On the Indifferentiability of Fugue and Luffa

Rishiraj Bhattacharyya[1] and Avradip Mandal[2]

[1] Cryptology Research Group, Applied Statistics Unit, Indian Statistical Institute, Kolkata. rishi_r@isical.ac.in

[2] Université du Luxembourg, Luxembourg. avradip.mandal@uni.lu

Abstract. Indifferentiability is currently considered to be an important security notion for a cryptographic hash function to instantiate Random Oracles in different security proofs. In this paper, we prove indifferentiability of Fugue and Luffa, two SHA3 second round candidates. We also analyze the indifferentiability of a modified Luffa mode replacing multiple small permutations by a single large permutation.
Our technique is quite general and can be applicable to any sponge based design which uses affine function for message insertion. To the best of our knowledge, our result for Luffa is the first indifferentiability analysis of a mode of operation based on variable (more than two) number of small permutations.

1 Introduction

Design of cryptographic hash functions typically involves two steps. One constructs a fixed input length primitive (like permutation or compression function) $f : \{0,1\}^m \rightarrow \{0,1\}^n$ and applies a *domain extension* technique C^f to build the hash function. Merkle-Damgård technique along with its variants are the most popular choice for domain extension.

In the last decade, cryptographic hash functions have gained immense importance to instantiate a truly *Random Function* in cryptographic protocols. Although, previous results prove that no hash function can accurately instantiate a random function [6], one can hope to gain some confidence by using a structurally rigid construction in order to resist any generic attack. Indifferentiability of hash functions, introduced by Coron et. al. in [10] extending the result of [13], is the strongest and appropriate criteria to establish generic rigidity of the mode of a hash function. Informally, under the notion of indifferentiability, to prove rigidity of a domain extension technique C (assuming the underlying primitive f to be ideal), one has to design a simulator S that can simulate the underlying ideal primitive and still remain Consistent to a Random Oracle $\mathcal{R}$ with respect to concerned mode of operation. If no distinguisher can distinguish between the output distribution of (C^f, f) from that of $(\mathcal{R}, S^{\mathcal{R}})$ with probability ϵ, the hash function C^f is said to be indifferentiable from a Random Oracle (RO) with advantage ϵ.

In [10], Coron et. al. proved that Merkle-Damgård mode is insecure under indifferentiability notion. However many of its variants, like strengthened-MD, chop-MD, MD with HAIFA padding was proven to be secure in [8–10]. In [4],

Bertoni et. al. proved the indifferentiability of sponge mode of operation. In [1, 5], domain extension technique of Grøstl and JH was proven to be indifferentiable. For a comparative discussion on the indifferentiability of SHA3 second round candidates, we refer the reader to [2]. In this work we obtain indifferentiability security bound for two NIST second round candidates Fugue and Luffa.

Our Results In this paper we analyze indifferentiability of domain extension of Fugue and Luffa. Indifferentiability analysis of the mode of both the hash function was open [2]. Fugue can be viewed as a variant of the sponge construction with a post-processor and a fixed output length. However, due to the difference in message insertion algorithm and application of a different post-processor in Fugue, one cannot directly plug in the bounds from [4]. Our technique to prove indifferentiability of Fugue is based on the technique used in [5], where one gives an upper bound for the simulated world interpolation probability and lower bound for the real world interpolation probability. In Section 4 we prove that under the assumption that two permutations in Fugue are independent random permutations over $\{0,1\}^{nt}$, Fugue mode of operation is indifferentiable from a random oracle with advantage $\frac{\mathcal{O}(\sigma^2)}{2^{(t-1)n}}$ where σ is the total number of message blocks queried by the distinguisher. Recently Aumasson and Phan [3] have the final round transformation in Fg do not really behave like a random permutation by showing a distinguisher After wards Halevi et al [12], actually showed Fugue mode of operation behaves like random oracle assuming some weak ideal functionality of the underlying permutations. However in their analysis the attacker is restricted in the sense, that she can not make inverse queries to the permutation.

Domain extension technique of Luffa is also similar to sponge; but message injection algorithm of Luffa uses all the chaining bits (as opposed to at most half the chaining bits of Sponge or Fugue) and instead of one large permutation, Luffa uses t small permutations. Hence one cannot readily use previous techniques. First we consider Luffa mode of operation with a single large permutations and message block of n bits. We show that if the underlying permutation is assumed to be one fixed random permutation over $\{0,1\}^{tn}$, the modified Luffa mode is indifferentiable from a Random Oracle with advantage $\frac{\mathcal{O}(\sigma^2)}{2^{(t-2)n}}$ where σ is the total number of message blocks queried by the distinguisher.

Finally, in Section 6, we consider the actual Luffa mode of operation. We prove that under the assumption that underlying permutations are independent random n bit permutations, Luffa mode of operation is indifferentiable from a Random Oracle with advantage $\frac{\mathcal{O}(q^4)}{2^n}$, where q is the maximum number of queries the distinguisher makes.

Our result is the first indifferentiability analysis of Luffa. Although we achieve much less than the birthday bound, one can view this as the security-efficiency trade-off of Luffa due to its multiple small permutations to handle a large chaining value.

2 Preliminaries

2.1 Mode of Operation

Informally speaking, a mode of operation is an algorithm to construct a hash function from a compression function.

Definition 1. *A mode of operation C with oracle access to compression function $f\{0,1\}^m \to \{0,1\}^n$ is an algorithm which defines a function $C^f : \{0,1\}^* \to \{0,1\}^n$.*

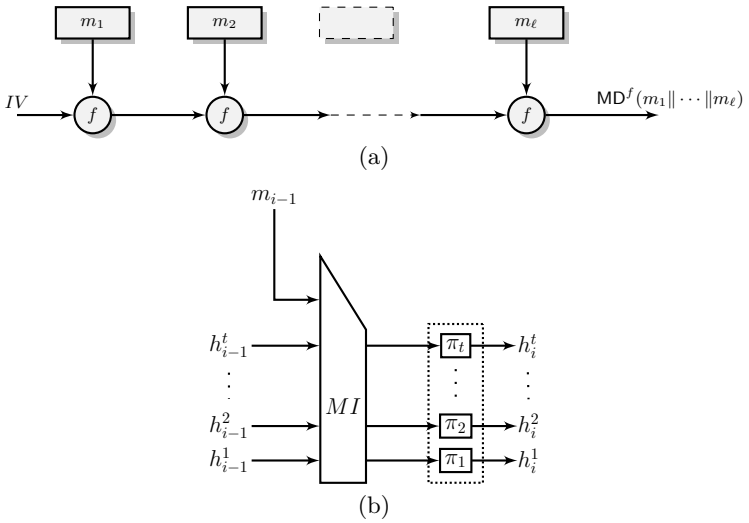

Fig. 1. (a) The Merkle-Damgård mode of operation based on compression function f and (b) The Luffa compression function

Below we describe the well known Merkle-Damgård or MD mode of operation.

Definition 2. *Let $IV \in \{0,1\}^n$ be a fixed initial value. Given a compression function $f : \{0,1\}^m \to \{0,1\}^n$, the well known Merkle-Damgård mode of operation (Fig. 1(a)) is defined as*

$$\mathsf{MD}^f_{IV}(m_1\|m_2\|\ldots\|m_l) = f(f(\ldots f(f(IV\|m_1)\|m_2)\ldots)\|m_l)$$

where $m_1, m_2, \ldots m_l \in \{0,1\}^{m-n}$.

There is a subtle difference between a hash function and a mode of operation. The mode of operation is actually a domain extension algorithm. If we supply a particular compression function f to the mode of operation algorithm we get a particular hash function. So when we think about a hash function, the compression function is fixed. Sometimes we will drop the subscript IV from MD^f_{IV} and write it as MD^f when the IV value is clear from the context (which is either IV_{Fg} or IV_{Lf}).

2.2 Luffa Mode of Operation

The compression function of Luffa, $f^{\Pi} : \{0,1\}^{(t+1)n} \rightarrow \{0,1\}^{tn}$ (Fig. 1(b)) is defined as, $f^{\Pi} \equiv P^{\Pi} \circ \mathsf{MI}_{\mathsf{Lf}}$, where $\mathsf{MI}_{\mathsf{Lf}} : \{0,1\}^{(t+1)n} \rightarrow \{0,1\}^{tn}$ is a fixed linear transformation and $P^{\Pi}(x^1 \| \cdots \| x^t) = \pi_1(x^1) \| \cdots \| \pi_t(x^t)$, $\Pi = (\pi_1, \cdots, \pi_t)$ being a tuple of t-independent random permutations on $\{0,1\}^n$.

