Indifferentiability of 8-Round Feistel Networks

Yuanxi Dai and John Steinberger

Institute for Interdisciplinary Information Sciences, Tsinghua University, Beijing.
dyx13@mails.tsinghua.edu.cn, jpsteinb@gmail.com

Abstract. We prove that a balanced 8-round Feistel network is indifferentiable from a random permutation, improving on previous 10-round results by Dachman-Soled et al. and Dai et al. Our simulator achieves security $O(q^8/2^n)$, similarly to the security of Dai et al. For further comparison, Dachman-Soled et al. achieve security $O(q^{12}/2^n)$, while the original 14-round simulator of Holenstein et al. achieves security $O(q^{10}/2^n)$.

Keywords. Feistel network, block ciphers

1 Introduction

For many cryptographic protocols the only known analyses are in a so-called *ideal primitive model*. In such a model, a cryptographic component is replaced by an idealized information-theoretic counterpart (e.g., a random oracle takes the part of a hash function, or an ideal cipher substitutes for a concrete blockcipher such as AES) and security bounds are given as functions of the query complexity of an information-theoretic adversary with oracle access to the idealized primitive. Early uses of such ideal models include Winternitz [33], Fiat and Shamir [19] (see proof in [28]) and Bellare and Rogaway [2], with such analyses rapidly proliferating after the latter paper.

Given the popularity of such analyses a natural question that arises is to determine the relative "power" of different classes of primitives and, more precisely, whether one class of primitives can be used to "implement" another. E.g., is a random function always sufficient to implement an ideal cipher, in security games where oracle access to the ideal cipher/random function is granted to all parties? The challenge of such a question is partly definitional, since the different primitives have syntactically distinct interfaces. (Indeed, it seems that it was not immediately obvious to researchers that such a question made sense at all [7].)

A sensible definitional framework, however, was proposed by Maurer et al. [23], who introduce a simulation-based notion of *indifferentiability*. This framework allows to meaningfully discuss the instantiation of one ideal primitive by a syntactically different primitive, and to compose such results. (Similar simulation-based definitions appear in [4, 5, 26, 27].) Coron et al. [7] are early adopters of the framework, and give additional insights.

Informally, given ideal primitives Z and Q, a construction C^Q (where C is some stateless algorithm making queries to Q) is *indifferentiable* from Z if there exists a simulator S (a stateful, randomized algorithm) with oracle access to Z

such that the pair (C^Q, Q) is statistically indistinguishable from the pair (Z, S^Z). The more efficient the simulator, the lower its query complexity, and the better the statistical indistinguishability, the more practically meaningful the result.

The present paper focuses on the natural question of implementing a permutation from one or more random functions (a small number of distinct random functions can be emulated by a single random function with a slightly larger domain) such that the resulting construction is indifferentiable from a random permutation. This means building a permutation $C : \{0,1\}^{m(n)} \to \{0,1\}^{m(n)}$ where

$$C = C[F_1, \ldots, F_r]$$

depends on a small collection of random functions $F_1, \ldots, F_r : \{0,1\}^n \to \{0,1\}^n$ such that the vector of $r+1$ oracles

$$(C[F_1, \ldots, F_r], F_1, \ldots, F_r)$$

is statistically indistinguishable from a pair

$$(Z, S^Z)$$

where $Z : \{0,1\}^{m(n)} \to \{0,1\}^{m(n)}$ is a random permutation from $m(n)$ bits to $m(n)$ bits, for some efficient simulator S. Thus, in this case, the simulator emulates the random functions $F_1, \ldots, F_r$, and it must use its oracle access to Z to invent answers that make the (fake) random functions $F_1, \ldots, F_r$ look "compatible" with Z, as if Z where really $C[F_1, \ldots, F_r]$. (On the other hand, the simulator does not know what queries the distinguisher might be making to Z.) Here $m(n)$ is polynomially related to n: concretely, the current paper discusses a construction with $m = 2n$.

The construction $C[F_1, \ldots, F_r]$ that we consider in this paper, and as considered in previous papers with the same goal as ours (see discussion below), is an r-round (balanced, unkeyed) *Feistel network*. To wit, given arbitrary functions $F_1, \ldots, F_r : \{0,1\}^n \to \{0,1\}^n$, we define a permutation

$$C[F_1, \ldots, F_r] : \{0,1\}^{2n} \to \{0,1\}^{2n}$$

by the following application: for an input $(x_0, x_1) \in \{0,1\}^{2n}$, values $x_2, \ldots, x_{r+1}$ are defined by setting

$$x_{i+1} = x_{i-1} \oplus F_i(x_i) \tag{1}$$

for $i = 1, \ldots, r$; then $(x_r, x_{r+1}) \in \{0,1\}^{2n}$ is the output of C on input (x_0, x_1). One can observe that C is a permutation since x_{i-1} can be computed from x_i and x_{i+1}, by (1). The value r is the number of *rounds* of the Feistel network. (See, e.g., Fig. 1.)

The question of showing that a Feistel network with a sufficient number of rounds is indifferentiable from a random permutation already has a growing history. Coron, Patarin and Seurin [9] show that an r-round Feistel network cannot be indifferentiable from a random permutation for $r \leq 5$, due to explicit attacks. They also give a proof that indifferentiability is achieved at $r = 6$, but this latter result was found to have a serious flaw by Holenstein et al. [20], who could only prove, as a replacement, that indifferentiability is achieved at $r = 14$ rounds. At the same time, Holenstein et al. found a flaw in the proof of indifferentiability of a 10-round simulator of Seurin's [31] (a simplified alternative to the 6-round simulator of [9]), after which Seurin himself found an explicit attack against his own simulator, showing that the proof could not be patched [32]. More recently, Dachman-Soled et al. [10] and the authors of the present paper [11] have presented independent indifferentiability proofs at 10 rounds.

In [11] we achieve slightly better security than other proofs ($O(q^8/2^n)$, compared to $O(q^{10}/2^n)$ for Holenstein et al. and $O(q^{12}/2^n)$ for Dachman-Soled et al.), and their work also introduces an interesting "last-in-first-out" simulator paradigm. In fact, the simulator of [11] is essentially Seurin's (flawed) 10-round simulator, only with "first-in-first-out" path completion replaced by "last-in-first-out" path completion. This change, as it turns out, is sufficient to repair the flaw discovered by Holenstein et al. [20].

In the current work we prove that an 8-round Feistel network is indifferentiable from a random permutation. The security, query complexity and runtime of our 8-round simulator are $O(q^8/2^n)$, $O(q^4)$ and $O(q^4)$ respectively, just like our previous 10-round simulator [11]. (The query complexity of previous simulators of Dachman-Soled et al. and Holenstein et al. can apparently be reduced to $O(q^4)$ as well with suitable optimizations [11], though higher numbers are quoted in the original papers.) In fact our work closely follows the ideas [11], and is obtained by making a number of small optimizations to that simulator in order to reduce it to 8 rounds. It remains open whether 6 or 7 rounds might suffice for indifferentiability.

Concerning our optimizations, more specifically, in [11, 20, 31] the "outer detect zone" requires four-out-of-four queries in order to trigger a path completion (the outer detect zone consists of four rounds, these being rounds 1, 2 and $r-1$, r). In the current paper, we optimize by always making the outer detect zone trigger a path completion as soon as possible, i.e., by completing a path whenever three-out-of-four matching queries occur in the outer detect zone. (This is similar to an idea of Dachman-Soled et al. [10].) By detecting a little earlier in this fashion, we can move the "adapt zones" on either side by one position towards the left and right edges of the network, effectively removing one round at either end, but this creates a fresh difficulty, as two of the four different types of paths detected by the outer detect zone cannot make use of the new translated adapt zones because the translated adapt zones overlap with the query that triggers the path. For these two types of paths (which are triggered by queries at round 2 or at round $r-1$), we use a brand new adapt zone instead, consisting of the middle two rounds of the network. (Rounds 4 and 5, in our 8-round design.) This itself creates another complication, since an adapted query should not trigger a path completion, lest the proof blow up, and since the "middle detect zone" is traditionally made up of rounds 4 and 5 precisely. We circumvent this problem with a fresh trick: We split the middle detect zone into two separate overlapping zones, each of which has *three* rounds: rounds 3, 4, 5 for one zone, rounds 4, 5, 6 for the other; after this change, adapted queries at rounds 4, 5 (and as argued within the proof) do not trigger either of the middle detect zones. The simulator's "termination argument" is slightly affected by the presence of two separate middle detect zones, but not much: one can observe that neither type of middle path detection adds queries at rounds 4 and 5, even though paths triggered by one middle detect zone can trigger a path in the other middle detect zone. Hence, the original termination argument of Coron et al. [9] (used in [11,14,20] and in many other places since) goes through practically unchanged.

The resulting 8-round simulator ends up having a highly symmetric structure: It can be abstracted as having four detect zones of three consecutive rounds each, with two "inner zones" (rounds 3, 4, 5 and 4, 5, 6) and two "outer zones" (rounds 1, 2, 8 and 1, 7, 8); each detect zone of three consecutive rounds detects "at either end" (e.g., the detect zone with rounds 3, 4, 5 detects at rounds 3 and 5, etc); the upshot is that each of rounds $1, \ldots, 8$ ends up being a detection point for exactly one of the four three-round detect zones. We refer to Fig. 2 in Section 3. A much more leisurely description of our simulator can be found in Section 3.

