# A Note on the Indifferentiability of the 10-Round Feistel Construction

Yannick Seurin

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

March 2011, updated September 2015

**Abstract.** Holenstein *et al.* (STOC 2011) have shown that the Feistel construction with fourteen rounds and public random round functions is indifferentiable from a random permutation. In the same paper, they pointed out that a previous proof for the 10-round Feistel construction by Seurin (PhD thesis) was flawed. However, they left open the question of whether the proof could be patched (leaving hope that the simulator described by Seurin could still be used to prove indifferentiability of the 10-round Feistel construction). In this note, we show that the proof *cannot* be patched (and hence that the simulator described by Seurin *cannot* be used to prove the indifferentiability of the 10-round Feistel construction) by describing a distinguishing attack that succeeds with probability close to one against this simulator. We stress that this does not imply that the 10-round Feistel construction is not indifferentiable from a random permutation (since our attack does not exclude the existence of a different simulator that would work).

## 1 Introduction

Indifferentiability has been introduced by Maurer, Renner and Holenstein [MRH04] as a way to formally study whether some construction $\mathcal{C}^{\boldsymbol{F}}$ based on some ideal primitive $\boldsymbol{F}$ "behaves" as some target ideal primitive $\boldsymbol{G}$. To show that $\mathcal{C}^{\boldsymbol{F}}$ is indifferentiable from $\boldsymbol{G}$, one has to exhibit an efficient simulator $\mathcal{S}$ with black-box access to $\boldsymbol{G}$ such that the two systems $(\mathcal{C}^{\boldsymbol{F}}, \boldsymbol{F})$ and $(\boldsymbol{G}, \mathcal{S}^{\boldsymbol{G}})$ are indistinguishable. Informally, the goal of the simulator is twofold: it must provide answers which are consistent with what the distinguisher can obtain by querying $\boldsymbol{G}$, without deviating too much from the distribution of answers of $\boldsymbol{F}$. When $\mathcal{C}^{\boldsymbol{F}}$ is indifferentiable from $\boldsymbol{G}$, a composition theorem ensures that any cryptosystem which is provably secure when used with the ideal primitive $\boldsymbol{G}$ remains provably secure when used with $\mathcal{C}^{\boldsymbol{F}}$, therefore enabling modular security proofs in idealized models.[1]

Soon after its introduction, the indifferentiability framework has been used by Coron *et al.* [CDMP05] to revisit the construction of a hash function from an ideal cipher: they showed that variants of the Merkle-Damgård domain extension method, when used with an ideal cipher in Davies-Meyer mode, are indifferentiable from a random oracle. This implies that any cryptosystem which is secure in the random oracle model can also be securely implemented in the ideal cipher model, in short that the ideal cipher model "implies" the random oracle model.

The other direction, namely whether it is possible to construct an ideal cipher from a random oracle, turned out to be harder to achieve. A natural candidate for this task is the Feistel construction. One round of this construction implements a permutation on $2n$ bits $\Psi^F$

[1] Restrictions have been later put forward regarding this somehow imprecise formulation of the composition theorem [RSS11].

from a function $F$ from $n$ bits to $n$ bits, and is defined as $\Psi^F(L, R) = (R, L \oplus F(R))$. Given $r$ round functions $(F_1, \ldots, F_r)$, the $r$-round Feistel construction is defined as

$$\Psi_r^{(F_1,\ldots F_r)} = \Psi^{F_r} \circ \ldots \circ \Psi^{F_1}.$$

In the following, we will simply denote $F$ the tuple $(F_1, \ldots, F_r)$ and $\Psi_r^F$ instead of $\Psi_r^{(F_1,\ldots,F_r)}$. A celebrated result by Luby and Rackoff [LR88] states the 3-round (resp. 4-round) Feistel construction yields a pseudorandom permutation (resp. a strong pseudorandom permutation) when the round functions are independent pseudorandom functions. It was conjectured quite early that a sufficient number of rounds would make the Feistel construction (with public random round functions) indifferentiable from a random permutation [CJP02, DP06]. To obtain a construction which is indifferentiable from an ideal cipher, it is then sufficient to prepend the key to the input to each round function.