To process arbitrary length messages the following padding rule $\mathsf{Pad}_{\mathsf{Lf}} : \{0,1\}^* \rightarrow (\{0,1\}^n)^+$ is used in Luffa. $\mathsf{Pad}_{\mathsf{Lf}}(M) = M \| 10^k$, where k is the smallest non-negative integer such that $|M| + k + 1 \equiv 0 \mod n$.

The Luffa-mode of operation is nothing but MD-mode of operation based on the compression function f^{Π} of the padded message, followed by a *finalization round*. If the required digest size n_h is same as n (the size of the permutations $\pi_1, \cdots, \pi_t$) then we pass the output of the $\mathsf{MD}^{f^{\Pi}}$, through a blank round of f^{Π} with message 0^n and process the Xor[3] of t-many permutation outputs as the final digest. If $n_h < n$, then we just Chop[4] the Xor value to give a n_h bit output. If $n < n_h \leq 2n$, then we follow the idea behind *sponge* construction, i.e. use two blank rounds f^{Π} with message 0^n each time, then concatenate the Xor of the two blank round outputs and Chop it to the desired digest size. Formally, the Luffa-mode of operation is defined as follows.

- If $n_h \leq n$,

$$\mathsf{Luffa}^{\Pi}(M) = \mathsf{Chop}_{n-n_h}(\mathsf{Xor}(\mathsf{MD}^{f^{\Pi}}(\mathsf{Pad}_{\mathsf{Lf}}(M) \| 0^n))).$$

- If $n < n_h \leq 2n$,

$$\mathsf{Luffa}^{\Pi}(M) = \mathsf{Chop}_{2n-n_h}\Big(\mathsf{Xor}\Big(\mathsf{MD}^{f^{\Pi}}(\mathsf{Pad}_{\mathsf{Lf}}(M) \| 0^n)\Big) \| \mathsf{Xor}\Big(\mathsf{MD}^{f^{\Pi}}(\mathsf{Pad}_{\mathsf{Lf}}(M) \| 0^n \| 0^n)\Big)\Big).$$

The Luffa-mode of operation uses a fixed initial value $IV_{\mathsf{Luffa}} = IV_1 \| \cdots \| IV_t$ to use with the MD-mode of operation. In Luffa specification [7], the size of the permutations are always $n = 256$ bits. For $n_h = 224$ and $n_h = 256$ bit digest size, $t = 3$ permutations are used. For $n_h = 384$ and $n_h = 512$ bit digest size, there are $t = 4$ and $t = 5$ many permutations respectively.

The Message Injection function $\mathsf{MI}_{\mathsf{Lf}}$ The message injection functions can be represented by a matrix over a ring $GF(2^8)^{n/8}$ (Note, the input length of each permutation is n bits or 8 blocks of $n/8$-bits). The definition polynomial of the field is given by $\phi(x) = x^8 + x^4 + x^3 + x + 1$. The map from an 8-block value $(a_0, \cdots, a_7)$ to an element of the ring can be defined by $(\sum_{0 \leq k < 8} a_{k,\ell} x^k)_{0 \leq \ell < n/8}$. The message injection function $\mathsf{MI}_{\mathsf{Lf}} : \{0,1\}^{(t+1)n} \rightarrow \{0,1\}^{tn}$ is defined as,

$$\mathsf{MI}_{\mathsf{Lf}}(h^1, \cdots, h^t, m) = [\mathsf{MI}_{\mathsf{Lf}}]_{t \times (t+1)} \cdot (h^1 \cdots h^t \ m)^T.$$

[3] $\mathsf{Xor} : \{0,1\}^{tn} \rightarrow \{0,1\}^n$ is defined as $\mathsf{Xor}(x^1 \| \cdots \| x^t) = x^1 \oplus \cdots \oplus x^t$.
[4] $\mathsf{Chop}_s : \{0,1\}^l \rightarrow \{0,1\}^{l-s}$ is defined as $\mathsf{Chop}_s(m \| m') = m$ where $|m'| = s$.

We will write $[\mathsf{MI}_{\mathsf{Lf}}]$ as $[TA]$, where $[T]$ is a $t \times t$ square matrix and A is a t-element column vector. In the specification of Luffa, some particular matrices $[\mathsf{MI}_{\mathsf{Lf}}]_{3\times4}, [\mathsf{MI}_{\mathsf{Lf}}]_{4\times5}$ and $[\mathsf{MI}_{\mathsf{Lf}}]_{5\times6}$ are defined. However our analysis holds whenever T is full rank (invertible) and each element of the column vector A has an inverse. For a detailed description of Luffa, the readers are referred to [7]. For simplicity, we will only consider the case $n = n_h$ in our security proofs. For other cases the same security bound can be derived in a similar but more involved manner.

LuffaS-mode of Operation We also define a simpler version of Luffa-mode of operation, LuffaS or Luffa based on a single permutation. Here the function P is modeled as a single nt-bit random permutation π. The compression function of LuffaS, $f^{\pi} : \{0,1\}^{(t+1)n} \rightarrow \{0,1\}^{tn}$ is defined as, $f^{\pi} \equiv \pi \circ MI_{\mathsf{Luffa}}$ and in case of n bit digest we have

$$\mathsf{LuffaS}^{\pi}(M) = \mathsf{Xor}(\mathsf{MD}^{f^{\pi}}(\mathsf{Pad}_{\mathsf{Lf}}(M) \| 0^n)).$$

2.3 Fugue Mode of Operation

The Fugue-mode of operation depends on $\mathit{\Pi} = (\pi_1, \pi_2)$ a pair of random permutations on $\{0,1\}^{nt}$. The compression function of Fugue, $g^{\pi_1} : \{0,1\}^{(n+1)t} \rightarrow \{0,1\}^{tn}$ is defined as, $g^{\pi_1} \equiv \pi_1 \circ \mathsf{MI}_{\mathsf{Fg}}$, where $\mathsf{MI}_{\mathsf{Fg}} : \{0,1\}^{(t+1)n} \rightarrow \{0,1\}^{tn}$ is a fixed linear transformation.

To process arbitrary length messages a suffix free padding rule $\mathsf{Pad}_{\mathsf{Fg}}$ is used. For our analysis it is sufficient assume that the padded message is multiple of n bits and it is suffix free. The Fugue-mode of operation is nothing but MD-mode of operation based on the compression function g^{π_1} followed by applying the random permutation π_2 and finally Chopping the output to the desired digest size. Formally, for n_h bit digest the Fugue-mode of operation is defined as,

$$\mathsf{Fugue}^{\mathit{\Pi}}(M) = \mathsf{Chop}_{tn-n_h}\Big(\pi_2\Big(\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M))\Big)\Big).$$

The Fugue-mode of operation uses a fixed initial value $IV_{\mathsf{Fugue}} = IV'_1 \| \cdots \| IV'_t$ to use with the MD-mode of operation. Depending on parameters there are 3 different linear transformations $\mathsf{MI}_{\mathsf{Fg}}$ (TIX in Fugue specification [11]). In our security proofs we only handle the following one,

$$\begin{aligned}\mathsf{MI}_{\mathsf{Fg}}(h^1, \cdots, h^t, m) = (m, h^2 \oplus h^{t-5}, h^3, h^4, \cdots, \\ h^8, h^9 \oplus m, h^{10}, h^{11} \oplus h^1, h^{12}, h^{13}, \cdots, h^t). \quad (1)\end{aligned}$$

For other cases, a similar security analysis holds.

2.4 Indifferentiability

The notion of indifferentiability, introduced by Maurer et. al. in [13], is a generalization of classical notion of indistinguishability. Loosely speaking, if an ideal primitive $\mathcal{G}$ is indifferentiable with a construction C based on another ideal primitive $\mathcal{F}$, then $\mathcal{G}$ can be safely replaced by $C^{\mathcal{F}}$ in any cryptographic construction. In other terms if a cryptographic construction is secure in $\mathcal{G}$ model then it is secure in $\mathcal{F}$ model.

Definition 3. Advantage:
Let F_i, G_i be probabilistic oracle algorithms. We define advantage *of the adversary $\mathcal{A}$ at distinguishing (F_1, F_2) from (G_1, G_2) as*

$$\mathsf{Adv}_{\mathcal{A}}((F_1, F_2), (G_1, G_2)) = |\Pr[\mathcal{A}^{F_1,F_2} = 1] - \Pr[\mathcal{A}^{G_1,G_2} = 1]|.$$

Definition 4. Indifferentiability [13]:
A Turing machine C with oracle access to an ideal primitive $\mathcal{F}$ is said to be $(t, q_C, q_{\mathcal{F}}, \varepsilon)$ indifferentiable from an ideal primitive $\mathcal{G}$ if there exists a simulator S with an oracle access to $\mathcal{G}$ and running time at most t, such that for any adversary $\mathcal{A}$, it holds that

$$\mathsf{Adv}_D((C^{\mathcal{F}}, \mathcal{F}), (\mathcal{G}, S^{\mathcal{G}})) < \varepsilon.$$

The adversary makes at most q_C queries to C or $\mathcal{G}$ and at most $q_{\mathcal{F}}$ queries to $\mathcal{F}$ or S. Similarly, $C^{\mathcal{F}}$ is said to be (computationally) indifferentiable from $\mathcal{G}$ if running time of $\mathcal{A}$ is bounded above by some polynomial in the security parameter k and ε is a negligible function of k.