Other Related Work. Before [9], Dodis and Puniya [13] investigated the indifferentiability of Feistel networks in the so-called *honest-but-curious* model, which is incomparable to the standard notion of indifferentiability. They found that in this case, a super-logarithmic number of rounds is sufficient to achieve indifferentiability. Moreover, [9] later showed that super-logarithmically many rounds are also necessary.

Besides Feistel networks, the indifferentiability of many other types of constructions (and particularly hash functions and compression functions) have been investigated. More specifically on the blockcipher side, [1] and [21] investigate the indifferentiability of key-alternating ciphers (with and without an idealized key scheduler, respectively). In a recent eprint note, Dodis et al. [14] investigate the indifferentiability of substitution-permutation networks, treating the S-boxes as independent idealized permutations. Moreover, the "LIFO" design philosophy of [11]—that also carries over to this work—is partly inspired by the latter simulator, as explained in [11].

It should be recalled that indifferentiability does not apply to a cryptographic game for which the adversary is stipulated to come from a special class that does not contain the computational class to which the simulator belongs (the latter class being typically "probabilistic polynomial-time"). See [29].

Finally, Feistel networks have been the subject of a very large body of work in the secret-key (or "indistinguishability") setting, such as in [22,24,25,30] and the references therein.

2 Definitions and Main Result

Feistel Networks. Let $r \geq 0$ and let $F_1, \ldots, F_r : \{0,1\}^n \to \{0,1\}^n$. Given values $x_0, x_1 \in \{0,1\}^n$ we define values $x_2, \ldots, x_{r+1}$ by

$$x_{i+1} = F_i(x_i) \oplus x_{i-1}$$

for $1 \leq i \leq r$. As noted in the introduction, the application

$$(x_0, x_1) \to (x_r, x_{r+1})$$

defines a permutation of $\{0,1\}^{2n}$. We let

$$\Psi[F_1, \ldots, F_r]$$

denote this permutation. We say that Ψ is an *r-round Feistel network* and that F_i is the *i-th round function* of Ψ.

In this paper, whenever a permutation is given as an oracle, our meaning is that both forward and inverse queries can be made to the permutation. This applies in particular to Feistel networks.

Indifferentiability. A *construction* is a stateless deterministic algorithm that evaluates by making calls to an external set of *primitives*. The latter are functions that conform to a syntax that is specified by the construction. Thus $\Psi[F_1, \ldots, F_r]$ can be seen as a construction with primitives $F_1, \ldots, F_r$. In the general case we notate a construction C with oracle access to a set of primitives Q as C^Q.

A primitive is *ideal* if it is drawn uniformly at random from the set of all functions meeting the specified syntax. A *random function* $F : \{0,1\}^n \to \{0,1\}^n$ is a particular case of an ideal primitive. Such a function is drawn uniformly at random from the set of all functions of domain $\{0,1\}^n$ and of range $\{0,1\}^n$.

A *simulator* is a stateful randomized algorithm that receives and answer queries, possibly being given oracles of its own. We assume that a simulator is initialized to some default state (which constitutes part of the simulator's description) at the start of each experiment. A simulator S with oracle access to an ideal primitive Z is notated as S^Z.

A *distinguisher* is an algorithm that initiates a query-response session with a set of oracles, that has a limited total number of queries, and that outputs 0 or 1 when the query-response session is over. In our case distinguishers are information-theoretic; this implies, in particular, that the distinguisher can "know by heart" the (adaptive) sequence of questions that will maximize its distinguishing advantage. In particular, one may assume without loss of generality that a distinguisher is deterministic.

Indifferentability seeks to determine when a construction C^Q, where Q is a set of ideal primitives, is "as good as" an ideal primitive Z that has the same syntax (interface) as C^Q. In brief, there must exist a simulator S such that having oracle access to the pair (C^Q, Q) (often referred to as the "real world") is indistinguishable from the pair (Z, S^Z) (often referred to as the "simulated world").

In more detail we refer to the following definition, which is due to Maurer et al. [23].

Definition 1. A construction C with access to a set of ideal primitives Q is (t_S, q_S, ε)-*indifferentiable* from an ideal primitive Z if there exists a simulator $S = S(q)$ such that

$$\Pr\left[D^{C^Q, Q} = 1\right] - \Pr\left[D^{Z, S^Z} = 1\right] \leq \varepsilon$$

for every distinguisher D making at most q queries in total, and such that S runs in total time t_S and makes at most q_S queries to Z. Here t_S, q_S and ε are functions of q, and the probabilities are taken over the randomness in Q, Z, S and (if any) in D.

As indicated, we allow S to depend on q.[1] The notation

$$D^{C^Q, Q}$$

indicates that D has oracle access to C^Q as well as to each of the primitives in the set Q. We also note that the oracle

$$S^Z$$

offers one interface for D to query for each of the primitives in Q; however the simulator S is "monolithic" and treats each of these queries with knowledge of the others.

Thus, S's job is to make Z look like C^Q by inventing appropriate answers for D's queries to the primitives in Q. In order to do this, S requires oracle access to Z. On the other hand, S doesn't know which queries D is making to Z.

Informally, C^Q is *indifferentiable* from Z if it is (t_S, q_S, ε)-indifferentiable for "reasonable" values of t_S, q_S and for ε negligibly small in the security parameter n. The value q_S in Definition 1 is called the *query complexity* of the simulator.

In our setting C will be the 8-round Feistel network Ψ and Q will be the set $\{F_1, \dots, F_8\}$ of round functions, with each round function being an independent random function. Consequently, Z (matching C^Q's syntax) will be a random permutation from $\{0,1\}^{2n}$ to $\{0,1\}^{2n}$, queriable (like C^Q) in both directions; this random permutation is notated P in the body of the proof.

Main Result. The following theorem is our main result. In this theorem, Ψ plays the role of the construction C, while $\{F_1, \dots, F_8\}$ (where each F_i is an independent random function) plays the role of Q, the set of ideal primitives called by C.

[1] This introduces a small amount of non-uniformity into the simulator, but which seems not to matter in practice. While in our case the dependence of S on q is made mainly for the sake of simplicity and could as well be avoided (with a more convoluted proof and a simulator that runs efficiently only with high probability), we note, interestingly, that there is one indifferentiabiliy result that we are aware of—namely that of [16]—for which the simulator crucially needs to know the number of distinguisher queries in advance.

Theorem 1. *The Feistel network $\Psi[F_1, \ldots, F_8]$ is (t_S, q_S, ε)-indifferentiable from a random $2n$-bit to $2n$-bit permutation with $t_S = O(q^4)$, $q_S = 32q^4 + 8q^3$ and $\varepsilon = 7400448q^8/2^n$. Moreover, these bounds hold even if the distinguisher is allowed to make q queries to* each *of its 9 (= 8 + 1) oracles.*

The simulator that we use to establish Theorem 1 is described in the next section. The proof of Theorem 1 can be found in the full version of this paper [12].

Miscellaneous Notations. We write $[k]$ for the set $\{1, \ldots, k\}$, $k \in \mathbb{N}$.

The symbol $\perp$ denotes an uninitialized or null value and can be taken to be synonymous with a programming language's **null** value, though we reserve the latter for uninitialized object fields. If T is a table, moreover, we write $x \in T$ to mean that $T(x) \neq \perp$. Correspondingly, $x \notin T$ means $T(x) = \perp$.

3 High-Level Simulator Overview

In this section we give a somewhat non-technical overview of our 8-round simulator which, like [20] and [11], is a modification of a 10-round simulator by Seurin [31].

Round Function Tables. We recall that the simulator is responsible for 8 interfaces, i.e., one for each of the rounds functions. These interfaces are available to the adversary through a single function, named

$$\mathrm{F}$$

and which takes two inputs: an integer $i \in [8]$ and an input $x \in \{0,1\}^n$.

Correspondingly to these 8 interfaces, the simulator maintains 8 tables, notated $F_1, \ldots, F_8$, whose fields are initialized to $\perp$: initially, $F_i(x) = \perp$ for all $x \in \{0,1\}^n$, all $i \in [8]$. (Hence we note that F_i is no longer the name of a round function, but the name of a table. The i-th round function is now $\mathrm{F}(i, \cdot)$.) The table F_i encodes "what the simulator has decided so far" about the i-th round function. For instance, if $F_i(x) = y \neq \perp$, then any subsequent distinguisher query of the form $\mathrm{F}(i, x)$ will simply return $y = F_i(x)$. Entries in the tables $F_1, \ldots, F_8$ are not overwritten once they have been set to non-$\perp$ values.

The $2n$-bit Random Permutation. Additionally, the distinguisher and the simulator both have oracle access to a random permutation on $2n$ bits, notated

$$\mathrm{P}$$

and which plays the role of the ideal primitive Z in Definition 1. Thus P accepts an input of the form $(x_0, x_1) \in \{0,1\}^n \times \{0,1\}^n$ and produces an output $(x_8, x_9) \in \{0,1\}^n \times \{0,1\}^n$. P's inverse P^{-1} is also available as an oracle to both the distinguisher and the simulator.

Distinguisher Intuition and Completed Paths. One can think of the distinguisher as checking the consistency of the oracles $\mathrm{F}(1, \cdot)$, ..., $\mathrm{F}(8, \cdot)$ with

P/P^{-1}. For instance, the distinguisher could choose random values $x_0, x_1 \in \{0,1\}^n$, construct the values $x_2, \ldots, x_9$ by setting

$$x_{i+1} \leftarrow \mathrm{F}(i, x_i) \oplus x_{i-1}$$

for $i = 2, \ldots, 9$, and finally check if $(x_8, x_9) = \mathrm{P}(x_0, x_1)$. (In the real world, this will always be the case; if the simulator is doing its job, it should also be the case in the simulated world.) In this case we also say that the values

$$x_1, \ldots, x_8$$

queried by the distinguisher form a *completed path*. (The definition of a "completed path" will be made more precise in the next section.)