Coron, Patarin, and Seurin [CPS08] published a first proof that the 6-round Feistel construction is indifferentiable from a random permutation. They also showed that at least six rounds are necessary to achieve indifferentiability by giving an attack for five rounds (we insist that the attack described in [CPS08] works *independently of the simulator*, so that it proves that the 5-round Feistel construction is *not indifferentiable* from a random permutation). Slightly later, in an effort to simplify the somehow complex proof of [CPS08], Seurin gave in his PhD thesis [Seu09] a similar but less intricate proof for the 10-round Feistel construction. Unfortunately, problems were detected in both proofs by Holenstein, Künzler, and Tessaro [Kün09, HKT11]. Holenstein *et al.* were even able to give an attack against the simulator for six rounds of [CPS08], therefore excluding any hope to patch the proof of [CPS08] without beforehand modifying the simulator. We emphasize that the distinguisher described by Holenstein *et al.* works only for the simulator of [CPS08], not for *any simulator*, so that it does not prove that the 6-round Feistel construction is not indifferentiable from a random permutation. To the best of our knowledge, it is still a reasonable conjecture that six rounds are sufficient to achieve indifferentiability, though simply describing a plausible simulator — not to say proving that it works— remains an open problem. Regarding the proof for ten rounds by Seurin [Seu09], Holenstein *et al.* could only point out the flaw in the proof, but they did not describe an attack as in the 6-round case. Hence, it was an open problem to establish whether the simulator described in [Seu09] actually worked (*i.e.* could be used to prove indifferentiability for ten rounds), or whether it could be attacked. The goal of this note is to describe an attack against the simulator of [Seu09]. Similarly to the distinguisher described by Holenstein *et al.* in the 6-round case, our distinguisher is tailored for the simulator described in [Seu09], and hence does not prove that the 10-round Feistel construction is not indifferentiable from a random permutation.

In [HKT11], Holenstein *et al.* also gave a proof that the 14-round Feistel construction is indifferentiable from a random permutation. Their simulator is very similar to the one described in [Seu09], the only difference being the use of four "buffer" rounds surrounding the adaptation rounds (see later for a more detailed account of the role of each round during simulation). These four buffer rounds do not play any role in the simulation strategy, but play on the other hand a crucial role when proving that the simulator "works". By removing these four rounds (and leaving the simulator otherwise unchanged), one exactly recovers the simulator for ten rounds described in [Seu09].

Since we aim at keeping this note short, we omit any formal definition of indifferentiability and refer the interested reader to [MRH04, CDMP05, CPS08, HKT11] for more details. In the following, we start with a description of the simulator for the 10-round Feistel construction as defined in [Seu09], and then describe and analyze the attack against this simulator.

Update (September 16, 2015). On September 8, 2015, two independent papers proposing a new proof of the indifferentiability of the 10-round Feistel construction were sent to the IACR ePrint archive: one by Dai and Steinberger [DS15], the other by Dachman-Soled, Katz, and Thiruvengadam [DSKT15].

## 2 Description of the Simulator for Ten Rounds

### 2.1 Informal Description

We start by giving a high-level overview of how the simulator $\mathcal{S}$ works (see also Figure 1). The simulator offers an interface $\texttt{Query}(i, x)$ that can be accessed by the distinguisher, where $i$ names which round function is queried and $x$ is the actual value which is queried. The simulator maintains hash tables $F_1, \ldots, F_{10}$ which map entries $x \in \{0,1\}^n$ to values $y \in \{0,1\}^n$ for each simulated round function. Initially, these hash tables are empty, meaning that round function values $F_i(x)$ are undefined for all $i \in \{1, \ldots, 10\}$ and all $x \in \{0,1\}^n$. The hash tables are then modified during the execution of the simulator. When the simulator receives a query $(i, x)$, it looks in hash table $F_i$ whether $F_i(x)$ is already defined. If this is the case, it simply returns the corresponding answer. Otherwise, it draws $F_i(x)$ uniformly at random in $\{0,1\}^n$. Moreover, for some specific values of $i$ (namely $i = 2$, 5, 6, or 9), it implements a "chain detection" mechanism followed by a "chain completion" mechanism to ensure consistency with the random permutation $\boldsymbol{P}$. The chain detection mechanism first puts in a queue Queue tuples $(x_k, x_{k+1}, k, \ell)$. The first three elements $(x_k, x_{k+1}, k)$ specify which chain must be completed, while the fourth element $\ell \in \{3, 7\}$, specifies which round functions will be "adapted" to ensure consistency with $\boldsymbol{P}$ ($F_3$ and $F_4$ when $\ell = 3$, or $F_7$ and $F_8$ when $\ell = 7$).

In more details, for a query $(2, x_2)$, the simulator checks for all values $(x_1, x_9, x_{10}) \in F_1 \times F_9 \times F_{10}$ whether $\boldsymbol{P}(x_2 \oplus F_1(x_1), x_1) = (x_{10}, x_9 \oplus F_{10}(x_{10}))$, and enqueues the tuple $(x_1, x_2, 1, 3)$ if this holds. The behavior for a query $(9, x_9)$ is symmetric, but in that case one has $\ell = 7$: namely, for all values $(x_1, x_2, x_{10}) \in F_1 \times F_2 \times F_{10}$, the simulator whether $\boldsymbol{P}(x_2 \oplus F_1(x_1), x_1) = (x_{10}, x_9 \oplus F_{10}(x_{10}))$, and enqueues the tuple $(x_1, x_2, 1, 7)$ if this holds.

When $i = 5$ or 6, the simulator detects and enqueues all newly created pairs $(x_5, x_6) \in F_5 \times F_6$. For each such pair it enqueues $(x_5, x_6, 5, \ell)$, where $\ell = 3$ if $i = 5$, and $\ell = 7$ if $i = 6$.