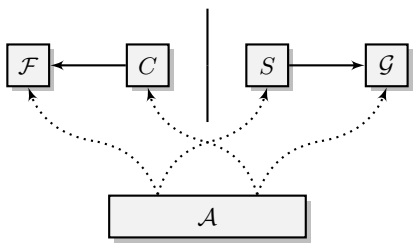

Fig. 2. The indifferentiability notion

We stress that in the above definition $\mathcal{G}$ and $\mathcal{F}$ can be two completely different primitives. As shown in Fig 2 the role of the simulator is to not only simulate the behavior of $\mathcal{F}$ but also remain consistent with the behavior of $\mathcal{G}$. Note that, the simulator does not know the queries made directly to $\mathcal{G}$, although it can query $\mathcal{G}$ whenever it needs. In this paper $\mathcal{G}$ is a variable input length Random oracle and $\mathcal{F}$ is a random permutation. As the objective of any adversary is to build a distinguisher , we use the term adversary and distinguisher interchangeably for the rest of the paper.

3 Main Tools for Bounding Distinguisher's Advantage

We follow a similar approach to [5, 8, 9] for proving indifferentiability of Fugue and Luffa. In the discussion below C-mode of operation usually refers to either Fugue or Luffa mode of operation based on $\Pi = (\pi_1, \cdots, \pi_t)$ a tuple of t random permutations. In other words, $C \in \{\mathsf{Fugue}, \mathsf{Luffa}\}$. In case of Fugue, we have $t = 2$. Below we recall some notations from [5].

Definition 5. Consistent Oracle:
A (small domain) probabilistic oracle algorithm G_2 is said to be Consistent to a (big domain) probabilistic oracle algorithm G_1 with respect to MO-mode of operation if for any point x (from the big domain), we have

$$\Pr[G_1(x) = MO^{G_2}(x)] = 1.$$

Note, Π is always Consistent to C^{Π}-mode of operation.

There might be some point x for which the value of $MO^{G_2}(x)$ gets fixed by the relations $G_2(x_1) = y_1, \cdots, G_2(x_q) = y_q$. Such x's are called evaluatable by the relations $G_2(x_1) = y_1, \cdots, G_2(x_q) = y_q$. Formally,

Definition 6. Evaluatable Queries:
A point $x \in Domain(MO^{G_2})$ is called evaluatable *with respect to MO-mode of operation (based on G_2) by the relations $G_2(x_1) = y_1, \cdots, G_2(x_q) = y_q$, if there exist a deterministic algorithm $\mathcal{B}$ such that,*

$$\Pr[MO^{G_2}(x) = \mathcal{B}(x, (x_1, y_1), \cdots, (x_q, y_q)) | G_2(x_1) = y_1, \cdots, G_2(x_q) = y_q] = 1.$$

In this paper the adversary is modeled as a deterministic, computationally unbounded [5] adversary $\mathcal{A}$ which has access to two oracles $\mathcal{O}_1$ and $\mathcal{O}_2$. Recall that $\mathcal{A}$ tries to distinguish the output distribution of (C^{Π}, Π) from that of (R, S^R). We say $\mathcal{A}$ queries $\mathcal{O}_1$ when it queries the oracle C^{π} or R and queries $\mathcal{O}_2$ when it queries the oracle Π or S^R. As we model Π as a tuple of t random permutations, the adversary has to mention the index of the random permutation it wants to query and whether she wants to make forward or inverse query. The forward and inverse query to the i^{th} random permutation are denoted by $\mathcal{O}_2(+i, \cdot)$ and $\mathcal{O}_2(-i, \cdot)$ respectively.

Definition 7. Distinguisher View:
The view $\mathcal{V}$ of the distinguisher is the query-response tuple

$$((M_1, h_1), \cdots, (M_{q_0}, h_{q_0}), (x_1^1, y_1^1), \cdots, (x_{q^1+q^{-1}}^1, y_{q^1+q^{-1}}^1), \cdots, (x_1^t, y_1^t), \cdots, (x_{q^t+q^{-t}}^t, y_{q^t+q^{-t}}^t)), \quad (2)$$

where

- $\mathcal{O}_1(M_1) = h_1, \cdots, \mathcal{O}_1(M_{q_0}) = h_{q_0}$

[5] Any deterministic adversary with unlimited resources is as powerful as a randomized adversary [14].

– $\mathcal{O}_2(+1, x_1^1) = y_1^1, \cdots, \mathcal{O}_2(+1, x_{q^1}^1) = y_{q^1}^1$
– $\mathcal{O}_2(-1, y_{q^1+1}^1) = x_{q^1+1}^1, \cdots, \mathcal{O}_2(-1, y_{q^1+q^{-1}}^1) = x_{q^1+q^{-1}}^1$
$\vdots$
– $\mathcal{O}_2(+t, x_1^t) = y_1^t, \cdots, \mathcal{O}_2(+t, x_{q^t}^t) = y_{q^t}^t$
– $\mathcal{O}_2(-t, y_{q^t+1}^t) = x_{q^t+1}^t, \cdots, \mathcal{O}_2(-t, y_{q^t+q^{-t}}^t) = x_{q^t+q^{-t}}^t$.

Definition 8. Input Output View:
For any view $\mathcal{V}$ as in (2), we define Input View $\mathcal{IV}$ and Output View $\mathcal{OV}$ as follows,

$$\mathcal{IV} = (M_1, \cdots, M_{q_0}, x_1^1, \cdots, x_{q^1}^1, y_{q^1+1}^1, \cdots, y_{q^1+q^{-1}}^1, \cdots, x_1^t, \cdots, x_{q^t}^t, y_{q^t+1}^t, \cdots, y_{q^t+q^{-t}}^t),$$
$$\mathcal{OV} = (h_1, \cdots, h_{q_0}, y_1^1, \cdots, y_{q^1}^1, x_{q^1+1}^1, \cdots, x_{q^1+q^{-1}}^1, \cdots, y_1^t, \cdots, y_{q^t}^t, x_{q^t+1}^t, \cdots, x_{q^t+q^{-t}}^t).$$

We recall the observations made in [5].

1. $\mathcal{V}$, $\mathcal{IV}$ and $\mathcal{OV}$ are actually ordered tuples. That means, the position of any element inside the tuple actually denotes the corresponding query number. So, in general $\mathcal{O}_1(\cdot)$, $\mathcal{O}_2(+i, \cdot)$ and $\mathcal{O}_2(-i, \cdot)$ queries should not be grouped together. But we write it like this to avoid further notational complexity.
2. For any deterministic *non-adaptive* attacker $\mathcal{IV}$ is always fixed.
3. For any deterministic *adaptive* attacker $\mathcal{IV}$ is actually determined by $\mathcal{OV}$ [14].
4. For any deterministic attacker (adaptive or non-adaptive) $\mathcal{V}$ is actually determined by $\mathcal{OV}$.

$\mathcal{OV}_{\mathcal{O}_1,\mathcal{O}_2}^{\mathcal{A}}$ be the random variable corresponding to the output view of attacker $\mathcal{A}$, obtained after interacting with $\mathcal{O}_1$, $\mathcal{O}_2$.

Definition 9. Consistent Output View:
For an attacker $\mathcal{A}$ an output view $\mathcal{OV}$ is said to be Consistent *with respect to $(\mathcal{O}_1, \mathcal{O}_2)$ if $\Pr[\mathcal{OV}_{\mathcal{O}_1,\mathcal{O}_2}^{\mathcal{A}} = \mathcal{OV}] > 0$. If an output view is not Consistent, then it is an* Inconsistent *output view.*

Definition 10. Irreducible (Output) View:

$$\mathcal{V} = ((M_1, h_1), \cdots, (M_{q_0}, h_{q_0}), (x_1^1, y_1^1), \cdots, (x_{q^1+q^{-1}}^1, y_{q^1+q^{-1}}^1), \cdots, (x_1^t, y_1^t), \cdots, (x_{q^t+q^{-t}}^t, y_{q^t+q^{-t}}^t)) \quad (3)$$

*is called C-*Irreducible *view if,*

1. *$M_1, \cdots, M_{q_0}$ are distinct,*
2. *All the elements of $\mathcal{X} = \{x_1^i, \cdots, x_{q^1+q^{-1}}^i\}$ are distinct and all the elements of $\mathcal{Y} = \{y_1^1, \cdots, y_{q^1+q^{-1}}^1\}$ are distinct.*

3. $M_1, \cdots, M_{q_0}$ *are not evaluatable by the relations,*

$$\pi_1(x_1^1) = y_1^1, \cdots, \pi_1(x_{q^1+q^{-1}}^1) = y_{q^1+q^{-1}}^1, \cdots, \\ \pi_1(x_1^t) = y_1^t, \cdots, \pi_1(x_{q^t+q^{-t}}^t) = y_{q^t+q^{-t}}^t \quad (4)$$

with respect to C-mode of operation.