It should be observed that the distinguisher has multiple options for completing paths; e.g., "left-to-right" (as above), "right-to-left" (starting from values x_8, x_9 and evaluating the Feistel network backwards), "middle-out" (starting with some values x_i, x_{i+1} in the middle of the network, and growing a path outwards to the left and to the right), "outward-in" (starting from the endpoints x_0, x_1, x_8, x_9 and going right from x_0, x_1 and left from x_8, x_9), etc. Moreover, the distinguisher can try to reuse the same query for several different paths, can interleave the completion of several paths in a complex manner, and so on.

To summarize, and for the purpose of intuition, one can picture the distinguisher as trying to complete all sorts of paths in a convoluted fashion in order to confuse and/or "trap" the simulator in a contradiction.

THE SIMULATOR'S DILEMMA. Clearly a simulator must to some extent detect which paths a distinguisher is trying to complete, and "adapt" the values along these paths such as to make the (simulated) Feistel network compatible with P. Concerning the latter, one can observe that a pair of missing consecutive queries is sufficient to adapt the two ends of a path to one another; thus if, say,

$$x_0, x_1, x_4, x_5, x_6, x_7, x_8, x_9$$

are values such that

$$F_i(x_i) \neq \perp$$

for $i \in \{1, 4, 5, 6, 7, 8\}$, and such that

$$x_{i+1} = x_{i-1} \oplus F_i(x_i)$$

for $i \in \{5, 6, 7, 8\}$, and such that

$$\mathrm{P}(x_0, x_1) = (x_8, x_9)$$

and such that

$$F_2(x_2) = F_3(x_3) = \perp$$

where $x_2 := x_0 \oplus F_1(x_1)$, $x_3 := F_4(x_4) \oplus x_5$, then by making the assignments

$$F_2(x_2) \leftarrow x_1 \oplus x_3 \qquad (2)$$

$$F_3(x_3) \leftarrow x_2 \oplus x_4 \qquad (3)$$

the simulator turns $x_1, \ldots, x_8$ into a completed path that is compatible with P. In such a case, we say that the simulator *adapts a path*. The values $F_2(x_2)$ and $F_3(x_3)$ are also said to be *adapted*.

In general, however, if the simulator always waits until the last minute (e.g., until only two adjacent undefined queries are left) before adapting a path, it can become caught in an over-constrained situation whereby several different paths request different adapted values for the same table entry. Hence, it is usual for simulators to give themselves a "safety margin" and to pre-emptively complete paths some time in advance. When pre-emptively completing a path, typical simulators sample all but two values along the path randomly, while "adapting" the last two values as above.

It should be emphasized that our simulator, like previous simulators [9,20,31], makes no distinction between a non-null value $F_i(x_i)$ that is non-null because the distinguisher has made the query $\mathrm{F}(i, x_i)$ or that is non-null because the simulator has set the value $F_i(x_i)$ during a pre-emptive path completion. (Such a distinction seems tricky to leverage, particularly since the distinguisher can know a value $F_i(x_i)$ without making the query $\mathrm{F}(i, x_i)$, simply by knowing adjacent values and by knowing how the simulator operates.) Moreover, the simulator routinely calls its own interface

$$\mathrm{F}(\cdot, \cdot)$$

during the process of path completion, and it should be noted that our simulator, again like previous simulators, makes no difference between distinguisher calls to F and its own calls to F.

One of the basic dilemmas, then, is to decide at which point it is worth it to complete a path; if the simulator waits too long, it is prone to finding itself in an over-constrained situation; if it is too trigger-happy, on the other hand, it runs the danger of creating out-of-control chain reactions of path completions, whereby the process of completing a path sets off another path, and so on. We refer to the latter problem (that is, avoiding out-of-control chain reactions) as the problem of *simulator termination*.

Seurin's 10-Round Simulator. Our 8-round simulator is based on "tweaking" a previous 10-round simulator of ours [11] which is itself based on Seurin's (flawed) 10-round simulator [31]. Unfortunately (and after some failed efforts of ours to find shortcuts) it seems that the best way to understand our 8-round simulator is to start back with Seurin's 10-round simulator, followed by the modifications of [11] and by the "tweaks" that bring the network down to 8 rounds.

In a nutshell, Seurin's simulator completes a path for *every* pair of values (x_5, x_6) such that $F_5(x_5)$ and $F_6(x_6)$ are defined, as well as for every 4-tuple of values

$$x_1,\, x_2,\, x_9,\, x_{10}$$

such that

$$F_1(x_1),\, F_2(x_2),\, F_9(x_9),\, F_{10}(x_{10})$$

are all defined, and such that

$$\mathrm{P}(x_0, x_1) = (x_{10}, x_{11})$$

where $x_0 := F_1(x_1) \oplus x_2$, $x_{11} := x_9 \oplus F_{10}(x_{10})$. By virtue of this, rounds 5 and 6 are called the *middle detect zone* of the simulator, while rounds 1, 2, 9, 10 are called the *outer detect zone*. (Thus whenever a detect zone "fills up" with matching queries, a path is completed.) Paths are adapted either at positions 3, 4 or else at positions 7, 8, as depicted in Fig. 1.

In a little more detail, a function call $\mathrm{F}(5, x_5)$ for which $F_5(x_5) = \perp$ triggers a path completion for every value x_6 such that $F_6(x_6) \neq \perp$; such paths are adapted at positions 3 and 4. Symmetrically, a function call $\mathrm{F}(6, x_6)$ for which $F_6(x_6) = \perp$ triggers a path completion for every value x_5 such that $F_5(x_5) \neq \perp$; such paths are adapted at positions 7 and 8. For the outer detect zone, a call $\mathrm{F}(2, x_2)$ such that $F_2(x_2) = \perp$ triggers a path completion for every tuple of values x_1, x_9, x_{10} such that $F_1(x_1)$, $F_9(x_9)$ and $F_{10}(x_{10})$ are defined, and such that the constraints listed above are satisfied (verifying these constraints thus requires a call to P or P^{-1}); such paths are adapted at positions 3, 4. Paths that are symmetrically triggered by a query $\mathrm{F}(9, x_9)$ are adapted at positions 7, 8. Function calls to $\mathrm{F}(2, \cdot)$, $\mathrm{F}(5, \cdot)$, $\mathrm{F}(6, \cdot)$ and $\mathrm{F}(9, \cdot)$ are the only ones to trigger path completions. (Indeed, one can easily convince oneself that sampling a new value $F_1(x_1)$ or $F_{10}(x_{10})$ can only trigger the outer detect zone with negligible probability; hence, this possibility is entirely ignored by the simulator.) To summarize, in all cases the completed path is adapted at positions that are immediately *next to* the query that triggers the path completion.

To more precisely visualize the process of path completion, imagine that a query

$$\mathrm{F}(2, x_2)$$

has just triggered the second type of path completion, for some corresponding values x_1, x_9 and x_{10}; then Seurin's simulator (which would immediately lazy sample the value $F_2(x_2)$ even before checking if this query triggers any path completions) would (a) make the queries

$$\mathrm{F}(8, x_8), \ldots, \mathrm{F}(6, x_6), \mathrm{F}(5, x_5)$$

to itself in that order, where $x_{i-1} := F_i(x_i) \oplus x_{i+1} = \mathrm{F}(i, x_i) \oplus x_{i+1}$ for $i = 9, \ldots, 6$, and (b) adapt the values $F_3(x_3)$, $F_4(x_4)$ as in (2), (3) where $x_3 := x_1 \oplus F_2(x_2)$, $x_4 := F_5(x_5) \oplus x_6$. In general, some subset of the table entries

$$F_8(x_8), \ldots, F_5(x_5)$$

(and more exactly, a prefix of this sequence) may be defined even before the queries $\mathrm{F}(8, x_8)$, $\ldots$, $\mathrm{F}(5, x_5)$ are made. The crucial fact to argue, however, is that $F_3(x_3) = F_4(x_4) = \perp$ right before these table entries are adapted.

Extending this example a little, say moreover that $F_6(x_6) = \perp$ at the moment when the above-mentioned query

$$\mathrm{F}(6, x_6)$$

is made. This will trigger another path completion for every value x_5^* such that

$F_5(x_5^*) \neq \perp$ at the moment when the query $\mathrm{F}(6, x_6)$ occurs. Analogously, such a path completion would proceed by making (possibly redundant) queries

$$\mathrm{F}(4, x_4^*), \ldots, \mathrm{F}(1, x_1^*), \mathrm{F}(10, x_{10}^*), \mathrm{F}(9, x_9^*)$$

for values $x_4^*, \ldots, x_1^*, x_0^*, x_{11}^*, x_{10}^*, x_9^*$ that are computed in the obvious way (with a query to P to go from (x_0^*, x_1^*) to (x_{10}^*, x_{11}^*), where $x_0^* := F_1(x_1^*) \oplus x_2^*)$, before adapting the path at positions 7, 8. The crucial fact to argue would again be that $F_7(x_7^*) = F_8(x_8^*) = \perp$ when the time comes to adapt these table values, where $x_8^* := F_9(x_9^*) \oplus x_{10}^*$, $x_7^* := x_5^* \oplus F_6(x_6)$.