Once the simulator has enqueued all newly created chains, it starts completing them by dequeuing tuples $(x_k, x_{k+1}, k, \ell)$ from Queue. For each such tuple, it "moves" forward and backward in the Feistel network starting from values $x_k$ and $x_{k+1}$, defining missing round function values at random, and making a call to $\boldsymbol{P}/\boldsymbol{P}^{-1}$ to "wrap around", until only $F_\ell(x_\ell)$ and $F_{\ell+1}(x_{\ell+1})$ remain undefined. These two values are then "adapted" by the simulator to ensure consistency with the random permutation by setting:

$$\begin{cases} F_\ell(x_\ell) := x_{\ell-1} \oplus x_{\ell+1} \\ F_{\ell+1}(x_{\ell+1}) := x_\ell \oplus x_{\ell+2}. \end{cases}$$

While completing a chain, new chains might be created due to the additional values put in the hash tables $F_i$. These new chains are enqueued, and the simulator keeps dequeuing chains

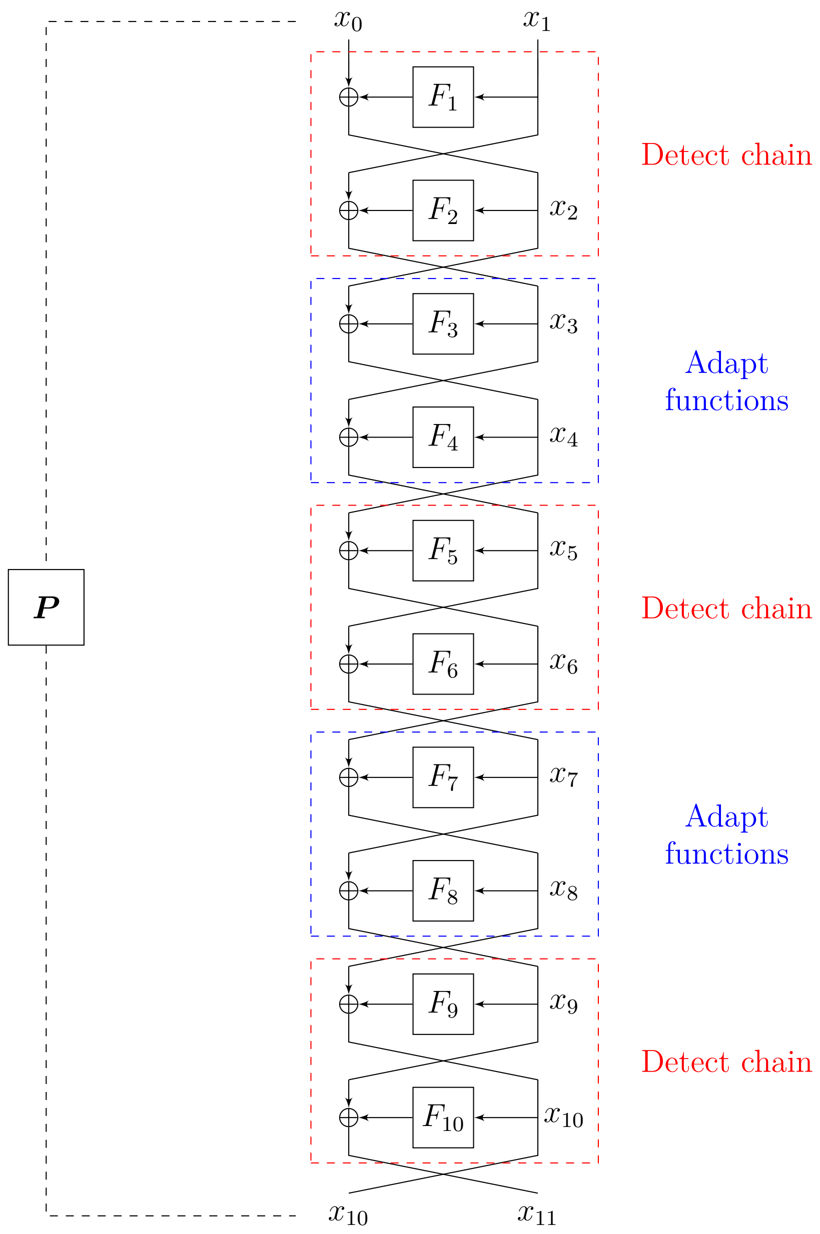

**Fig. 1.** Detection and adaptation zones used by the simulator for the 10-round Feistel construction.

until the queue is empty. At this point, it returns the answer to the original query of the distinguisher.

The simulator also maintains a set COMPLETED containing previously completed chains to avoid completing the same chain twice.

For completeness, we give a detailed description of the simulator (adapted from [HKT11]) in the next section.