For an attacker $\mathcal{A}$, an output view $\mathcal{OV}$ is called Irreducible if the corresponding view $\mathcal{V}$ is Irreducible.

Theorem 1. *[5] If there exists a simulator S^R (aborting with a probability at most ε') Consistent to a random oracle R, with respect to C-mode of operation, such that for any attacker $\mathcal{A}$ making at most q queries , the relation*

$$\Pr[\mathcal{OV}^{\mathcal{A}}_{C^{\Pi},\Pi} = \mathcal{OV}] \geq (1-\varepsilon)\Pr[\mathcal{OV}^{\mathcal{A}}_{R,S^R} = \mathcal{OV}],$$

holds for all possible Consistent (with respect to the oracles C^{Π}, Π) and Irreducible output views $\mathcal{OV}$; then for any attacker $\mathcal{A}$ making at most q queries we have

$$\mathsf{Adv}_{\mathcal{A}}((C^{\Pi},\Pi),(R,S^R)) \leq \varepsilon + \varepsilon'.$$

4 Indifferentiability Security Analysis of Fugue

In this section we show the Fugue-mode of operation is indifferentiable from a random function R. If one could show the compression function of Fugue, g^{π_1} is indifferentiable from a random oracle, then he can try to apply the results of [10] to show Fugue is indifferntiable from a random oracle. However, π_1 being a random permutation attacker has access to inverse queries and g^{π_1} is not really indifferentiable from a random oracle. Hence, to prove indifferentiability security of Fugue we adopt the direct approach as outlined in Section 3. Our main result of this section is the following.

Theorem 2. *Let $\Pi = (\pi_1, \pi_2)$ be a pair of independent random permutations over $\{0,1\}^{nt}$. Let Fugue^{Π} be the Fugue mode of operation with n_h bit digest. There exists a simulator $S^{\mathcal{R}}$ such that for any adversary $\mathcal{A}$ which makes at most q queries*

$$\mathsf{Adv}_{\mathcal{A}}((\mathsf{Fugue}^{\Pi},\Pi),(\mathcal{R},S^{\mathcal{R}})) \leq \frac{\mathcal{O}(q^2+q\sigma+\sigma^2)}{2^{n(t-1)}} + \frac{\mathcal{O}(q^2)}{2^{nt-n_h}}$$

where σ is the total number of blocks in the queries to R or Fugue^{Π}.

To start with, we build a simulator S^R Consistent to the random oracle R, with respect to Fugue-mode of operation.

4.1 Simulator of Fugue

The simulator maintains two partial permutations $\pi_1^*, \pi_2^* : \{0,1\}^{nt} \to \{0,1\}^{nt}$ initially empty and a (weighted) directed graph (in fact a tree) $G = (V, E)$. Each element of the vertex set V is an nt-bit element. V is initialized to IV_{Fugue} and the edge set E is initially empty. Whenever the simulator answers $S^R(+i, x)$ query as y or $S^R(-i, y)$ query as x it updates the partial permutation π_i^*, as $\pi_i^*(x) = y$. There exists an edge v_1 to v_2 with weight m in the edge set (or $v_1 \xrightarrow{m} v_2 \in E$) if and only if

$$g^{\pi_1^*}(v_1 \| m) = v_2.$$

While, answering a query the simulator also need to keep track of already defined input/output points. At any instance, $\mathcal{X}(\pi_i^*)$ and $\mathcal{Y}(\pi_i^*)$ be the set of already defined input and output points of the partial permutation π_i^* respectively. The simulator always answers such a way such that π_i^* behaves as a random (partial) permutation and G is a tree with root node as IV_{Fugue}. In fact, if there is a path in G from IV_{Fugue} to v with weights (messages) $m_1, \cdots, m_k$ that would imply

$$\mathsf{MD}^{g^{\pi_1^*}}(m_1 \| \cdots \| m_k) = v.$$

We recall that Fugue-mode of operation is nothing but processing the output of $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(\cdot))$ through π_2 and chopping $nt - n_h$ many bits of π_2 output to generate n_h bit digest size. Hence to remain Consistent with R, if $m_1 \| \cdots \| m_k$ is an appropriate padded message (corresponding to message M) the simulator should make sure

$$\mathsf{Chop}_{nt-n_h}(\pi_2^*(v)) = R(M).$$

The simulator needs to be careful about two things. Firstly, the tree structure of the graph is always maintained and secondly, whenever any new node v gets added to the tree the partial permutation π_2^* should not be already defined at point v. When the simulator fails to do so, it outputs $\perp$ to abort. The simulator only allows the creation of a new node during $S^R(+1, x)$ queries. Note, in this case whether a new node would be created only depends on x the query input. This can be checked easily, for each $v \in V$ find out whether there exists a message m such that

$$\mathsf{MI}_{\mathsf{Fg}}(v \| m) = x.$$

Also, a single $S^R(+1, x)$ query should not be able to create more than one new node.

$S^R(+1, x)$ query : When $x \notin \mathcal{X}(\pi_1^*)$ the simulator samples y from $\{0,1\}^{tn} \setminus \mathcal{Y}(\pi_1^*)$ uniformly. The simulator aborts, when either of the following conditions get violated. Otherwise it updates the partial permutation π_1^* and returns y, if y is a new node it also updates the graph G accordingly.

- $\mathbf{Bad}_{\mathsf{Fg}}^{+11}$: y is a new node and there exist $m_1, m_2 \in \{0,1\}^n$, $v \in V$ such that $\mathsf{MI}_{\mathsf{Fg}}(y \| m_1) = \mathsf{MI}_{\mathsf{Fg}}(v \| m_2)$.

– $\mathbf{Bad}^{+12}_{\mathsf{Fg}}$: y is a new node and there exist $m \in \{0,1\}^n$, $z \in \mathcal{X}(\pi_1^*)$ such that $\mathsf{MI}_{\mathsf{Fg}}(y\|m) = z$.
– $\mathbf{Bad}^{+13}_{\mathsf{Fg}}$: y is a new node, the path from IV_{Fugue} leading to y is a valid padded message and $y \in \mathcal{X}(\pi_2^*)$.

$S^R(-1, y)$ query : When $y \notin \mathcal{Y}(\pi_1^*)$ the simulator samples x from $\{0,1\}^{tn} \setminus \mathcal{X}(\pi_1^*)$ uniformly. The simulator aborts when the following condition gets violated. Otherwise it updates the partial permutation π_1^* and returns x.

– $\mathbf{Bad}^{-11}_{\mathsf{Fg}}$: There exist $m \in \{0,1\}^n$ and $v \in V$ such that $\mathsf{MI}_{\mathsf{Fg}}(v\|m) = x$.

$S^R(+2, x)$ query : When $x \notin \mathcal{X}(\pi_2^*)$, at first the simulator checks whether $x \in V$ or not. If $x \notin V$ or the path from IV_{Fugue} leading to x is not a valid padded message the simulator returns y sampled uniformly from $\{0,1\}^{tn} \setminus \mathcal{Y}(\pi_2^*)$ and updates π_2^* accordingly. In case $\mathsf{Pad}_{\mathsf{Fg}}(M)$, a valid padded message, is the path from IV_{Fugue} to x then the simulator aborts if the following condition holds. Otherwise, it returns $R(M)$ and updates π_2^* accordingly.

– $\mathbf{Bad}^{+21}_{\mathsf{Fugue}}$: $R(M) \in \mathcal{Y}(\pi_2^*)$.

$S^R(-2, y)$ query : When $y \notin \mathcal{Y}(\pi_2^*)$, the simulator samples $x \in \{0,1\}^{tn} \setminus \mathcal{X}(\pi_2^*)$ uniformly. It aborts if the following condition gets violated, otherwise it returns x and updates the partial permutation π_2^*.

– $\mathbf{Bad}^{-21}_{\mathsf{Fg}}$: There exists $v \in V$ such that $x = v$.