In Seurin's simulator, moreover, paths are completed on a first-come-first-serve (or FIFO[2]) basis: while paths are "detected" immediately when the query that triggers the path completion is made, this information is shelved for later, and the actual path completion only occurs after all previously detected paths have been completed. In our example, for instance, the path triggered by the query $\mathrm{F}(2, x_2)$ would be adapted before the path triggered by the query $\mathrm{F}(6, x_6)$. The imbroglio of semi-completed paths is rather difficult to keep track of, however, and indeed Seurin's simulator was later found to suffer from a real "bug" related to the simultaneous completion of multiple paths [20,32].

MODIFICATIONS OF [11]. For the following discussion, we will say that x_2, x_5 constitute the *endpoints* of a path that is adapted at positions 3, 4; likewise, x_6, x_9 constitute the *endpoints* of a path that is adapted at positions 7, 8. Hence, the endpoints of a path are the two values that flank the adapt zone. We say that an endpoint x_i is *unsampled* if $F_i(x_i) = \perp$ and *sampled* otherwise. Succinctly, the philosophy espoused in [11] is to not sample the endpoints of a path until right before the path is about to be adapted or, even more succinctly, "to sample randomness at the moment it is needed". This essentially results in two main differences with Seurin's simulator, which are (i) changing the order in which paths are completed and (ii) doing "batch adaptations" of paths, i.e., adapting several paths at once, for paths that happen to share endpoints.

To illustrate the first point, return to the above example of a query

$$\mathrm{F}(2, x_2)$$

that triggers a path completion of the second type with respect to some values x_1, x_9, x_{10}. Then by definition

$$F_2(x_2) = \perp$$

at the moment when the call $\mathrm{F}(2, x_2)$ is made. Instead of immediately sampling $F_2(x_2)$, as in the original simulator, this value is kept "pending" (the technical term is "pending query") until it comes time to adapt the path. Moreover, and keeping the notations from the previous example, note that the query

$$\mathrm{F}(6, x_6)$$

[2] FIFO: First-In-First-Out. LIFO: Last-In-First-Out.

will not result in $F_6(x_6)$ being immediately lazy sampled either (assuming, that is, $F_6(x_6) = \perp$) as long as there is at least one value x_5^* such that $F_5(x_5^*) \neq \perp$, since in such a case x_6 is the endpoint of a path-to-be-completed (namely, the path which we notated as $x_1^*, \ldots, x_5^*, x_6, x_7^*, \ldots, x_{10}^*$ above), and, according to the new policy, this endpoint must be kept unsampled until that path is adapted. In particular, the value $x_5 = F_6(x_6) \oplus x_7$ from the "original" path *cannot be computed* until the "secondary" path containing x_5^* and x_6 has been completed (or even more: until *all* secondary paths triggered by the query $\mathrm{F}(6, x_6)$ have been completed). In other words, the query $\mathrm{F}(6, x_6)$ "holds up" the completion of the first path. In practical terms, paths that are detected during the completion of another path take precedence over the original path, so that path completion becomes a LIFO process.

Implicitly, the requirement that both endpoints of a path *remain* unsampled until further notice means that both endpoints are *initially* unsampled. For the "starting" endpoint of the path (i.e., where the path is detected) this is obvious, since the path cannot be triggered otherwise, while for the "far" endpoint of the path one can argue that it holds with high probability.

As for "batch adaptations" the intuitive idea is that paths that share unsampled endpoints must be adapted (and in particular have their endpoints lazy sampled) simultaneously. In this event, the group of paths that are collectively sampled[3] and adapted will be an equivalence class in the transitive closure of the relation "shares an endpoint with". Note that paths adapted at 3, 4 can only share their endpoints[4] with other paths adapted at 3, 4, while paths adapted at 7, 8 can only share their endpoints with other paths adapted at 7, 8. Hence the paths in such an equivalence class will, in particular, all have the same adapt zone. Moreover, the batch adaptation of such a group of paths cannot happen at any point in time, but must happen when the group of paths is "stable": none of the endpoints of the paths in the group should currently be a trigger for a path completion that has not yet been detected, or that has started to complete but that has not yet reached its far endpoint. It so turns out, moreover, that the topological structure of such an equivalence class (with endpoints as nodes and paths as edges) will be a tree with all but negligible probability, simplifying many aspects of the simulator and of the proof.

While this describes the (simple) high-level idea of batch adaptations, the implementation details are more tedious. In fact, at this point it is useful to focus on these details.

Further Details: Pending Queries, Trees, Etc. Keeping with the 10-round simulator of [11], if a query $\mathrm{F}(i, x_i)$ occurs with $F_i(x_i) = \perp$ and $i \in \{2, 5, 6, 9\}$ the simulator creates a so-called *pending query* at that position, and for that value of x_i. (Strictly speaking, the pending query is the pair (i, x_i).) One

[3] In this context we use the verb "sampled" as a euphemism for "have their endpoints sampled".

[4] Recall that the endpoints of a path with adapt zone 3, 4 are x_2 and x_5, and that the endpoints of a path with adapt zone 7, 8 are x_6 and x_9.

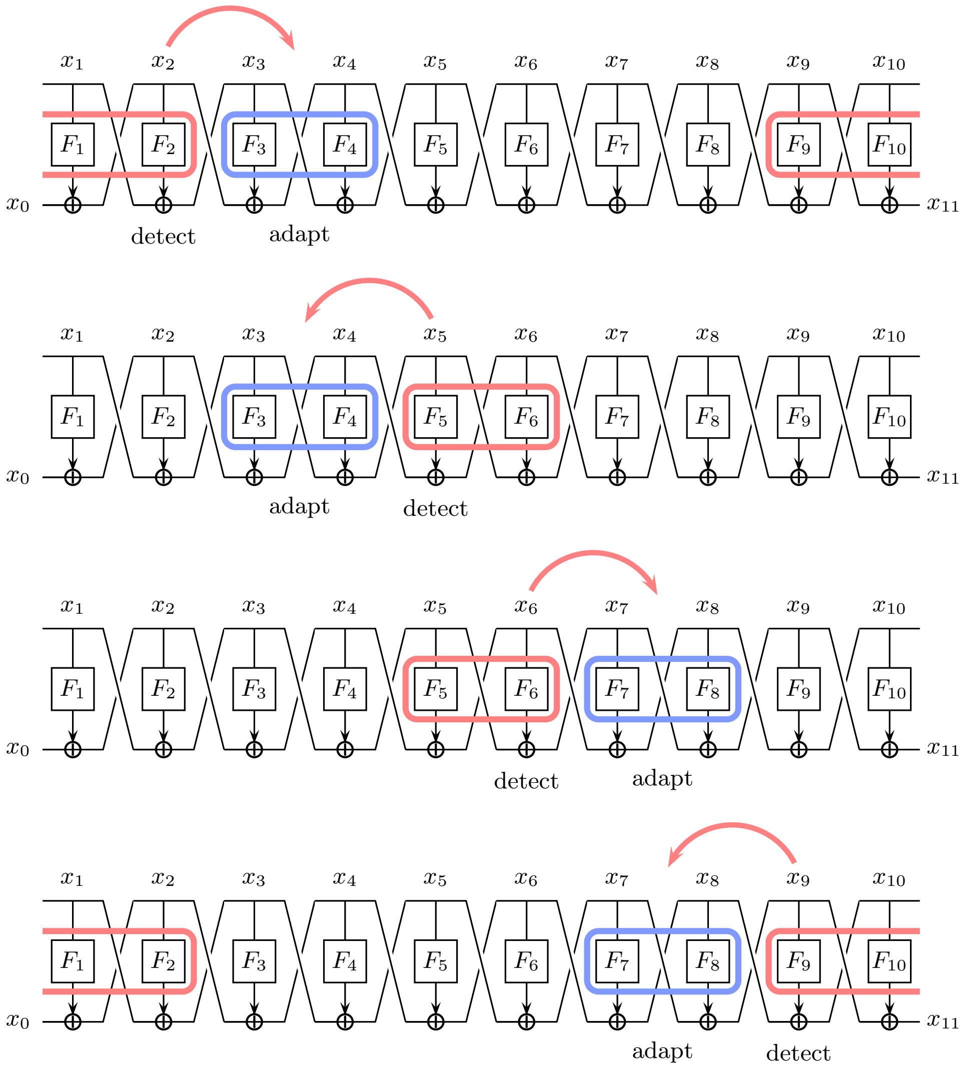
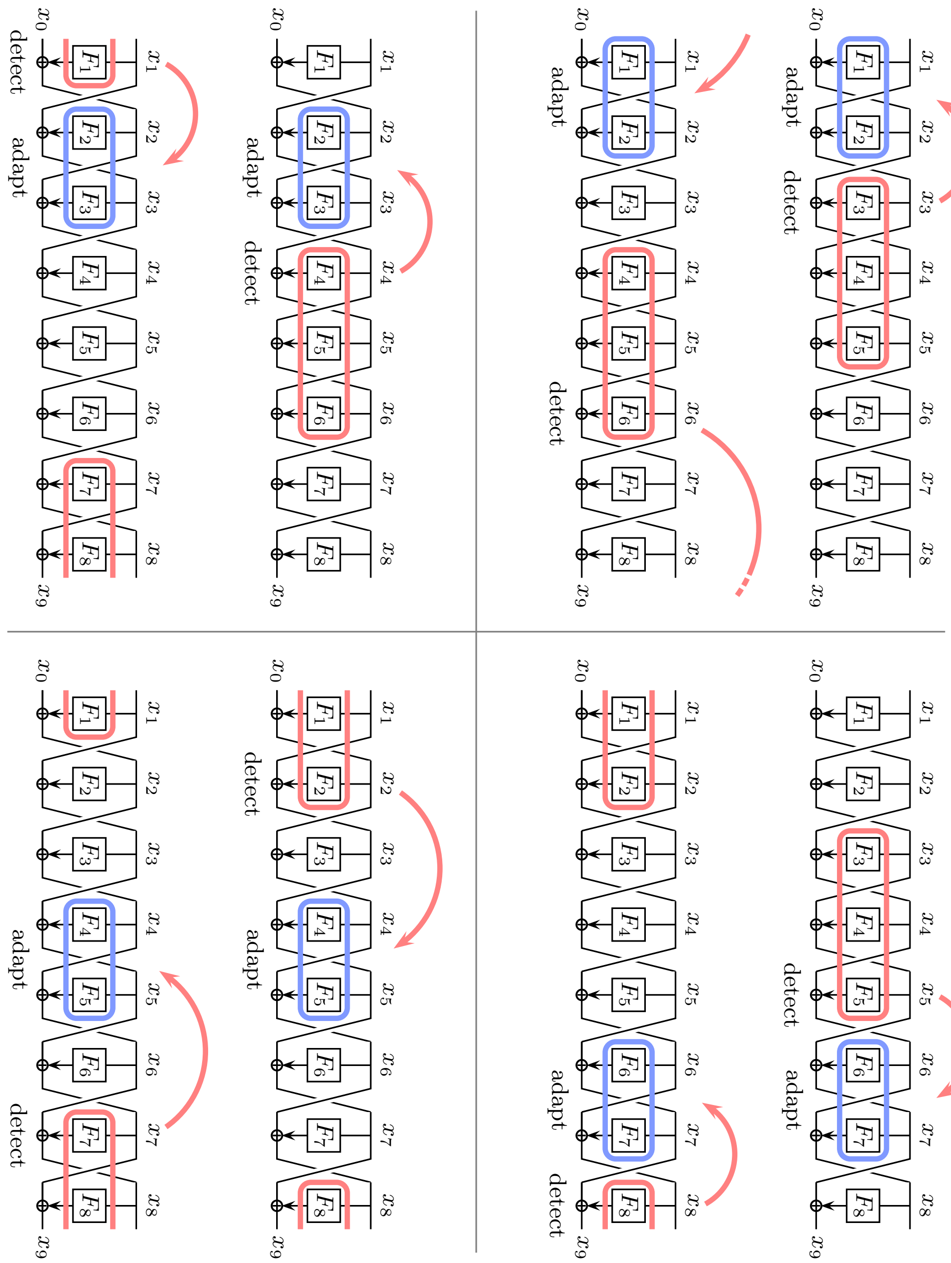