## 2.2 Definition of the Simulator in Pseudocode

We now describe the simulator in pseudocode. The formalism is very similar to the one of [HKT11]. Procedure `Query`$(i, x)$ is the only "public" procedure that can be queried by the distinguisher.

```
**Simulator**

hash tables $F_1, \dots, F_{10}$ (initially empty)
queue QUEUE (initially empty)
set COMPLETED (initially empty)

**procedure** Query($i$,$x$):
    $y :=$ InQuery($i, x$)
    **while** QUEUE $\neq \emptyset$ **do**
        $(x_k, x_{k+1}, k, \ell) :=$ QUEUE.Dequeue()
        **if** $(x_k, x_{k+1}, k) \notin$ COMPLETED **then**
            \\ complete the chain
            $(x_{\ell-1}, x_\ell) :=$ EvalForward($x_k, x_{k+1}, k, \ell - 1$)
            $(x_{\ell+1}, x_{\ell+2}) :=$ EvalBackward($x_k, x_{k+1}, k, \ell + 1$)
            Adapt($x_{\ell-1}, x_\ell, x_{\ell+1}, x_{\ell+2}, \ell$)
            \\ add corresponding partial chains to set COMPLETED
            $(x_1, x_2) :=$ EvalBackward($x_k, x_{k+1}, k, 1$)
            $(x_5, x_6) :=$ EvalForward($x_1, x_2, 1, 5$)
            COMPLETED := COMPLETED $\cup \{(x_1, x_2, 1), (x_5, x_6, 5)\}$
    **return** $y$

**procedure** InQuery($i$,$x$):
    **if** $x \notin F_i$ **then**
        $F_i(x) \leftarrow_\$ \{0, 1\}^n$
        **if** $i \in \{2, 5, 6, 9\}$ **then**
            EnqueueNewChains($i, x$)
    **return** $F_i(x)$

**procedure** Adapt($x_{\ell-1}$,$x_\ell$,$x_{\ell+1}$,$x_{\ell+2}$,$\ell$):
    $F_\ell(x_\ell) := x_{\ell-1} \oplus x_{\ell+1}$
    $F_{\ell+1}(x_{\ell+1}) := x_\ell \oplus x_{\ell+2}$

**procedure** EnqueueNewChains($i$,$x$):
    **if** $i = 2$ **then**
        **for all** $(x_1, x_2, x_9, x_{10}) \in F_1 \times \{x\} \times F_9 \times F_{10}$ **do**
            **if** $\boldsymbol{P}(x_2 \oplus F_1(x_1), x_1) = (x_{10}, x_9 \oplus F_{10}(x_{10}))$ **then**
                Queue.Enqueue$(x_1, x_2, 1, 3)$
    **else if** $i = 9$ **then**
        **for all** $(x_1, x_2, x_9, x_{10}) \in F_1 \times F_2 \times \{x\} \times F_{10}$ **do**
            **if** $\boldsymbol{P}(x_2 \oplus F_1(x_1), x_1) = (x_{10}, x_9 \oplus F_{10}(x_{10}))$ **then**
                Queue.Enqueue$(x_1, x_2, 1, 7)$
    **else if** $i = 5$ **then**
        **for all** $(x_5, x_6) \in \{x\} \times F_6$ **do**
            Queue.Enqueue$(x_5, x_6, 5, 3)$
    **else if** $i = 6$ **then**
        **for all** $(x_5, x_6) \in F_5 \times \{x\}$ **do**
            Queue.Enqueue$(x_5, x_6, 5, 7)$

**procedure** EvalForward$(x_k, x_{k+1}, k, \ell)$:
    **while** $k \neq \ell$ **do**
        **if** $k = 10$ **then**
            $(x_0, x_1) := \boldsymbol{P}^{-1}(x_{10}, x_{11})$
            $k := 0$
        **else**
            $x_{k+2} := x_k \oplus$ InQuery$(k + 1, x_{k+1})$
            $k := k + 1$
    **return** $(x_\ell, x_{\ell+1})$

**procedure** EvalBackward$(x_k, x_{k+1}, k, \ell)$:
    **while** $k \neq \ell$ **do**
        **if** $k = 0$ **then**
            $(x_{10}, x_{11}) := \boldsymbol{P}(x_0, x_1)$
            $k := 10$
        **else**
            $x_{k-1} := x_{k+1} \oplus$ InQuery$(k, x_k)$
            $k := k - 1$
    **return** $(x_\ell, x_{\ell+1})$
```

## 3 An Attack Against the Simulator for Ten Rounds

### 3.1 Description of the Attack

We describe a distinguisher $\mathcal{D}$ which interacts generically with oracles $(P, F)$, where $F = (F_1, \ldots, F_{10})$, which might be either $(\Psi_{10}^{\boldsymbol{F}}, \boldsymbol{F})$ or $(\boldsymbol{P}, \mathcal{S}^{\boldsymbol{P}})$. We will see that this distinguisher makes the simulator overwrite a value with probability one when interacting with $(\boldsymbol{P}, \mathcal{S}^{\boldsymbol{P}})$. See also Figure 2 for an illustration of the attack.

