y., Mennink, B., Steinberger, J.: On the indifferentiability of key-alternating ciphers. In: Canetti, R., Garay, J. (eds.) Advances in Cryptology - CRYPTO 2013, Lecture Notes in Computer Science, vol. 8042, pp. 531–550. Springer Berlin Heidelberg (2013)
3. Andreeva, E., Bogdanov, A., Mennink, B.: Towards understanding the known-key security of block ciphers. In: Moriai, S. (ed.) Fast Software Encryption, pp. 348–366. Lecture Notes in Computer Science, Springer Berlin 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. Lecture Notes in Computer Science, Springer Berlin 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.) Fast Software Encryption, pp. 3–18. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2014)
8. Biryukov, A., Roy, A., Velichkov, V.: Differential analysis of block ciphers simon and speck. In: Fast Software Encryption 2014. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2014, to appear)
9. Black, J.: The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In: Robshaw, M. (ed.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 4047, pp. 328–340. Springer Berlin Heidelberg (2006)
10. Bouillaguet, C., Dunkelman, O., Leurent, G., Fouque, P.A.: Another look at complementation properties. In: Hong, S., Iwata, T. (eds.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 6147, pp. 347–364. Springer Berlin 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.) Advances in Cryptology – ASIACRYPT 2014, Lecture Notes in Computer Science, vol. 8873, pp. 179–199. Springer Berlin Heidelberg (2014)
12. Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited. J. ACM 51(4), 557–594 (Jul 2004)

13. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-damgård revisited: How to construct a hash function. In: Shoup, V. (ed.) Advances in Cryptology – CRYPTO 2005, Lecture Notes in Computer Science, vol. 3621, pp. 430–448. Springer Berlin 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.) Advances in Cryptology - CRYPTO 2008, Lecture Notes in Computer Science, vol. 5157, pp. 1–20. Springer Berlin Heidelberg (2008)
15. Demay, G., Gaži, P., Hirt, M., Maurer, U.: Resource-restricted indifferentiability. In: Johansson, T., Nguyen, P. (eds.) Advances in Cryptology – EUROCRYPT 2013, Lecture Notes in Computer Science, vol. 7881, pp. 664–683. Springer Berlin Heidelberg (2013)
16. Guo, J., Jean, J., Nikolić, I., Sasaki, Y.: Meet-in-the-middle attacks on generic feistel constructions. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014, Lecture Notes in Computer Science, vol. 8873, pp. 458–477. Springer Berlin Heidelberg (2014)
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. pp. 89–98. STOC '11, ACM, New York, NY, USA (2011)
18. Isobe, T., Shibutani, K.: Generic key recovery attack on feistel scheme. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology – ASIACRYPT 2013, Lecture Notes in Computer Science, vol. 8269, pp. 464–485. Springer Berlin Heidelberg (2013)
19. Knudsen, L., Rijmen, V.: Known-key distinguishers for some block ciphers. In: Kurosawa, K. (ed.) Advances in Cryptology - ASIACRYPT 2007, Lecture Notes in Computer Science, vol. 4833, pp. 315–324. Springer Berlin Heidelberg (2007)
20. Lampe, R., Seurin, Y.: How to construct an ideal cipher from a small set of public permutations. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology – ASIACRYPT 2013, Lecture Notes in Computer Science, vol. 8269, pp. 444–463. Springer Berlin Heidelberg (2013)
21. Lampe, R., Seurin, Y.: Security analysis of key-alternating feistel ciphers. In: Fast Software Encryption 2014. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2014, to appear)
22. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal on Computing 17(2), 373–386 (1988)
23. Mandal, A., Patarin, J., Seurin, Y.: On the public indifferentiability and correlation intractability of the 6-round feistel construction. In: Cramer, R. (ed.) Theory of Cryptography, Lecture Notes in Computer Science, vol. 7194, pp. 285–302. Springer Berlin Heidelberg (2012)
24. Maurer, U., Pietrzak, K.: The security of many-round luby-rackoff pseudo-random permutations. In: Biham, E. (ed.) Advances in Cryptology – EUROCRYPT 2003, Lecture Notes in Computer Science, vol. 2656, pp. 544–561. Springer Berlin Heidelberg (2003)
25. Maurer, U., Renner, R., Holenstein, C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Naor, M. (ed.) Theory of Cryptography, Lecture Notes in Computer Science, vol. 2951, pp. 21–39. Springer Berlin Heidelberg (2004)
26. Patarin, J.: Pseudorandom permutations based on the d.e.s. scheme. In: Cohen, G., Charpin, P. (eds.) EUROCODE '90, Lecture Notes in Computer Science, vol. 514, pp. 193–204. Springer Berlin Heidelberg (1991)

27. Patarin, J.: Security of random feistel schemes with 5 or more rounds. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004, Lecture Notes in Computer Science, vol. 3152, pp. 106–122. Springer Berlin Heidelberg (2004)
28. Ristenpart, T., Shacham, H., Shrimpton, T.: Careful with composition: Limitations of the indifferentiability framework. In: Paterson, K. (ed.) Advances in Cryptology – EUROCRYPT 2011, Lecture Notes in Computer Science, vol. 6632, pp. 487–506. Springer Berlin Heidelberg (2011)
29. Sasaki, Y., Emami, S., Hong, D., Kumar, A.: Improved known-key distinguishers on feistel-sp ciphers and application to camellia. In: Susilo, W., Mu, Y., Seberry, J. (eds.) Information Security and Privacy, Lecture Notes in Computer Science, vol. 7372, pp. 87–100. Springer Berlin Heidelberg (2012)
30. Sasaki, Y., Yasuda, K.: Known-key distinguishers on 11-round feistel and collision attacks on its hashing modes. In: Joux, A. (ed.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 6733, pp. 397–415. Springer Berlin Heidelberg (2011)
31. Seurin, Y.: Primitives et protocoles cryptographiques àsécurité prouvée. Ph.D. thesis, PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, France, 2009 (2009)
32. 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.) Advances in Cryptology – ASIACRYPT 2014, Lecture Notes in Computer Science, vol. 8873, pp. 158–178. Springer Berlin Heidelberg (2014)
33. Todo, Y.: Upper bounds for the security of several feistel networks. In: Boyd, C., Simpson, L. (eds.) Information Security and Privacy, Lecture Notes in Computer Science, vol. 7959, pp. 302–317. Springer Berlin Heidelberg (2013)