4.2 Upper bound on abort probability

Let q_1 be the upper bound on number of S^R queries. We have also seen the number of nodes in the graph G can increase by at most 1 for every S^R query. Hence, before answering any S^R query the graph G can contain only q_1 many nodes. Below we make a few observation about the function $\mathsf{MI}_{\mathsf{Fg}}$ as in (1).

1. $\mathsf{MI}_{\mathsf{Fg}}(v_1\|m_1) = \mathsf{MI}_{\mathsf{Fg}}(v_2\|m_2)$ implies $m_1 = m_2$.
2. For any $v \in \{0,1\}^{nt}$, there exist at most 2^n many $v' \in \{0,1\}^{nt}$ such that the relation $\mathsf{MI}_{\mathsf{Fg}}(v\|m) = \mathsf{MI}_{\mathsf{Fg}}(v'\|m)$, holds for some $m \in \{0,1\}^n$.
3. Observations 1 and 2 actually imply for any $v \in \{0,1\}^{nt}$ there exist at most 2^n many $v' \in \{0,1\}^{nt}$ such that the relation $\mathsf{MI}_{\mathsf{Fg}}(v\|m) = \mathsf{MI}_{\mathsf{Fg}}(v'\|m')$, holds for some $m, m' \in \{0,1\}^n$.
4. For any $z \in \{0,1\}^{nt}$ there exist at most 2^n many $v \in \{0,1\}^{nt}$ such that the relation $\mathsf{MI}_{\mathsf{Fg}}(v\|m) = z$, holds for some $m \in \{0,1\}^n$.

Utilizing the above observations one can deduce the following upper bounds on $\Pr[\mathbf{Bad}^{\pm ij}_{\mathsf{Fg}}]$, during a single S^R query.

$\Pr[\mathbf{Bad}^{+11}_{\mathsf{Fg}}] \leq \frac{2^n q}{2^{nt}-q}$ $\quad$ $\Pr[\mathbf{Bad}^{+12}_{\mathsf{Fg}}] \leq \frac{2^n q_1}{2^{nt}-q}$ $\quad$ $\Pr[\mathbf{Bad}^{+13}_{\mathsf{Fg}}] \leq \frac{q}{2^{nt}-q}$

$\Pr[\mathbf{Bad}^{-11}_{\mathsf{Fg}}] \leq \frac{2^n q_1}{2^{nt}-q}$ $\quad$ $\Pr[\mathbf{Bad}^{+21}_{\mathsf{Fg}}] \leq \frac{q}{2^{nt}}$ $\quad$ $\Pr[\mathbf{Bad}^{-21}_{\mathsf{Fg}}] \leq \frac{q}{2^{nt}-q}$

Note, the upper bound on $\Pr[\mathbf{Bad}^{+21}_{\mathsf{Fg}}]$ is derived assuming there is no chopping. When there is no chopping the simulator aborts when $R(M) \in \mathcal{Y}(\pi_2^*)$, otherwise it returns $R(M)$. Hence, the probability of abort is exactly $\frac{|\mathcal{Y}(\pi_2^*)|}{2^{nt}}$. In case $n - n_h$ bits are chopped, the abort probability is even less. Hence, if Abort is the event that the simulator aborts in any one of the q queries made by the attacker, assuming $q \leq 2^{nt-1}$ we have

$$\Pr[\mathsf{Abort}] \leq \frac{6q^2}{2^{n(t-1)}}.$$

4.3 Interpolation Probability of $\mathcal{OV}^{\mathcal{A}}_{(R,S^R)}$

There is always an one to one mapping between any (output) view and the simulators internal state (that is the graph G and the partial permutations π_1^*, π_2^*). Hence any Irreducible output view $\mathcal{OV}$, either implies that the simulator should abort, in which case $\Pr[\mathcal{OV}^{\mathcal{A}}_{(R,S^R)} = \mathcal{OV}] = 0$, or the simulator do not abort, in which case we have

$$\Pr[\mathcal{OV}^{\mathcal{A}}_{(R,S^R)} = \mathcal{OV}] = \frac{1}{2^{n_h q_0}} \prod_{i=1}^{q^1+q^{-1}} \prod_{j=1}^{q^2+q^{-2}} \frac{1}{(2^{nt}-i+1)(2^{nt}-j+1)}.$$

Hence, for any Consistent output view $\mathcal{OV}$ (with respect to Fugue^{Π}, Π oracles) we have

$$\Pr[\mathcal{OV}^{\mathcal{A}}_{(R,S^R)} = \mathcal{OV}] \leq \frac{1}{2^{n_h q_0}} \prod_{i=1}^{q^1+q^{-1}} \prod_{j=1}^{q^2+q^{-2}} \frac{1}{(2^{nt}-i+1)(2^{nt}-j+1)}.$$

Also for any, Inconsistent output view $\mathcal{OV}$ (with respect to Fugue^{Π}, Π oracles) we have $\Pr[\mathcal{OV}^{\mathcal{A}}_{(R,S^R)} = \mathcal{OV}] = 0$.

4.4 Interpolation Probability of $\mathcal{OV}^{\mathcal{A}}_{(\mathsf{Fugue}^{\Pi},\Pi)}$

Over here we aim to give a lower bound on $\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{Fugue}^{\Pi},\Pi)} = \mathcal{OV}]$ for any Consistent Irreducible output view $\mathcal{OV}$. For that we will consider the notion of *Good* $\mathsf{NCFugue}$-Irreducible views, which is similar to the notion of MD-Irreducible views in [5, 9]. Informally $\mathsf{NCFugue}$-Irreducible view is an Irreducible attacker view where the attacker has access to $\mathsf{NCFugue}^{\Pi}(\cdot) \equiv \pi_2(\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(\cdot)))$ oracle instead of $\mathsf{Fugue}^{\Pi}(\cdot) \equiv \mathsf{Chop}_{nt-n_h}(\pi_2(\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(\cdot))))$ oracle (So the attacker essentially receives more information. However, the attacker does not use this extra information to decide its future queries). We will obtain the lower bound in two steps.

- For any Consistent *Good* $\mathsf{NCFugue}$-Irreducible output view $\mathcal{OV}_{\mathsf{NC}}$ obtain a lower bound of
  $\Pr[\mathcal{OV}^{\mathcal{A}'}_{(\mathsf{NCFugue}^{\Pi},\Pi)} = \mathcal{OV}_{\mathsf{NC}}]$ as p (Here, $\mathcal{A}'$ is the modified attacker which has access to $\mathsf{NCFugue}^{\Pi}$ oracle instead of Fugue^{Π} oracle.)

– For any Consistent Fugue-Irreducible output view $\mathcal{OV}$ we give a lower bound (as N) on number of possible Good NCFugue-Irreducible output view $\mathcal{OV}_{\mathsf{NC}}$ such that,

$$\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{NCFugue}^{\Pi},\Pi)} = \mathcal{OV} | \mathcal{OV}^{\mathcal{A}'}_{(\mathsf{MDFugue}^{\Pi},\Pi)} = \mathcal{OV}_{\mathsf{NC}}] = 1 \tag{5}$$

This would imply, $\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{Fugue}^{\Pi},\Pi)} = \mathcal{OV}] \geq pN$. Before going further we define the notion Good NCFugue-Irreducible output view.

Definition 11. *For an attacker $\mathcal{A}'$, interacting with $(\mathsf{NCFugue}^{\Pi}, \Pi)$ the view*

$$\mathcal{V}_{\mathsf{NC}} = ((M_1, r_1), \cdots, (M_{q_0}, r_{q_0}), (x_1^1, y_1^1), \cdots, (x^1_{q^1+q^{-1}}, y^1_{q^1+q^{-1}}), \cdots, (x_1^2, y_1^2), \cdots, (x^2_{q^2+q^{-2}}, y^2_{q^2+q^{-2}})) \tag{6}$$

is a Good NCFugue-*Irreducible view if $\mathcal{V}_{\mathsf{NC}}$ is an* NCFugue-*Irreducible view, $r_1, \cdots, r_{q_0}$ are all different and they are different from $y_1^2, \cdots, y^2_{q^2+q^{-2}}$. An output view $\mathcal{OV}_{\mathsf{NC}}$ is called Good* NCFugue-*Irreducible output view, if the corresponding view $\mathcal{V}_{\mathsf{NC}}$ is Good* NCFugue-*Irreducible.*

In the following theorem proof of which is given in Appendix A, we get the lower bound on $\Pr[\mathcal{OV}^{\mathcal{A}'}_{(\mathsf{NCFugue}^{\Pi},\Pi)} = \mathcal{OV}_{\mathsf{NC}}]$ or estimate of p.