The distinguisher first chooses two arbitrary values $x_1$ and $x_2$, and evaluates the chain $(x_1, x_2, 1)$ backward up to the input to $F_5$, defining the following values (by making appropriate queries to $F_1$, $P$, …, $F_6$):

$$\begin{aligned}
x_0 &:= x_2 \oplus F_1(x_1)\\
(x_{10}, x_{11}) &:= P(x_0, x_1)\\
x_9 &:= x_{11} \oplus F_{10}(x_{10})\\
x_8 &:= x_{10} \oplus F_9(x_9)\\
x_7 &:= x_9 \oplus F_8(x_8)\\
x_6 &:= x_8 \oplus F_7(x_7)\\
x_5 &:= x_7 \oplus F_6(x_6).
\end{aligned}$$

Then it chooses an arbitrary value $\bar{x}_1 \neq x_1$, and evaluates the chain $(\bar{x}_1, x_2, 1)$ backward up to the input to $F_8$, defining the following values (by making appropriate queries to $F_1$, $P$, …, $F_9$):

$$\begin{aligned}
\bar{x}_0 &:= x_2 \oplus F_1(\bar{x}_1)\\
(\bar{x}_{10}, \bar{x}_{11}) &:= P(\bar{x}_0, \bar{x}_1)\\
\bar{x}_9 &:= \bar{x}_{11} \oplus F_{10}(\bar{x}_{10})\\
\bar{x}_8 &:= \bar{x}_{10} \oplus F_9(\bar{x}_9).
\end{aligned}$$

Then it defines

$$x'_5 := x_5 \oplus x_1 \oplus \bar{x}_1,$$

as well as:

$$x'_4 := x_6 \oplus F_5(x'_5),$$

by making a query to $F_5$, and evaluates the chain $(x'_4, x_5, 4)$ backward, defining the following values (by making appropriate queries to $F_4$, …, $F_9$):

$$\begin{aligned}
x'''_3 &:= x_5 \oplus F_4(x'_4)\\
x'''_2 &:= x'_4 \oplus F_3(x'''_3)\\
x'''_1 &:= x'''_3 \oplus F_2(x'''_2)\\
x'''_0 &:= x'''_2 \oplus F_1(x'''_1)\\
(x'''_{10}, x'''_{11}) &:= P(x'''_0, x'''_1)\\
x'''_9 &:= x'''_{11} \oplus F_{10}(x'''_{10})\\
x'''_8 &:= x'''_{10} \oplus F_9(x'''_9).
\end{aligned}$$

Finally, the distinguisher checks whether evaluating the partial chain $(x'_5, x'_6, 5)$ is consistent with the permutation $P$. In more details, it computes the input $(x'_0, x'_1)$ and the output $(x'_{10}, x'_{11})$ corresponding to $(x'_5, x'_6, 5)$ by making appropriate queries to the $F_i$'s, and checks whether $P(x'_0, x'_1) = (x'_{10}, x'_{11})$. If this holds, it returns 1, otherwise it returns 0.

## 3.2 Analysis of the Attack

First, it is clear that the distinguisher always outputs 1 when interacting with $(\Psi_{10}^{\boldsymbol{F}}, \boldsymbol{F})$ since the partial chain $(x'_5, x'_6, 5)$ is always consistent with the permutation $\Psi_{10}^{\boldsymbol{F}}$ in this case. We will