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 [17,20] – 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, we are not clear whether 17-round single-key $SKAF^*_{17}$ can achieve indifferentiability or not. In fact, we *think* $SKAF^*_{17}$ may be proven to be indifferentiable by some much more complex analysis.

B One Round of SIMON

Following the notation convention in the specification of SIMON [6], for $k \in GF(2)^n$, the key-dependent SIMON2n round function is the two-stage Feistel map $R_k : GF(2)^n \times GF(2)^n \to GF(2)^n \times GF(2)^n$ defined by

$$R_k(x, y) = (y \oplus f(x) \oplus k, x)$$

where $f(x) = (Sx \& S^8 x) \oplus S^2 x$ and k is the round key. The notation $S^j x$ denotes left circular shifting x by j bits. It is illustrated in Fig. 4 (left) (which is Figure 3.1 of the specification of SIMON [6]).

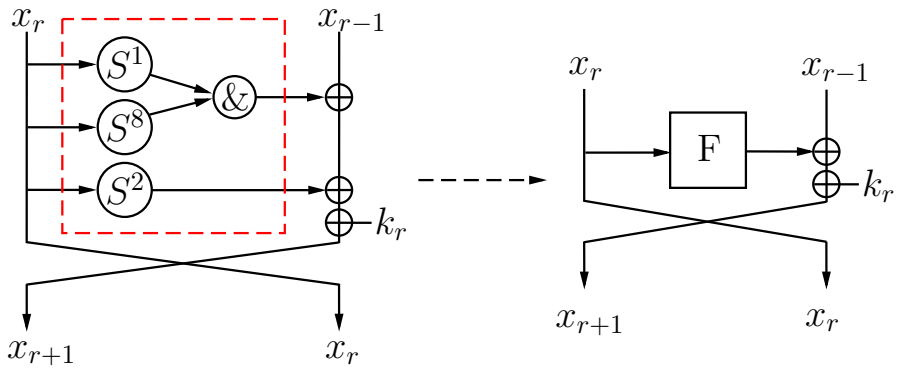

Fig. 4. One round of the SIMON family, and to the construction KAF^*. In the figure, S^i denotes the operation of left circular shifting by i bits.

C On the Key-undefined Chains

As already mentioned in the main part, the influences of the key-undefined chains are limited to a very small extent. This is formally captured by the lemma in this section: when a chain turns from key-undefined to key-defined, the two "ends" of the chain must be out of the tables.

Lemma 26. *Consider a random tape accessing and subsequent entry setting action $G_i(x_i) := \varphi_i(x_i)$ in a good execution $\mathbf{D}^{\Sigma_1'(\widetilde{E}(\eta),\widetilde{S}(\varphi))}$, where $i \in \{2,\ldots,20\}$. If a chain C is key-undefined before this action, while turns to table-defined after this action, then this action can only change $val^-_{i-2}(C)$ and $val^+_{i+2}(C)$ from $\perp$ to non-empty values.*

Proof. By the assumptions of this lemma, before the action we must have $C = (x_{i-1}, x_i, x_{i+1})$ and $x_{i-1} \in \mathcal{H} \wedge x_{i+1} \in \mathcal{H}$.

Clearly, $val^-_{i-2}(C)$ and $val^+_{i+2}(C)$ are changed from $\perp$ to non-empty values by this action. We argue that the changes are limited to these two values to establish the claim. We first argue that $val^+_{i+3}(C) = \perp$ (when $i \leq 19$) or $val^+_0(C) = val^+_1(C) = \perp$ (when $i = 20$) after the action; the reasoning for $val^-_{i-3}(C) = \perp$ (when $i \geq 3$) or $val^-_{22}(C) = \perp$ (when $i = 2$) after the action is similar by symmetry. Towards a contradiction, assume that $val^+_{i+3}(C) \neq \perp$ (when $i \leq 19$) or $val^+_0(C) \neq \perp \vee val^+_1(C) \neq \perp$ (when $i = 20$) after the action. Then:

- when $i \leq 19$, $val^+_{i+3}(C) \neq \perp$ implies $val^+_{i+2}(C) \in G_{i+2}$ after the action; hence **BadHit** happens during the action $G_i(x_i) := \varphi_i(x_i)$, namely $\varphi_i(x_i) = x_i \oplus G_{i+1}(x_{i+1}) \oplus x_{i-1} \oplus x_{i+1} \oplus val^+_{i+2}(C)$;
- when $i = 20$, $val^+_0(C) \neq \perp \vee val^+_1(C) \neq \perp$ implies $(-, k, (val^+_{22}(C), x_{21})) \in E$ after the action, for some $k \in \{0,1\}^n$; hence **BadHit** happens during the action $G_{20}(x_{20}) := \varphi_{20}(x_{20})$, namely $\varphi_{20}(x_{20}) = k \oplus x_{19} \oplus x_{21}$;

Therefore $val^+_{i+3}(C) = \perp$ (when $i \leq 19$) or $val^+_0(C) = val^+_1(C) = \perp$ (when $i = 20$) after the action. These establish the claim. □