Theorem 3. *For any Consistent Good* NCFugue-*Irreducible output view $\mathcal{OV}_{\mathsf{NC}}$ ($\mathcal{V}_{\mathsf{NC}}$ as in* (6) *being the corresponding view),*

$$\begin{aligned}&\Pr[\mathcal{OV}^{\mathcal{A}'}_{(\mathsf{NCFugue}^{\Pi},\Pi)} = \mathcal{OV}_{\mathsf{NC}}] \\ &\geq (1 - \frac{2\sigma(q+\sigma)}{2^{(t-1)n}})\frac{1}{2^{ntq_0}} \prod_{i=1}^{q^1+q^{-1}} \prod_{j=1}^{q^2+q^{-2}} \frac{1}{(2^{nt}-i+1)(2^{nt}-j+1)},\end{aligned}$$

where σ is the total number of message blocks present in NCFugue *queries, q is the total number of* NCFugue *and Π queries such that $q + \sigma < 2^{nt-1}$.*

In the following theorem (proof sketch Appendix B) we get an estimate of N.

Theorem 4. *For any Consistent* Fugue-*Irreducible view $\mathcal{V}$ as in* (3) *(with $t = 2$), there exist at least*

$$2^{(nt-n_h)q_0}\Big(1 - \frac{q^2}{2^{nt-n_h}}\Big)$$

many Consistent Good NCFugue-*Irreducible view $\mathcal{V}_{\mathsf{NC}}$ as in* (6) *such that, $\mathsf{Chop}_{nt-n_h}(r_1) = h_1, \cdots, \mathsf{Chop}_{nt-n_h}(r_{q_0}) = h_{q_0}$, where q is the total number of* Fugue$^{\Pi}$ *and Π queries.*

All together Theorem 3, Theorem 4 and the upper bound from Section 4.3 would imply for any Consistent output view $\mathcal{OV}$ (with respect to $\mathsf{Fugue}^{\Pi}, \Pi$ oracles)

$$\begin{aligned}\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{Fugue}^{\Pi},\Pi)} = \mathcal{OV}] &\geq \Big(1 - \frac{2\sigma(q+\sigma)}{2^{(t-1)n}}\Big)\Big(1 - \frac{q^2}{2^{nt-n_h}}\Big) \times \frac{1}{2^{n_h q_0}} \\ &\quad \times \prod_{i=1}^{q^1+q^{-1}} \prod_{j=1}^{q^2+q^{-2}} \frac{1}{(2^{nt}-i+1)(2^{nt}-j+1)} \\ &\geq \Big(1 - \frac{2\sigma(q+\sigma)}{2^{(t-1)n}}\Big)\Big(1 - \frac{q^2}{2^{nt-n_h}}\Big) \Pr[\mathcal{OV}^{\mathcal{A}}_{(R,S^R)} = \mathcal{OV}]\end{aligned}$$

Applying the above inequality together with $\Pr[\mathsf{Abort}]$ to Theorem 1 we prove Theorem 2.

5 Indifferentiability Security Analysis of LuffaS

We recall that Luffa with single permutation or LuffaS is based on a random permutation $\pi : \{0,1\}^{tn} \to \{0,1\}^{tn}$. We view the the Message Insertion transformation $\mathsf{MI}_{\mathsf{Lf}}(Y\|m)$ as $TY + Am$ where $Y \in (\{0,1\}^n)^t, m \in \{0,1\}^n$. We view the chaining value Y as a vector of t many n bit elements. By "forward query input" we mean the input of forward query or the output of inverse query made to the simulator. Our main result of this section is the following

Theorem 5. *Let π be an independent random permutations over $\{0,1\}^{nt}$. Let LuffaS^{Π} be the* LuffaS *mode of operation with n bit message blocks. There exists a simulator $S^{\mathcal{R}}$ such that for any adversary $\mathcal{A}$ which makes at most q queries*

$$\mathsf{Adv}_{\mathcal{A}}((\mathsf{LuffaS}^{\pi}, \pi), (\mathcal{R}, S^{\mathcal{R}})) \leq \frac{\mathcal{O}(\sigma(q+\sigma))}{2^{nt}} + \frac{\mathcal{O}(q^2)}{2^{n(t-2)}}$$

where σ is the total number of blocks in the queries to R or LuffaS^{π}.

As in Section 4 we start by describing a Consistent Simulator.

5.1 Simulator of LuffaS

The simulator maintains a partial permutation $\pi^* : \{0,1\}^{nt} \to \{0,1\}^{nt}$ to keep track of the all the received queries and responses it made so far. By $\mathcal{X}(\pi^*)$ and $\mathcal{Y}(\pi^*)$ we denote the set of forward query input and the set of forward query output so far. It also maintains a directed graph G whose vertex set $V = \{0,1\}^{tn}$. These vertices are actually the output of the simulator. An edge from vertex Y to vertex Y', marked by m, implies that $(TY + Am, Y') \in \pi^*$. Initially π^* is empty and $V = \{IV_{\mathsf{Luffa}}\}$. The objective of the simulator is to maintain following properties in the graph

- **No cycle**: the graph G is actually a tree, hence there is no cycle in the graph.

– **No collision**: For any three nodes Y_1, Y_2 and Y_3 in the graph, both (Y_1, Y_3) and (Y_2, Y_3) are not edges in G.

In addition to the properties of the tree, to guard against the *length-extension attack*, the simulator also tries to ensure that the final input of any path from IV_{Luffa} is not in $\mathcal{X}(\pi^*)$ yet.

Handling forward query $S^{\mathcal{R}}(+, X)$ If $X \notin \mathcal{X}(\pi^*)$,simulator checks whether for any existing node Y' and a message m, $\mathsf{MI}_{\mathsf{Lf}}(Y', m) = X$. If such a node exist for $m = 0^n$, then the simulator, retrieves M from labels of the edges of path from IV_{Luffa} to Y' and queries $\mathcal{R}(M)$. Then queries the simulator samples $Y = (y_1, \cdots, y_t)$ uniformly at random from $\{0,1\}^{nt} \setminus \mathcal{Y}(\pi^*)$ conditioned on $y_1 + \cdots + y_t = R(M)$. If $m \neq O^n$, the simulator samples $Y \leftarrow \{0,1\}^{tn} \setminus \mathcal{Y}(\pi^*)$ uniformly at random. In both the cases, a node Y is created and an edge (Y', Y) with label m is added. If no such Y' is found, simulator just the simulator samples $Y \leftarrow \{0,1\}^{tn} \setminus \mathcal{Y}(\pi^*)$ uniformly at random. If Y sets one of the following **Bad** events true, the simulator aborts. Otherwise, the simulator updates the partial permutation π^* inserting (X, Y) and returns Y as response.

We define the following events as **Bad** events for the simulator.

– $\mathbf{Bad}^{+1}_{\mathsf{Lf}}$ (Possible Collision): Let Y be the new node and Y_2 be any other node of the graph. $T(Y + Y_2) = Am$ has a solution for m.
– $\mathbf{Bad}^{+2}_{\mathsf{Lf}}$: Let Y be the new node. If, for some m, $TY + Am \in \mathcal{X}(\pi^*)$.

$\mathbf{Bad}^{+2}_{\mathsf{Lf}}$ ensures that there is no cycle in the graph and the final input (input to the finalization) for any properly padded message M is not yet in $\mathcal{X}(\pi^*)$.

Handling inverse query $S^{\mathcal{R}}(-, Y)$ While handling inverse queries, the simulator has no control over the input. The adversary might choose the inverse query Y in such a way that for some node Y' in the graph and some m, m', $TY + Am = TY' + Am'$. However, as long as the output of the inverse query do not make a new chain from IV, and the simulator remains Consistent with $\mathcal{R}$ with respect to the mode. Hence, while answering an inverse query the simulator needs to make sure that for no existing node Y' and any message m, $TY' + Am$ matches with the output. This condition together with $\mathbf{Bad}^{+1}_{\mathsf{Lf}}$ and $\mathbf{Bad}^{+2}_{\mathsf{Lf}}$ keeps the simulator Consistent with $\mathcal{R}$. So, if $Y \notin \mathcal{Y}(\pi^*)$, the simulator samples $X \leftarrow \{0,1\}^{tn} \setminus \mathcal{X}(\pi^*)$ and check for the following **Bad** event.

– $\mathbf{Bad}^{-1}_{\mathsf{Lf}}$: For some vertex Y' in the graph and some message $m \in \{0,1\}^n$, $\mathsf{MI}_{\mathsf{Lf}}(Y', m) = X$.

If the **Bad** is true then the simulator aborts. Otherwise, it adds (X, Y) to π^* and returns X.
In Appendix C we give an upper bound on the abort probability of the simulator and we have the following lemma.