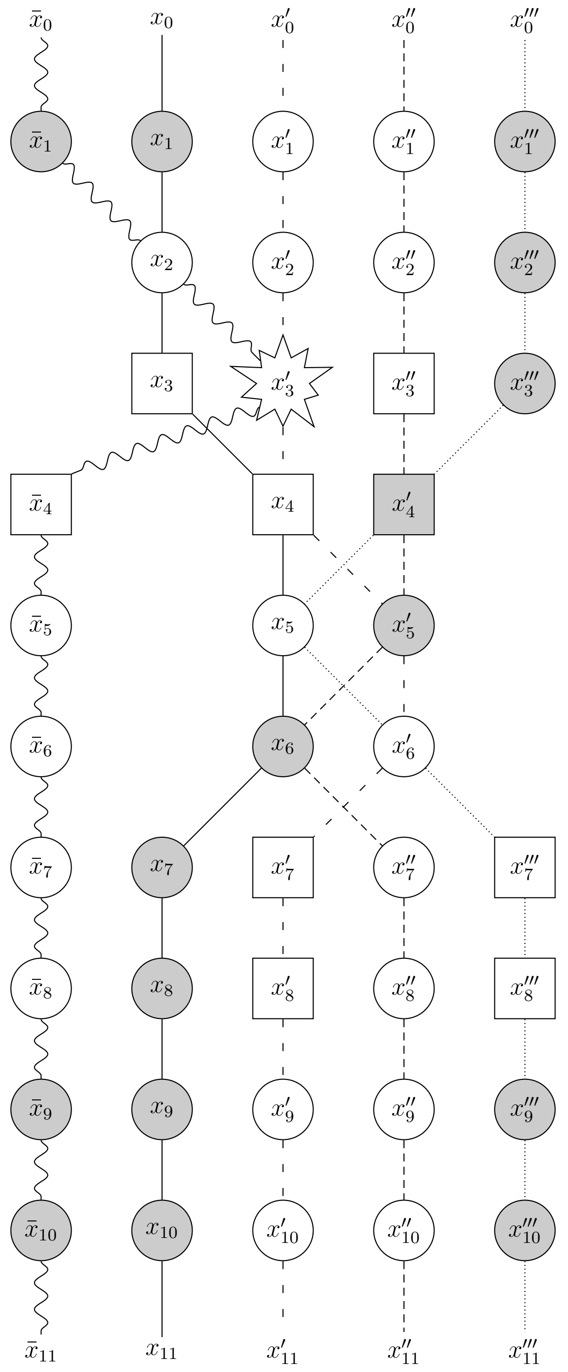

**Fig. 2.** The attack on the simulator for the 10-round Feistel construction. Values in circles are those for which $F_i(x)$ is randomly set by the simulator, while values in squares are those such that the value $F_i(x)$ is adapted by the simulator. Grayed values are those that are queried by the distinguisher up to query $F_9(x'''_9)$: queries needed to check consistency of the partial chain $(x'_5, x'_6, 5)$ are not shown. The burst for value $x'_3$ indicates that the simulator overwrites the corresponding entry during the attack.

now see that it outputs 1 only with negligible probability when interacting with $(\boldsymbol{P}, \mathcal{S}^{\boldsymbol{P}})$. For this, we will show that the simulator overwrites (with probability one) the entry corresponding to $F_3$ in the computation path corresponding to $(x'_5, x'_6, 5)$ (for later reference, this value will be denoted $x'_3$). More precisely, at some point during the simulation, evaluating partial chain $(x'_5, x'_6, 5)$ is consistent with the random permutation (namely immediately after the simulator has adapted this partial chain). Later in the execution, the simulator overwrites $F_3(x'_3)$ with some random value independent of the previous value, so that evaluating $(x'_5, x'_6, 5)$ after this point has only a probability $2^{-n}$ to be consistent with the random permutation.

In order to show the above claim that $F_3(x'_3)$ is overwritten during the execution, we now analyze the internal behavior of the simulator when faced with the sequence of queries of the distinguisher (see also Figure 2). Until query $(9, \bar{x}_9)$ included, the simulator just sets round function values randomly and does not detect any partial chain. Immediately after query $(5, x'_5)$, it enqueues $(x'_5, x_6, 5, 3)$, and dequeues it immediately. It defines values

$$\begin{aligned} x''_7 &:= x'_5 \oplus F_6(x_6) \\ x''_8 &= x_6 \oplus F_7(x''_7) \\ &\vdots \\ x''_3 &:= x''_1 \oplus F_2(x''_2), \end{aligned}$$

setting missing round function values randomly, and adapts $F_4(x'_4)$ and $F_3(x''_3)$. Partial chains $(x'_5, x_6, 5)$ and $(x''_1, x''_2, 1)$ are then added to Completed.

Then, all queries from $(3, x'''_3)$ to $(10, x'''_{10})$ are simply answered randomly, and no partial chain is detected. Finally, query $(9, x'''_9)$ causes partial chain $(x'''_1, x'''_2, 1, 7)$ to be enqueued and dequeued immediately. While completing this partial chain, the simulator randomly sets $F_5(x_5)$ where $x_5 = x'''_3 \oplus F_4(x'_4) = x_7 \oplus F_6(x_6)$, enqueues $(x_5, x_6, 5, 3)$, computes $x'_6 := x'_4 \oplus F_5(x_5)$, randomly sets $F_6(x'_6)$, enqueues $(x_5, x'_6, 5, 7)$ and $(x'_5, x'_6, 5, 7)$, and adapts round functions $F_7$ and $F_8$ for some irrelevant inputs $x'''_7$ and $x'''_8$. Partial chains $(x'''_1, x'''_2, 1)$ and $(x_5, x'_6, 5)$ are added to Completed.

Once this is done, the simulator dequeues $(x_5, x_6, 5, 3)$. It evaluates the Feistel backward up to the input value $x_2$ for $F_2$ (note that all round values needed are already defined in the corresponding hash tables), randomly sets $F_2(x_2)$, enqueues $(\bar{x}_1, x_2, 1, 3)$ (as well as $(x_1, x_2, 1, 3)$, but note that $(x_1, x_2, 1)$ is the chain that is being completed), and defines $x_3 := x_1 \oplus F_2(x_2)$ and $x_4 := x_6 \oplus F_5(x_5)$. Then it adapts $F_3(x_3) := x_2 \oplus x_4$ and $F_4(x_4) := x_3 \oplus x_5$, and adds $(x_1, x_2, 1)$ and $(x_5, x_6, 5)$ to Completed.