x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}
F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8 F_9 F_{10}
x_0
x_{11}
detect
adapt
adapt
detect
detect
adapt
adapt
detect

Fig. 1. A sketch of the 10-round simulator from [11] (and also Seurin's 10-round simulator). Rounds 5 and 6 form one detect zone; rounds 1, 2, 9 and 10 form another detect zone; rounds 3 and 4 constitute the left adapt zone, 7 and 8 constitute the right adapt zone; red arrows point from the position where a path is detected (a.k.a., "pending query") to the adapt zone for that path.



x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
F_1 F_2 F_3 F_4 F_5 F_6 F_7 F_8
adapt
detect

Fig. 2. A sketch of our 8-round simulator drawn in the same style as Fig. 1. Red groups of three queries are detect zones; when a query completing a detect zone (a.k.a., "pending query") occurs at one of the endpoints of the zone, a path completion is triggered; the adapt zone for that path completion is shown in blue; the four quadrants correspond to the four possible adapt zones. (The adapt zone at positions F_1, F_2 in the upper right quadrant could equivalently be moved to F_7, F_8.)

can think of a pending query as a kind of "beacon" that periodically[5] checks for new paths to trigger, as per the rules of Fig. 1. E.g., a pending query

$$(2, x_2)$$

will trigger a new path to complete for any tuple of values x_1, x_9, x_{10} such that (same old!)

$$F_1(x_1) \neq \bot, F_9(x_9) \neq \bot, F_{10}(x_{10}) \neq \bot$$

and such that

$$\mathrm{P}(x_0, x_1) = (x_{10}, x_{11})$$

where $x_0 := F_1(x_1) \oplus x_2$, $x_{11} := x_9 \oplus F_{10}(x_{10})$. The tuple of queries x_1, x_9, x_{10} is also called a *trigger* for the pending query $(2, x_2)$. For a pending query $(9, x_9)$, a trigger is a tuple x_1, x_2, x_{10} subject to the symmetric constraints. For a pending query $(5, x_5)$, a trigger is any value x_6 such that $F_6(x_6) \neq \bot$, and likewise any value x_5 such that $F_5(x_5) \neq \bot$ is a trigger for any pending query $(6, x_6)$. We note that a pending query *triggers* a path when there exists a *trigger* for the pending query. Hence there the word "trigger" has two slightly different uses (as a noun and as a verb).

We differentiate the endpoints of a path according to which one triggered the path: the pending query that triggered the path is called the *origin* of the path, while the other endpoint (if and when present) is the *terminal* of the path.

While pending queries are automatically created each time a function call $\mathrm{F}(i, x_i)$ occurs with $F_i(x_i) = \bot$ and with $i \in \{2, 5, 6, 9\}$, the simulator also has a separate mechanism[6] at its disposal for directly creating pending queries without calling $\mathrm{F}(\cdot, \cdot)$ by this mechanism. In particular, whenever the simulator reaches the terminal of a path, the simulator turns the terminal into a pending query.

In short: (i) all path endpoints are pending queries, so long as the path has not been sampled and adapted; (ii) pending queries keep triggering paths as long as there are paths to trigger.

For the following, we will use the following extra terminology from [11]:

- A path is *ready* when it has been extended to the terminal, and the terminal has been made pending.
- A ready path with endpoints 2, 5 is called a "$(2, 5)$-path", and a ready path with endpoints 6, 9 is called a "$(6, 9)$-path".
- Two ready paths are *neighbors* if they share an endpoint; let a *neighborhood* be an equivalence class of ready paths under the transitive closure of the neighbor relation. We note that a neighborhood consists either of all $(2, 5)$-paths or consists all of $(6, 9)$-paths.
- A pending query is *stable* if it has no "new" triggers (that is, no triggers for which the simulator hasn't already started to complete a path), and if paths already triggered by the pending query are ready.

[5] The simulator is not multi-threaded, but this metaphor is still helpful.

[6] This might sound a bit ad-hoc right now, but it actually corresponds to the most natural way of programming the simulator, as will become clearer in the technical simulator overview.

- A neighborhood is *stable* if all the endpoints of all the paths that it contains are stable.

A neighborhood can be visualized as a graph with a node for each endpoint and an edge for each ready path. As mentioned above, these neighborhoods actually turn out to be trees with high probability. (The simulator aborts otherwise.) We will thus speak of a $(2,5)$-*tree* for a neighborhood consisting of $(2,5)$-paths and of a $(6,9)$-*tree* for a neighborhood consisting of $(6,9)$-paths. Moreover, the simulator uses an actual tree *data structure* to keep track of each (i,j)-tree under completion, thus adding further structure to the simulation process.

To summarize, when a query $\mathrm{F}(i, x_i)$ triggers a path completion, the simulator starts growing a tree that is "rooted" at the pending query (i, x_i); for other endpoints of paths in this tree (i.e., besides (i, x_i)), the simulator "plants" a pending query at that endpoint without making a call to $\mathrm{F}(\cdot,\cdot)$, which pending query tests for further paths to complete, and which may thus cause the tree to grow even larger, etc. If and when the tree becomes stable, the simulator samples all endpoints of all paths in the tree, and adapts all these paths.[7]

The growth of a $(2,5)$-tree may at any moment be interrupted by the apparition of a new $(6,9)$-tree (specifically, when a query to $\mathrm{F}(6,\cdot)$ or $\mathrm{F}(9,\cdot)$ triggers a new path completion), in which case the $(2,5)$-tree is put "on hold" while the $(6,9)$-tree is grown, sampled and adapted; vice-versa, a $(6,9)$-tree may be interrupted by the apparition of a new $(2,5)$-tree. In this fashion, a "stack of trees" that alternates between $(2,5)$- and $(6,9)$-trees is created. Any tree that is not the last tree on the stack contains a non-ready path (the one, that is, that was interrupted by the next tree on the stack), and so is not stable. For this reason, in fact, the only tree that can become stable at a given moment is the last tree on the stack.

We also note that in certain cases (and more specifically for pending queries

[7] In more detail, when a tree becomes stable the simulator lazy samples

$$F_i(x_i)$$

for every endpoint (a.k.a., pending query) in the tree. Then if the tree is, say, a $(2,5)$-tree, the simulator can compute the values

$$\begin{aligned} x_3 &:= x_1 \oplus F_2(x_2) \\ x_4 &:= F_5(x_5) \oplus x_6 \end{aligned}$$

and set

$$\begin{aligned} F_3(x_3) &:= x_2 \oplus x_4 \\ F_4(x_4) &:= x_3 \oplus x_5 \end{aligned}$$

for each path in the tree. If two paths "collide" by having the same value of x_3 or x_4 the simulator aborts. Likewise the simulator aborts if either $F_3(x_3) \neq \perp$ or $F_4(x_4) \neq \perp$ for some path, before adapting those values. We call this two-step process "sampling and adapting" the $(2,5)$-tree. The process of sampling and adapting a $(6,9)$-tree is analogous.

at positions 5 and 6), trees higher up in the stack can affect the stability of nodes of trees lower down in the stack: a node that used to be stable loses its stability after a higher-up tree has been created, sampled and adapted. Hence, the simulator always re-checks all nodes of a tree "one last time" before deeming a tree stable, after a tree stops growing—and such a check will typically, indeed, uncover new paths to complete that weren't there before. Moreover, because the factor that determines when these new paths will be adapted is the timestamp of the *pending query* to which they are attached, rather than the timestamp of the *actual last query* that completed a trigger for this pending query, it is a matter of semantic debate whether the simulator of [11] is really "LIFO" or not. (But conceptually at least, it seems safe to think of the simulator as LIFO.)