Lemma 1. *Let $S^{\mathcal{R}}$ be the simulator for* Luffa *mode of operation with single permutation. For any attacker $\mathcal{A}$ making at most $q \leq 2^n$ queries*

$$\Pr[\mathsf{Abort}] \text{ or } \Pr[S^{\mathcal{R}} \rightarrow \perp] \leq \frac{\mathcal{O}(q^2)}{2^{(t-2)n}}.$$

Finally, similar to Section 4.3 for any Consistent output view $\mathcal{OV}$ (with respect to $\mathsf{LuffaS}^{\pi}, \pi$ oracles) we have

$$\Pr[\mathcal{OV}^{\mathcal{A}}_{(R,S^R)} = \mathcal{OV}] \leq \frac{1}{2^{nq_0}} \prod_{i=1}^{q^1+q^{-1}} \frac{1}{(2^{nt}-i+1)}.$$

5.2 Interpolation Probability of $\mathcal{OV}^{\mathcal{A}}_{(\mathsf{LuffaS}^{\pi},\pi)}$

In this section we aim obtain a lower bound on $\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{LuffaS}^{\pi},\pi)} = \mathcal{OV}]$ for any Irreducible Consistent output view $\mathcal{OV}$. Similar to Section 4.4 we consider the notion of *Good* NXLuffaS-Irreducible views. Informally, an NXLuffaS-Irreducible view is an Irreducible view where the attacker has access to $\mathsf{NXLuffaS}^{\pi}(\cdot) \equiv \mathsf{MD}^{f^{\pi}}(\mathsf{Pad}_{\mathsf{Lf}}(\cdot)\|0^n)$ oracle instead of $\mathsf{LuffaS}^{\pi}(\cdot) = \mathsf{Xor}(\mathsf{MD}^{f^{\pi}}(\cdot)\|0^n)$ oracle. As before, $\mathcal{A}'$ is the modified attacker which has access to $\mathsf{NXLuffaS}^{\pi}$ oracle instead of LuffaS^{π} oracle. Our strategy is same as before, i.e. for any Consistent Good NXLuffaS-Irreducible output view $\mathcal{OV}_{\mathsf{NX}}$ obtain a lower bound on $\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{NXLuffaS}^{\pi},\pi)} = \mathcal{OV}_{\mathsf{NX}}]$ and for any Consistent LuffaS-Irreducible view $\mathcal{V}$ obtain a lower bound on number of corresponding Good Consistent NXLuffaS-Irreducible view $\mathcal{V}_{\mathsf{NX}}$.

Definition 12. *For an attacker $\mathcal{A}'$, interacting with $(\mathsf{NXLuffaS}^{\pi}, \pi)$, the view*

$$\mathcal{V}_{\mathsf{NX}} = ((M_1, g_1), \cdots, (M_{q_0}, g_{q_0}), (X_1, Y_1), \cdots, (X_{q^1+q^{-1}}, Y_{q^1+q^{-1}})) \tag{7}$$

is a Good NXLuffaS*-Irreducible view iff $\mathcal{V}_{\mathsf{NX}}$ is a* NXLuffaS*-Irreducible view and*

- *$g_1, \cdots, g_{q_0}$ are distinct and they do not collide with $\{Y_1, \cdots, Y_{q^1+q^{-1}}\}$*
- *For any $X \in \{IV, X_1, \cdots, X_{q^1+q^{-1}}\}$ there does not exist any $M \in \{0,1\}^n$ such that,*
  $\mathsf{MI}_{\mathsf{Lf}}(g_i, M) = X$ for any $i \in \{1, \cdots, q_0\}$.

The output view $\mathcal{OV}_{\mathsf{NX}}$ is called Good NXLuffaS*-Irreducible output view, if the corresponding view $\mathcal{V}_{\mathsf{NX}}$ is Good* NXLuffaS*-Irreducible.*

To obtain a lower bound on $\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{NXLuffaS}^{\pi},\pi)} = \mathcal{OV}_{\mathsf{NX}}]$ we follow an approach similar to proof of Theorem 3. However, for any $z \in \{0,1\}^{nt}$, $m \in \{0,1\}^n$ there exist an unique $y \in \{0,1\}^{nt}$ such that, $\mathsf{MI}_{\mathsf{Lf}}(y\|m) = z$.[6] As a result while assigning the output value of $\mathsf{MD}^{f^{\pi}}$ one can have at most $2(q+\sigma)$ many forbidden values, where q is the total number of attacker queries and σ is the total number of message blocks present in LuffaS queries. As before, we can obtain the following theorem.

[6] Compared to 2^n many possible $y \in \{0,1\}^{nt}$ in case $\mathsf{MI}_{\mathsf{Fg}}$.

Theorem 6. *For any Consistent Good* NXLuffaS*-Irreducible output view* $\mathcal{OV}_{\mathsf{NX}}$ *(*$\mathcal{V}_{\mathsf{NX}}$ *as in* (7) *being the corresponding the view),*

$$\Pr[\mathcal{OV}^{\mathcal{A}'}_{(\mathsf{NXLuffaS}^{\pi},\pi)} = \mathcal{OV}_{\mathsf{NX}}] \geq \Big(1 - \frac{2\sigma(\sigma+q)}{2^{nt}}\Big)\frac{1}{2^{ntq_0}} \prod_{i=1}^{q^1+q^{-1}} \frac{1}{(2^{nt}-i+1)},$$

where σ *is the total number of message blocks present in* NXLuffaS *queries,* q *is the total number* NXLuffaS *and* π *queries and* $(q+\sigma) < 2^{nt-1}$.

With an approach, similar to proof of Theorem 4 and using the previous observation regarding $\mathsf{MI}_{\mathsf{Lf}}$ one can also have the following theorem.

Theorem 7. *For any Consistent* LuffaS*-Irreducible view* $\mathcal{V}$ *as in* (3) *(with* $t = 1$*), there exist at least*

$$2^{n(t-1)q_0}\Big(1 - \frac{q^2}{2^{n(t-2)}}\Big)$$

many Consistent Good NXLuffaS*-Irreducible view* $\mathcal{V}_{\mathsf{NX}}$ *as in* (7) *such that,* $\mathsf{Xor}(g_1) = h_1, \cdots,$
$\mathsf{Xor}(g_{q_0}) = h_{q_0}$*, where* q *is the total number of* LuffaS *and* π *queries.*

Following the final discussion of Section 4.4 we get the following. For any Consistent output view $\mathcal{OV}$(with respect to $\mathsf{LuffaS}^{\pi}, \pi$ oracles) we have,

$$\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{LuffaS}^{\pi},\pi)} = \mathcal{OV}] \geq \Big(1 - \frac{2\sigma(\sigma+q)}{2^{nt}}\Big)\Big(1 - \frac{q^2}{2^{n(t-2)}}\Big)\Pr[\mathcal{OV}^{\mathcal{A}}_{(R,S^R)} = \mathcal{OV}].$$

Applying the above inequality together with $\Pr[\mathsf{Abort}]$ to Theorem 1 we prove Theorem 5.

6 Indifferentiability Security Analysis of Luffa

Theorem 8. *Let* $\Pi = (\pi_1, \pi_2, \cdots, \pi_t)$ *be a collection of independent random permutations over* $\{0,1\}^n$*. Let* Luffa^{Π} *be the* Luffa *mode of operation with* n *bit message blocks and* n *bit digest. There exists a simulator* $S^{\mathcal{R}}$ *such that for any adversary* $\mathcal{A}$ *which makes at most* q *queries*

$$\mathsf{Adv}_{\mathcal{A}}((\mathsf{Luffa}^{\Pi}, \Pi), (\mathcal{R}, S^{\mathcal{R}})) \leq \frac{\mathcal{O}(\sigma^2+q^2)}{2^n} + \frac{\mathcal{O}(q^4)}{2^n}$$

where σ *is the total number of blocks in the queries to* R *or* Luffa^{Π}.

6.1 Simulator for Luffa

In this section we describe the simulator for Luffa mode of operation. As in Section 5, we view the $MI_{\mathsf{Lf}}(Y, m)$ transformation as an affine function $TY + Am$ where T is a $t \times t$ square matrix, A is a coefficient vector of length t and the chaining value Y is a vector of length t. For each permutation $\pi_j;\ j = 1, 2, \cdots, t$

the simulator keeps a partial permutation π_j^* of input-output relations derived so far. By $\mathcal{X}(\pi_j^*)$ and $\mathcal{Y}(\pi_j^*)$ we denote the set of forward query input and the set of forward query output for partial permutation π_j^*, derived so far. Simulator also maintains a directed graph G whose vertex set $V \subseteq \{0,1\}^{tn}$. Initially $V = \{(IV_1, \cdots, IV_t)\}$ These vertices are actually tuple of the outputs of the simulator that are part a chain starting from $IV_{\mathsf{Luffa}} = (IV_1, \cdots, IV_t)$. An edge from vertex Y to vertex Y', marked by m, implies that for each $j \in \{1, \cdots, t\}$, $(T_jY + A_jm, Y'_j) \in \pi_j^*$. T_j represents the j^{th} row of matrix T.