Then the simulator considers the next partial chain in the queue which is $(x_5, x'_6, 5, 7)$, but disregard it since $(x_5, x'_6, 5) \in$ Completed, and proceeds to dequeuing $(x'_5, x'_6, 5, 7)$. Observe that

$$x'_6 \oplus F_5(x'_5) = x_6 \oplus F_5(x_5) = x_4.$$

This is a direct consequence of the collision of $(x'_5, x_6, 5)$ and $(x_5, x'_6, 5)$ at round 4, namely

$$x'_4 = x_6 \oplus F_5(x'_5) = x'_6 \oplus F_5(x_5).$$

Note that $F_4(x_4)$ has been adapted while completing partial chain $(x_5, x_6, 5, 3)$. Hence, when evaluating the Feistel backward for $(x'_5, x'_6, 5)$, the simulator defines $x'_3 := x'_5 \oplus F_4(x_4)$, randomly sets $F_3(x'_3)$, and finishes completing the chain, adapting $F_7$ and $F_8$ for some irrelevant input values $x'_7$ and $x'_8$.

Finally, it dequeues $(\bar{x}_1, x_2, 1, 3)$. The crucial observation is that the completion of this chain will overwrite $F_3(x'_3)$. Indeed, one has $F_4(x_4) = x_3 \oplus x_5 = x_1 \oplus F_2(x_2) \oplus x_5$, so that

$$\begin{aligned} \bar{x}_1 \oplus F_2(x_2) &= \bar{x}_1 \oplus x_1 \oplus x_5 \oplus F_4(x_4) \\ &= x'_5 \oplus F_4(x_4) \\ &= x'_3. \end{aligned}$$

Hence, $F_3(x'_3)$ is overwritten during completion of $(\bar{x}_1, x_2, 1, 3)$ with some random value $x_2 \oplus \bar{x}_4$ independent of its previous value (indeed $\bar{x}_4$ is randomly set by the simulator when evaluating the chain $(\bar{x}_1, x_2, 1)$ backward), which renders $(x'_5, x'_6, 5)$ inconsistent with the random permutation.

The attack has been validated using the implementation of the simulator for ten rounds provided with [HKT11]. The Python script for the attack is given in Appendix A.

## 3.3 Where Does the Proof Strategy for Fourteen Rounds Break?

A substantial part of the intermediate results of [HKT11] leading to the proof that the 14-round Feistel construction is indifferentiable from a random permutation can be transposed to the 10-round case. In this last section, we informally explain where the proof strategy "breaks" for ten rounds.

We will use the following notation that has been introduced in [HKT11]. Fix hash tables $F_1, \dots, F_{10}$. Given a chain $C = (x_k, x_{k+1}, k)$, and $i \in \{1, \dots, 10\}$, $\texttt{val}^+_i(C)$ (resp. $\texttt{val}^-_i(C)$) is defined as the input value for round function $F_i$ obtained by moving forward (resp. backward) in the Feistel construction, or $\perp$ if at some point the computation stops because some value is missing in the hash tables (note that this might imply a call to $\boldsymbol{P}/\boldsymbol{P}^{-1}$, see [HKT11] for more details).

A crucial part of the proof of [HKT11] is to show that (for most executions) the simulator never overwrites entries in hash tables $F_\ell$ and $F_{\ell+1}$ when adapting a chain. For this, a first step is to show that just before the random assignment which leads to partial chain $C$ being enqueued to be adapted at position $\ell$, one has $\texttt{val}^+_\ell(C) = \perp$ and $\texttt{val}^-_{\ell+1}(C) = \perp$ (this is a consequence of Lemma 3.26 of [HKT11]). This also holds in the 10-round case. A second step is to show that between the moment where $C$ is enqueued, and the moment where it is dequeued, the completion of other chains does not lead to $\texttt{val}^+_\ell(C) \in F_\ell$ or $\texttt{val}^+_{\ell+1}(C) \in F_{\ell+1}$. This obviously does not hold in the 10-round case, as the attack described in this note demonstrates. In more details, there is some crucial lemma which holds in the 14-round case but not in the 10-round case: namely, in the 14-round case, on can show that during an adaptation assignment, for any partial chain $C$ which is not equivalent to the chain being completed and any $i \in \{1, \dots, 10\}$, $\texttt{val}^+_i(C)$ and $\texttt{val}^-_i(C)$ does not change (this is Lemma 3.23 (*b*) of [HKT11]). This does not hold in the 10-round case as we now analyze.

Consider the adaptation assignment $F_4(x_4) := x_3 \oplus x_5$ during completion of partial chain $A = (x_5, x_6, 5)$. Before this assignment occurs, partial chains $B = (x'_5, x'_6, 5)$ and $C = (\bar{x}_1, x_2, 1)$ have been enqueued to be adapted respectively at position 7 and 3. Just before the adaptation assignment $F_4(x_4) := x_3 \oplus x_5$, $\texttt{val}^+_3(C) = x'_3$ and $\texttt{val}^-_3(B) = \perp$. Just after the adaptation assignment, $\texttt{val}^-_3(B) = \texttt{val}^+_3(C) = x'_3$, *i.e.* partial chains $B$ and $C$ collide at round 3. In particular, $\texttt{val}^-_3(B)$ does not remain constant during the adaptation assignment $F_4(x_4) := x_3 \oplus x_5$. Since $B$ is dequeued before $C$, completion of chain $B$ sets $F_3(x'_3)$ randomly, and the simulator then overwrites $F_3(x'_3)$ when completing $C$.