STRUCTURAL VS. CONCEPTUAL CHANGES. Of the main changes introduced in [11] to Seurin's simulator, one can note that "batch adaptations" are in some sense a conceptual convenience. Indeed, one way or another every non-null value

$$F_j(x_j)$$

for $j \notin \{3, 4, 7, 8\}$ ends up being randomly and independently sampled in their simulator, as well as in Seurin's; so one might as well load a random value into $F_j(x_j)$ as soon as the query $\mathrm{F}(j, x_j)$ is made, as in Seurin's original simulator, as long as we take care to keep on completing paths in the correct order. While correct, this approach is conceptually less convenient, because the "freshness" of the random value $F_j(x_j)$ is harder to argue when that randomness is needed (e.g., to argue that adapted queries do not collide, etc). In fact, our 10-round simulator is an interesting case where the search for a syntactically convenient usage of randomness naturally leads to structural changes that turn out to be critical for correctness.

One should note that the idea of batch adaptations already appears explicitly in the simulator of [14], which, indeed, formed part of the inspiration for [11]. In [14], however, batch adaptations are purely made for conceptual convenience.

Readers seeking more concrete insights can also consult Seurin's attack against his own 10-round simulator [32] and check this attack fails under the LIFO path completion just outlined.

THE 8-ROUND SIMULATOR. In the 10-round simulator, the outer detect zone is in some sense unnecessarily large: for any set of four matching queries that complete the outer detect zone, the simulator can "see" the presence of matching queries already by the third query.

To wit, say the distinguisher chooses random values x_0, x_1, makes the query

$$(x_{10}, x_{11}) \leftarrow \mathrm{P}(x_0, x_1)$$

to P, then queries $\mathrm{F}(1, x_1)$ and $\mathrm{F}(10, x_{10})$. At this point, even if the simulator knows that the values x_1 and x_{10} are related by some query to P, the simulator has no hope of finding *which* query to P, because there are exponentially many possibilities to try for x_0 and/or x_{11}. However, as soon as the distinguisher makes either of the queries

$$\mathrm{F}(2, x_2) \qquad \text{or} \qquad \mathrm{F}(9, x_9)$$

where $x_2 := x_0 \oplus \mathrm{F}(1, x_1)$, $x_9 := \mathrm{F}(10, x_{10}) \oplus x_{11}$, then the simulator has enough information to draw a connection between the queries being made at the left- and right-hand sides of the network. (E.g., if the query $\mathrm{F}(2, x_2)$ is made, the simulator can compute x_0 from $F_1(x_1)$ and x_2, can call $\mathrm{P}(x_0, x_1)$, and recognize, in P's output, the value x_{10} for which it has already answered a query.) More generally, anytime the distinguisher makes three-out-of-four matching queries in the 10-round outer detect zone, the simulator has enough information to reverse-engineer the relevant query to $\mathrm{P}/\mathrm{P}^{-1}$ and, thus, to see a connection between the queries being made at either side of the network.

This observation (which is also made by Dachman-Soled et al. [10], though our work is independent of theirs) motivates the division of the 4-round outer detect zone into two separate outer detect zones of three (consecutive) rounds each. In the eight-round simulator, then, these two three-round outer detect zones are made up of rounds 1, 2, 8 and rounds 1, 7, 8, respectively. Both of these detect zones detect "at the edges" of the detect zone. I.e., the 1, 7, 8 detect zone might trigger a path completion through queries to $\mathrm{F}(7, \cdot)$ and $\mathrm{F}(1, \cdot)$, whereas the 1, 2, 8 detect zone might trigger a path completion through queries to $\mathrm{F}(2, \cdot)$ or to $\mathrm{F}(8, \cdot)$. (Once again the possibility of "completing" a detect zone by a query at the middle of the detect zone is ignored because this event has negligible chance of occuring.)

E.g., a query

$$\mathrm{F}(7, x_7)$$

such that $F_7(x_7) = \perp$ and for which there exists values x_0, x_1, x_8 such that $F_8(x_8) \neq \perp$, $F_1(x_1) \neq \perp$, and such that $\mathrm{P}^{-1}(x_8, x_9) = (x_0, x_1)$ where $x_9 = x_7 \oplus F_8(x_8)$ would trigger the 1, 7, 8 detect zone, and produce a path completion. Similarly, a query

$$\mathrm{F}(1, x_1)$$

such that $F_1(x_1) = \perp$ and for which there exists values x_0, x_7, x_8 such that $F_7(x_7) \neq \perp$, $F_8(x_8) \neq \perp$, and such that $\mathrm{P}^{-1}(x_8, x_9) = (x_0, x_1)$ where $x_9 = x_7 \oplus F_8(x_8)$ would trigger the 1, 7, 8 detect zone as well.

When a path is detected at position 1 or at position 8, we can respectively adapt the path at positions 2, 3 or at positions 6, 7—i.e., we adapt the path in an adapt zone that is immediately adjacent to the position that triggered the path completion, as in the[8] 10-round simulator. However, for paths detected at positions 2 and 7, the same adapt zones cannot be used, and we find it more convenient to adapt the path at rounds 4, 5, as depicted in the bottom left quadrant of Fig. 1.

To keep the proof manageable, however, one of the imperatives is that an "adapted" query should not trigger a new path completion. If we kept the middle detect zone as rounds 4, 5 only (by analogy with the 10-round simulator, where the middle detect zone consists of rounds 5 and 6), then the queries that we adapt at rounds 4, 5 would trigger new path completions of themselves—a mess! However, this problem can be avoided by splitting the middle detect zone into

[8] Henceforth, "the" 10-round simulator refers to the simulator of [11].

two *enlarged* middle detect zones of three rounds each: one middle detect zone consisting of rounds 3, 4, 5 and one consisting of rounds 4, 5, 6. As before, each of these zones detects "at the edges". After this change, bad dreams are dissipated, and the 8-round simulator recovers essentially the same functioning as the 10-round simulator. The sum total of detect and adapt zones, including which adapt zone is used for paths detected at which point, is shown in Fig. 2.

The 8-round simulator utilizes the same "pending query" mechanism as the 10-round simulator. In particular, now, each query

$$\mathrm{F}(j, x_j)$$

with $F_j(x_j) = \perp$ creates a new pending query (j, x_j), because paths are now detected at all positions, and each pending query will detect for paths as depicted[9] in Fig. 2, with there being exactly one type of "trigger" for each position j. A path triggered by a pending query is first extended to a designated terminal (the "other" endpoint of the path), the position of which is a function of the pending query that triggered the path (this position is shortly to be discussed), which becomes a new pending query of its own, etc. As in the 10-round simulator, the simulator turns the terminal into a pending query without making a call to $\mathrm{F}(\cdot, \cdot)$.

For the 10-round simulator, we recall that the possible endpoint positions of a path are 2, 5 and 6, 9. The 8-round simulator has more variety, as the endpoints of a path do not always directly flank the adapt zone for that path. Specifically:

– paths detected at positions 1 and 4, as in the top left quadrant of Fig. 2, have endpoints 1, 4; before such paths are adapted, they include only the values x_1, x_4, x_5, x_6, x_7, x_8
– paths detected at positions 3 and 6, as in the top right quadrant of Fig. 2, have endpoints 3, 6; before such paths are adapted, they include only the values x_3, x_4, x_5, x_6
– paths detected at positions 2 and 7, as in the bottom left quadrant of Fig. 2, have endpoints 2, 7; before such paths are adapted, they include only the values x_1, x_2, x_7, x_8
– paths detected at positions 5 and 8, as in the bottom right quadrant of Fig. 2, have endpoints 5, 8; before such paths are adapted, they include only the values x_1, x_2, x_3, x_4, x_5, x_8

Hence, paths with endpoints 1, 4 or 5, 8 are familiar from the 10-round simulator. (Being the analogues, respectively, of paths with endpoints 2, 5 or 6, 9.) On the other hand, paths with endpoints 3, 6 or 2, 7 are shorter, containing only four

[9] To solidify things with some examples, a "trigger" for a pending query $(5, x_5)$ is a pair values of x_3, x_4 such that $F_3(x_3) \neq \perp$, $F_4(x_4) \neq \perp$ and such that $x_3 \oplus F_4(x_4) = x_5$, corresponding to the rightmost, bottommost diagram of Fig. 2; a "trigger" for a pending query $(1, x_1)$ is pair of values x_7, x_8 such that $F_7(x_7) \neq \perp$, $F_8(x_8) \neq \perp$, and such that $\mathrm{P}^{-1}(x_8, x_9) = (*, x_1)$ where $x_9 := x_7 \oplus F_8(x_8)$, corresponding to the leftmost, topmost diagram of Fig. 2. Etc.

values before adaptation takes place. As in the 10-round simulator, we speak of an "(i,j)-path" for paths with endpoints i, j. We also say that a path is *ready* once it has reached both its endpoints and these have been turned into pending queries, and that two ready paths are *neighbors* if they share an endpoint.