A possible way to remedy this problem would be to modify the simulator in order to do some backtracking and "re-adapt" chains which have been previously completed if they are rendered inconsistent when some value is overwritten during a subsequent adaptation assignment. For example, to counter the attack described in this note, the simulator could erase values defined when completing partial chain $B = (x'_5, x'_6, 5)$, and complete it again, taking into account the new value of $F_3(x'_3) = x_2 \oplus \bar{x}_4$ that has been set when completing partial chain $C = (\bar{x}_1, x_2, 1)$. Whether such a strategy can be used to patch the simulator is out of the scope of this note.

## References

[CDMP05] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448. Springer, 2005.

[CJP02] Jean-Sébastien Coron, Antoine Joux, and David Pointcheval. Equivalence Between The Random Oracle Model and the Random Cipher Model. Dagstuhl Seminar, 2002.

[CPS08] Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In David Wagner, editor, *Advances in Cryptology - CRYPTO 2008*, volume 5157 of *LNCS*, pages 1–20. Springer, 2008.

[DP06] Yevgeniy Dodis and Prashant Puniya. On the Relation Between the Ideal Cipher and the Random Oracle Models. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography Conference - TCC 2006*, volume 3876 of *LNCS*, pages 184–206. Springer, 2006.

[DS15] Yuanxi Dai and John Steinberger. Feistel Networks: Indifferentiability at 10 Rounds. ePrint Archive, Report 2015/874, 2015. Available at http://eprint.iacr.org/2015/874.

[DSKT15] Dana Dachman-Soled, Jonathan Katz, and Aishwarya Thiruvengadam. 10-Round Feistel is Indifferentiable from an Ideal Cipher. ePrint Archive, Report 2015/876, 2015. Available at http://eprint.iacr.org/2015/876.

[HKT11] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The Equivalence of the Random Oracle Model and the Ideal Cipher Model, Revisited. In Lance Fortnow and Salil P. Vadhan, editors, *Symposium on Theory of Computing - STOC 2011*, pages 89–98. ACM, 2011. Full version available at http://arxiv.org/abs/1011.1264.

[Kün09] Robin Künzler. Are the random oracle and the ideal cipher models equivalent? Master's thesis, ETH Zurich, Switzerland, 2009.

[LR88] Michael Luby and Charles Rackoff. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal on Computing*, 17(2):373–386, 1988.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *Theory of Cryptography Conference- TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, 2004.

[RSS11] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with Composition: Limitations of the Indifferentiability Framework. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 487–506. Springer, 2011.

[Seu09] Yannick Seurin. *Primitives et protocoles cryptographiques à sécurité prouvée.* PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, France, 2009.

## A Python Script for the Attack on Ten Rounds

```
def tenRoundsAttack(sim, perm):
    x1 = randomInt()
    x2 = randomInt()
    print "Querying x1,1"
    x0 = x2^sim.query(x1, 1)
    print "Querying P(x0,x1)"
    (x10,x11) = perm.fwQuery(x0,x1)
    print "Querying x10,10"
    x9 = sim.query(x10, 10)^x11
    print "Querying x9,9"
    x8 = sim.query(x9, 9)^x10
    print "Querying x8,8"
    x7 = sim.query(x8, 8)^x9
    print "Querying x7,7"
    x6 = sim.query(x7, 7)^x8
    print "Querying x6,6"
    x5 = sim.query(x6, 6)^x7
    x1b = randomInt()
    print "Querying x1b,1"
    x0b = x2^sim.query(x1b, 1)
    print "Querying P(x0b,x1b)"
    (x10b,x11b) = perm.fwQuery(x0b,x1b)
    print "Querying x10b,10"
    x9b = sim.query(x10b, 10)^x11b
    print "Querying x9b,9"
    x8b = sim.query(x9b, 9)^x10b
    x5p = x5 ^ x1 ^ x1b
    print "Querying x5p,5"
    x4p = sim.query(x5p, 5)^x6
    print "Querying x4p,4"
    x3t = sim.query(x4p, 4)^x5
    print "Querying x3t,3"
    x2t = sim.query(x3t, 3)^x4p
    print "Querying x2t,2"
    x1t = sim.query(x2t, 2)^x3t
    print "Querying x1t,1"
    x0t = sim.query(x1t, 1)^x2t
    print "Querying P(x0t,x1t)"
    (x10t,x11t) = perm.fwQuery(x0t,x1t)
    print "Querying x10t,10"
    x9t = sim.query(x10t, 10)^x11t
    print "Querying x9t,9"
    x8t = sim.query(x9t, 9)^x10t
```