Since, by virtue of the endpoint positions, a $(1,4)$-path can only share an endpoint with a $(1,4)$-path, a $(2,7)$-path can only share an endpoint with a $(2,7)$-path, a $(3,6)$-path can only share an endpoint with $(3,6)$-path, and a $(5,8)$-path can only share an endpoint with a $(5,8)$-path, neighborhoods (which are the transitive closure of the neighbor relation) are always comprised of the same kind of (i,j)-path. As in the 10-round simulator, these neighborhoods are actually topological trees, giving rise, thus, to "$(1,4)$-trees", "$(2,7)$-trees", "$(3,6)$-trees" and "$(5,8)$-trees". Given this, the 8-round simulator functions entirely analogously to the 10-round simulator, only with more different types of paths and of trees (which does not make an important difference) and with a slightly modified mechanism for adapting $(2,7)$- and $(3,6)$-trees, which are the trees for which the path endpoints are not directly adjacent to the adapt zone (which does not make an important difference either).

Concerning the latter point, when a $(2,7)$- or $(3,6)$-tree is adapted, some additional queries have to be lazy sampled for each path before reaching the adapt zone. (In the case of a $(3,6)$-tree, each path even requires a query to P^{-1}.) But because the endpoints of each path are lazy sampled as the first step of the batch adaptation process, there is negligible chance that these extra queries will trigger a new path completion. So for those queries the 8-round simulator directly lazy samples the tables F_i without even calling its own $\mathrm{F}(\cdot,\cdot)$ interface.

As a small piece of trivia (since it doesn't really matter to the simulator), one can check, for instance, that a $(1,4)$-tree may be followed either by a $(2,7)$-, $(3,6)$-, or a $(5,8)$-tree on the stack—i.e., while making a $(1,4)$-path ready, we may trigger any of the other three types of paths—and symmetrically the growth of a $(5,8)$-tree may be interrupted by any of the three other types of trees. On the other hand, $(2,7)$-trees and $(3,6)$-trees have shorter paths, and in fact when such trees are grown *no* queries to $\mathrm{F}(\cdot,\cdot)$ are made, which means that such trees never see their growth interrupted by anything. In other words, a $(3,6)$- or $(2,7)$-tree will only appear as the last tree in the tree stack, if at all.

Overall, it is imperative that pending queries be kept *unsampled* until the relevant tree becomes stable, and is adapted. In particular, the simulator must not overwrite the pending queries of trees lower down in the tree stack while working on the current tree.

In fact, and like [11], our simulator *cannot* overwrite pending queries because it keeps a list of all pending queries, and aborts rather than overwrite a pending query. Nonetheless, one must show that the chance of such an event is negligible. The analysis of this bad event is lengthy but also straightforward. Briefly, this bad event can only occur if ready and non-ready paths arrange to form a certain type of cycle, and the occurence of such cycles can be reduced to the occurence of a few different "local" bad events whose (negligible) probabilities are easily bounded.

The Termination Argument. The basic idea of Coron et al.'s [9] termination argument (which only needs to be lightly adapted for our use) is that each path detected in one of the outer detect zones is associated with high probability to a P-query previously made by the distinguisher. Since the distinguisher only has q queries total, this already implies that the number of path completions triggered by the outer detect zones is at most q with high probability.

Secondly, whenever a path is triggered by one of the middle detect zones, this path completion does not add any new entries to the tables F_4, F_5. Hence, only two mechanisms add entries to the tables F_4 and F_5: queries directly made by the distinguisher and path completions triggered by the outer detect zones. Each of these accounts for at most q table entries in each of F_4, F_5, so that the tables F_4, F_5 do not exceed size $2q$. But *every* completed path corresponds to a *unique* pair of entries in F_4, F_5. (I.e., no two completed paths have the same x_4 *and* the same x_5.) So the total number of paths ever completed is at most $(2q)^2 = 4q^2$.

Further Details. A more technical description of the simulator and the pseudocode are given in the full version of this paper [12].

4 Proof Overview

In this section we give an overview of the proof for Theorem 1, using the simulator described in Section 3 as the indifferentiability simulator. Details of the proof are given in the full version [12].

In order to prove that our simulator successfully achieves indifferentiability as defined by Definition 1, we need to upper bound the advantage of any distinguisher, as well as the time and query complexity of the simulator; the latter is related to the *termination argument* for our simulator, already sketched at the end of the last section.

Game Sequence. Our proof uses a sequence of five games, $\mathrm{G}_1, \ldots, \mathrm{G}_5$, with G_1 being the simulated world and G_5 being the real world. Every game offers the same interface to the distinguisher, consisting of functions F, P and P^{-1}.

In the simulated world G_1, P and P^{-1} are answered by an oracle according to a random permutation and its inverse; the simulator, as described in Section 3, is in charge of answering distinguisher queries to F.

The randomness used in the experiment is read from *explicit random tapes*, similar to [20]. In particular, the random permutation is encoded by a tape $p : \{0,1\}^{2n} \to \{0,1\}^{2n}$ (whose inverse is accessible via p^{-1}). The simulator has access to 8 tapes $f_1, \ldots, f_8$, which are independent uniform random mappings from $\{0,1\}^n$ to $\{0,1\}^n$; when randomly sampling the value of a query (i, x_i), the simulator reads the value of $f_i(x_i)$ and sets $F_i(x_i) \leftarrow f_i(x_i)$. Since each query is sampled at most once, the tape entry hasn't been read before and its value is uniformly and independently distributed in $\{0,1\}^n$. Note that each f_i encodes a random function, and f_i will be used as the i-th round function in the real world

G_5.

A brief synopsis of the changes that occur in the games is as follows:

In G_2: The simulator triggers a path at the outer detect zones only if the distinguisher has issued the permutation query in the path.

Recall that the outer detect zones consist of rounds 1, 7, 8 or of rounds 1, 2, 8; to check whether three queries in an outer detect zone are in the same path, the simulator has to call $P(x_0, x_1)$ or $P^{-1}(x_8, x_9)$ in G_1. In G_2, instead of calling $P^{(-1)}$, the simulator performs a "peek" operation that accesses the query history of the permutation oracle[10]; the path is triggered only if the permutation query is in the history and the three queries are in the same path. Then, the simulator queries P or P^{-1} only when completing a triggered path; therefore, if a permutation query is issued by the simulator, the path containing the permutation query must have been completed (and cannot be triggered again).

Although the change may result in "false negatives" when detecting triggered paths, such false negatives remain unlikely as long as the simulator is efficient.

In G_3: The simulator adds a number of checks that may cause it to abort in places where it did not abort in G_2. Some of these checks cannot be included in G_1 because they also (like the modifications in G_2) involve the simulator "peeking" at the distinguisher's queries to the permutation oracle.

The checks in G_3 are added to catch "bad events" at the earliest possible stage, i.e., at the moment after the relevant randomness has been sampled. If these checks pass, we can show that the simulator will not abort at further key points down the line, such as by attempting to overwrite an existing entry $F_i(x_i)$ or by attempting to overwrite a pending query. (I.e., the checks in G_3 are sufficient conditions for the execution to maintain a "good" structure.)

In G_4: The most important transition occurs in this game, as the oracles P, P^{-1} no longer rely on the random permutation tape $p : \{0,1\}^{2n} \to \{0,1\}^{2n}$, but instead evaluate an 8-round Feistel network using the random tapes $f_1, \ldots, f_8$ (i.e., the same ones used by the simulator) as round functions. Apart from this change to P, P^{-1}, the simulator remains identical.

In G_5: This is the real world, meaning that $F(i, x)$ directly returns the value $f_i(x)$. In particular G_5 never aborts, unlike the previous four games.

Definition 2. The *advantage* of a distinguisher D at distinguishing games G_i and G_j is defined as

$$\Delta_D(G_i, G_j) = \Pr_{G_i}[D^{F,P,P^{-1}} = 1] - \Pr_{G_j}[D^{F,P,P^{-1}} = 1] \tag{4}$$

where the probabilities are taken over the coins of the relevant game as well as over D's coins, if any.

As G_5 never aborts, and as the distinguisher's job is to maximize

$$\Delta_D(G_1, G_5) = \Pr_{G_1}[D^{F,P,P^{-1}} = 1] - \Pr_{G_5}[D^{F,P,P^{-1}} = 1],$$

10 This operation cannot be included in G_1 as the original simulator is not allowed to see the distinguisher's permutation queries.

we can assume without loss of generality that D outputs 1 if the game aborts. In particular, since G_2 is identical to G_3 except for the possibility that G_3 may abort when G_2 does not, we then have

$$\Delta_D(\mathrm{G}_2, \mathrm{G}_5) \leq \Delta_D(\mathrm{G}_3, \mathrm{G}_5)$$

so we can upper bound $\Delta_D(\mathrm{G}_1, \mathrm{G}_5)$ as

$$\begin{aligned}\Delta_D(\mathrm{G}_1, \mathrm{G}_5) &\leq \Delta_D(\mathrm{G}_1, \mathrm{G}_2) + \Delta_D(\mathrm{G}_2, \mathrm{G}_5)\\ &\leq \Delta_D(\mathrm{G}_1, \mathrm{G}_2) + \Delta_D(\mathrm{G}_3, \mathrm{G}_5)\\ &\leq \Delta_D(\mathrm{G}_1, \mathrm{G}_2) + \Delta_D(\mathrm{G}_3, \mathrm{G}_4) + \Delta_D(\mathrm{G}_4, \mathrm{G}_5)\end{aligned}$$

by the triangle inequality. Hence, the proof focuses on the individual transitions $\mathrm{G}_1 \to \mathrm{G}_2$, $\mathrm{G}_3 \to \mathrm{G}_4$ and $\mathrm{G}_4 \to \mathrm{G}_5$.

G_1-G_2 Transition. The two games (with the same random tapes) are visibly different only if a path is triggered in G_1 but not in G_2, i.e., when a path triggered by the outer detect zones in G_1 contains a permutation query that hasn't been issued in G_2. In this case (and assuming that G_1 and G_2 are being "run" simultaneously on the same distinguisher D, with the same random tapes $f_1, \ldots, f_8 : \{0,1\}^n \to \{0,1\}^n$ and $p : \{0,1\}^{2n} \to \{0,1\}^{2n}$) we say that G_1 and G_2 *diverge*.

More precisely, one can show that divergence occurs if and only if the simulator makes a certain call of the form[11] "$\mathrm{CheckP}^+(x_1, x_2, x_8)$" such that (i) $p(x_0, x_1)$ is unread in G_2, where[12] $x_0 = F_1(x_1) \oplus x_2$, and (ii) the first n bits of the unread value $p(x_0, x_1)$ are equal to x_8. In fact, this notion can be defined with respect to the execution of G_2 *alone* (i.e., without examining the execution of G_1) which then makes it straightforward to examine. In more detail, the probability of divergence of occuring in G_2 is obtained by a simple union bound over all calls to the CheckP^+/CheckP^- procedures, where the number of such calls is upper bounded thanks to the (previously established) simulator efficiency. (Indeed, the proof actually starts off by arguing various efficiency metrics in G_1 and G_2.)

G_3-G_4 Transition. For this transition, a randomness mapping argument is used, as introduced by [20]. We also take advantage of some refinements introduced by [1, 14]. Specifically, following [1], we use "footprints" and eschew the use of a "two-way random function"; and following [14], we additively cancel the probabilities of abort in G_4 in separate transitions from G_3 to G_4 and from G_4 to G_5 in order to avoid double-counting these probabilities.

As usual, the randomness mapping argument consists of two steps: bounding the abort probability in G_3, and mapping the randomness of non-aborting executions of G_3 to the randomness of "matching" executions of G_4. Bounding the abort probability in G_3 is the more technically involved of the two steps.

[11] Or a call "$\mathrm{CheckP}^-(x_1, x_7, x_8)$" subject to symmetric conditions.

[12] The fact that $\mathrm{CheckP}^+(x_1, x_2, x_8)$ is called in the first place implies that $F_1(x_1) \neq \bot$ in either game.

A small novelty that we introduce also concerns the randomness mapping argument. Specifically, a randomness map needs to be defined with respect to a distinguisher D that "completes" all paths (that contain a permutation query issued by D). Making the assumption that D completes all paths is without loss of generality, but costs a multiplicative factor in the number of queries that is equal to the number of rounds—potentially annoying! However, we note that if D is allowed q queries to each of its $r + 1$ oracles (the permutation plus the r rounds functions), then the assumption that D completes all paths can be made at the cost of only doubling the number of D's queries. Moreover, there is no real cost in giving D the power to query each of its oracles q times, since most proofs effectively allow this anyway.

G_4-G_5 TRANSITION. A non-aborting execution of G_4 is identical to an execution of G_5 with the same random tape, so the advantage in distinguishing between these two games is upper bounded by the simulator's abort probability in G_4.

ACKNOWLEDGMENTS. Yuanxi Dai was supported by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003. John Steinberger was funded by National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61373002, 61361136003, and by the China Ministry of Education grant number 20121088050.

References

1. Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the indifferentiability of key-alternating ciphers. In *Advances in Cryptology—CRYPTO 2013* (volume 1), pages 531–550.
2. Mihir Bellare and Phillip Rogaway, Random oracles are practical: A paradigm for designing efficient protocols, In Proceedings of the 1st ACM Conference on Computer and Communications Security (1993), pages 62–73.
3. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In EUROCRYPT 2008, LNCS 4965, pages 181–197, 2008.
4. Ran Canetti, Security and composition of multi-party cryptographic protocols. Journal of Cryptology, 13(1): 143–202, 2000.
5. Ran Canetti, Universally composable security: A new paradigm for cryptographic protocols. In Proc. 42nd IEEE Symposium on Foundations of Computer Science (FOCS), pages 136–145, 2001.
6. Shan Chen and John Steinberger. Tight Security Bounds for Even-Mansour Ciphers, EUROCRYPT 2014, LNCS 8441, pp. 327–350, 2014.
7. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In Victor Shoup, editor, *Advances in Cryptology—CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer-Verlag, 14–18 August 2005.
8. Jean-Sébastien Coron, Yevgeniy Dodis, Avradip Mandal, and Yannick Seurin. A domain extender for the ideal cipher. To appear in the *Theory of Cryptography Conference* (TCC), February 2010.

9. Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In CRYPTO 2008, LNCS 5157, pages 1–20, 2008.
10. Dana Dachman-Soled, Jonathan Katz, and Aishwarya Thiruvengadam, 10-Round Feistel is Indifferentiable from an Ideal Cipher. To appear in EUROCRYPT 2016, Technical Report 2015/876, IACR eprint archive, 2015.
11. Yuanxi Dai and John Steinberger, Indifferentiability of 10-Round Feistel Networks. Technical Report 2015/874, IACR eprint archive, 2015.
12. Yuanxi Dai and John Steinberger, Indifferentiability of 8-Round Feistel Networks. Technical Report 2015/1069, IACR eprint archive, 2015.
13. Yevgeniy Dodis and Prashant Puniya, On the Relation Between the Ideal Cipher and the Random Oracle Models. Proceedings of TCC 2006, 184–206.
14. Yevgeniy Dodis, Tianren Liu, Martijn Stam, and John Steinberger, On the Indifferentiability of Confusion-Diffusion Networks. Technical Report 2015/680, IACR eprint archive, 2015.
15. Yevgeniy Dodis, Leonid Reyzin, Ronald L. Rivest, and Emily Shen. Indifferentiability of permutation-based compression functions and tree-based modes of operation, with applications to md6. In Orr Dunkelman, editor, *Fast Software Encryption: 16th International Workshop, FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 104–121. Springer-Verlag, 22–25 February 2009.
16. Yevgeniy Dodis, Thomas Ristenpart, John Steinberger, and Stefano Tessaro, To Hash or Not to Hash Again? (In)differentiability Results for H^2 and HMAC. Advances in CryptologyCRYPTO 2012, LNCS 7417, pp. 348–366.
17. Hörst Feistel. Cryptographic coding for data-bank privacy. IBM Technical Report RC-2827, March 18 1970.
18. Hörst Feistel, William A. Notz, J. Lynn Smith. Some Cryptographic Techniques for Machine-to-Machine Data Communications. IEEE proceedings, **63**(11), pages 1545–1554, 1975.
19. Amos Fiat and Adi Shamir, How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, Advances in Cryptology CRYPTO 86, volume 263 of LNCS, pages 186–194.
20. Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The equivalence of the random oracle model and the ideal cipher model, revisited. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 89–98. ACM, 2011.
21. Rodolphe Lampe and Yannick Seurin. How to construct an ideal cipher from a small set of public permutations. ASIACRYPT 2013, LNCS 8269, 444–463. Springer, 2013.
22. M. Luby and C. Rackoff. How to construct pseudorandom permutations and pseudorandom functions. SIAM Journal on Computing, 17(2):373–386, April 1988.
23. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *First Theory of Cryptography Conference — TCC 2004*, LNCS 2951, 21–39. Springer, 2004.
24. M. Naor and O. Reingold, On the construction of pseudorandom permutations: Luby-Rackoff revisited, J. of Cryptology, 1999. Preliminary Version: STOC 1997.
25. Jacques Patarin, Security of balanced and unbalanced Feistel Schemes with Linear Non Equalities. Technical Report 2010/293, IACR eprint arxiv.

26. Birgit Pfitzmann and Michael Waidner, Composition and integrity preservation of secure reactive systems. In 7th ACM Conference on Computer and Communications Security, pages 245–254. ACM Press, 2000.
27. Birgit Pfitzmann and Michael Waidner, A model for asynchronous reactive systems and its application to secure message transmission. Technical Report 93350, IBM Research Division, Zürich, 2000.
28. David Pointcheval and Jacques Stern, Security Proofs for Signature Schemes. EUROCRYPT 1996, LNCS 1070, pp. 387–398.
29. Thomas Ristenpart and Hovav Shacham and Thomas Shrimpton, Careful with Composition: Limitations of the Indifferentiability Framework. EUROCRYPT 2011, LNCS 6632, pp. 487–506.
30. Phillip Rogaway, Viet Tung Hoang, On Generalized Feistel Networks. EUROCRYPT 2010, LNCS 6223, pp. 613–630, Springer, 2010.
31. Yannick Seurin, *Primitives et protocoles cryptographiques à sécurité prouvée*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, France, 2009.
32. Yannick Seurin, A Note on the Indifferentiability of the 10-Round Feistel Construction. Available from `yannickseurin.free.fr/pubs/Seurin_note_ten_rounds.pdf`.
33. R. Winternitz. A secure one-way hash function built from DES. *Proceedings of the IEEE Symposium on Information Security and Privacy*, pp. 88–90. IEEE Press, 1984.