While answering a new forward query x for permutation i, the simulator first checks whether x can be an input of the finalization stage of any chain in the graph. To find it, simulator searches for each existing node Y whether $x = T_iY$ (Recall that for finalization, $m = 0^n$) and for $t-2$ many $j \neq i$, $T_jY \in \mathcal{X}(\pi_j^*)$ ($S(+, i, x)$ is the last but one query of the finalization). If simulator can find such a unique Y, it queries $R(M)$ and tries to set the output of all other permutation maintaining consistency with R. For example, when $t = 3$, the simulator checks whether there exists a state Y and an $x_j \in \mathcal{X}(\pi_j^*)$ for some $j \neq i$. If such a node exists, simulator finds a valid padded message M traversing the path from IV to Y, get $R(M)$ and set the output $y_i = \pi_i^*(x)$ and $y_k = \pi_k^*(T_kY)$ $(k \neq i, j)$ so that $y_i \oplus y_j \oplus y_k = R(M)$ $((x_j, y_j) \in \pi_j^*)$. Simulator also checks whether this new (y_1, y_2, y_3) state maintains some properties of the tree.

If, x is not the penultimate query of a finalization stage, simulator checks whether output of x can make a new node in a chain in the graph; i. e. whether there is a node Y in the graph such that for some m, $x = T_jY + A_jm$ and for all other $i \neq j, T_iY + A_im \in \mathcal{X}(\pi_j^*)$. If no such node exists, the simulator outputs randomly only maintaining the permutation property. If such a node exists, the simulator tries to sample the output so that the new node Y' maintains the following properties of the tree:

- **No cycle** Let Y' be the new node. For any other node $Y = (y_1, \cdots, y_t)$, in the graph, for any $m \in \{0,1\}^n$, $\exists j \in \{1, \cdots, t\}$, $y_j \neq \pi_j^*(T_jY' + A_jm)$.
- **No Partial Collision** For any two nodes Y, Y' in the graph, there is no message $m_1, m_2 \in \{0,1\}^n$, such that for any two $i, j \in \{1, \cdots, t\}$, $T_jY + A_jm_1 = T_jY' + A_jm_2$ and $T_iY + A_im_1 = T_iY' + A_im_2$.
- **Free final inputs** For any two state Y, Y' in the graph and for all $j \in \{1, ..., t\}$, $T_jY \neq T_jY'$.

In Figure 3 dotted lines indicate possible violation of the tree properties on creating the darkened node. Overall the simulator samples a y maintaining the permutation property and checks that the following **Bad** events do not occur.

- The properties of tree are not satisfied.
- **Final input collides with previous input:** For any $j \in \{1, \cdots, t\}$, $T_jY' \in \mathcal{X}(\pi_j^*)$.
- **Partial collision with non-chain inputs:** $\exists m \in \{0,1\}^n$ and $\exists i, j \in \{1, \cdots, t\}$ such that $T_iY' + A_im \in \mathcal{X}(\pi_i^*)$ and $T_jY' + A_jm \in \mathcal{X}(\pi_j^*)$.

For the inverse query of permutation i, the simulator tries to ensure that the output does not create a new node in the graph. For each existing node Y and for all $j \neq i$, consider the set $\Gamma_2(Y, j) = \{x' = T_i Y + A_i m | m = A_j^{-1}(x_j + T_j Y), x_j \in \mathcal{X}(\pi_j^*)\}$. Note that as A is a vector, A_j is an element from the underlying field. On input y simulator samples x from $\{0,1\}^n \setminus \mathcal{X}(\pi_j^*)$, and sets the **Bad** true if $x \in \Gamma_2(Y, j)$ for some node Y and for some $j \neq i$. Formal description of the simulator is given in Algorithm 3 and Algorithm 4 in Appendix E.

In Appendix D, we find an upper bound for the abort probability of the simulator and we have the following lemma.

Lemma 2. *For any attacker $\mathcal{A}$ interacting with $(S^{\mathcal{R}}, \mathcal{R})$ making at most q queries to the simulator,*

$$\Pr[S^{\mathcal{R}} \rightarrow \perp] \leq \frac{\mathcal{O}(q^4)}{2^n - q}.$$

6.2 Interpolation Probability of $\mathcal{OV}^{\mathcal{A}}_{(\mathsf{Luffa}_{\Pi}, \Pi)}$

With an approach similar to Section 4.4 and Section 5.2 one can show, for any Consistent Luffa-Irreducible output view $\mathcal{OV}$ we have,

$$\Pr[\mathcal{OV}^{\mathcal{A}}_{(\mathsf{Luffa}_{\Pi}, \Pi)} = \mathcal{OV}] \geq \left(1 - \frac{2t\sigma^2}{2^n}\right)\left(1 - \frac{4q^2}{2^n}\right) \Pr[\mathcal{OV}^{\mathcal{A}}_{(R, S^R)} = \mathcal{OV}].$$

Applying the above inequality together with Lemma 2 to Theorem 1 we prove Theorem 8.

7 Conclusion

In this paper, we proved indifferentiability of domain extension algorithms of Fugue and Luffa. Although, none of them are in the final round, the modes of these hash functions are interesting in their own right. Specifically, domain extension algorithm of Luffa opens up a interesting research direction regarding security efficiency tradeoff. Improving our bound for Luffa remains an interesting open problem too. However, such an analysis seems to need substantial insight.

References

1. Elena Andreeva, Bart Mennink, and Bart Preneel. On the indifferentiability of the grøstl hash function. In *SCN*, pages 88–105, 2010.
2. Elena Andreeva, Bart Mennink, and Bart Preneel. Security reductions of the second round sha-3 candidates. Cryptology ePrint Archive, Report 2010/381, 2010. http://eprint.iacr.org/.
3. Jean-Phillip Aumasson and Raphael C. W. Phan. Distinguisher for the full final round of fugue-256. NIST Second Sha3 Conference, 2010.

4. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *EUROCRYPT*, pages 181–197, 2008.
5. Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Security analysis of the mode of jh hash function. In *FSE*, pages 168–191, 2010.
6. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
7. Christophe De Canniere, Hisayoshi Sato, and Dai Watanabe. *Hash Function Luffa: Specification Ver 2.0.1 @Sha3 Zoo*, 2009.
8. Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable security analysis of popular hash functions with prefix-free padding. In *ASIACRYPT*, pages 283–298, 2006.
9. Donghoon Chang and Mridul Nandi. Improved indifferentiability security analysis of chopmd hash function. In *FSE*, pages 429–443, 2008.
10. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In *CRYPTO*, pages 430–448, 2005.
11. Shai Halevi, William E. Hall, and Charanjit S. Jutla. *The Hash Function "Fugue" @Sha3 Zoo*, 2009.
12. Shai Halevi, William E. Hall, Charanjit S. Jutla, and Arnab Roy. Weak ideal functionalities for designing random oracles with application to fugue. Sha3 Zoo, 2010.
13. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC*, pages 21–39, 2004.
14. Mridul Nandi. A simple and unified method of proving indistinguishability. In *INDOCRYPT*, pages 317–334, 2006.

A Proof of Theorem 3

For a Consistent Good $\mathsf{NCFugue}$-Irreducible output view $\mathcal{OV}_{\mathsf{NC}}$, $\mathcal{V}_{\mathsf{NC}}$ as in (6) be the corresponding attacker view for the attacker $\mathcal{A}'$. The probability of the attacker view $\mathcal{V}_{\mathsf{NC}}$ is in fact same as the probability that the relations (8), (9) and (10) get satisfied for random permutations π_1 and π_2.

$$\mathsf{NCFugue}^{(\pi_1,\pi_2)}(M_1) = h_1, \cdots, \mathsf{NCFugue}^{(\pi_1,\pi_2)}(M_{q_0}) = h_{q_0} \tag{8}$$

$$\pi_1(x_1^1) = y_1^1, \cdots, \pi_1(x_{q^1+q^{-1}}^1) = y_{q^1+q^{-1}}^1 \tag{9}$$

$$\pi_2(x_1^2) = y_1^2, \cdots, \pi_2(x_{q^2+q^{-2}}^2) = y_{q^2+q^{-2}}^2 \tag{10}$$

We will give a lower bound on number of permutation pairs (π_1, π_2) such that relations (8), (9) and (10) are satisfied. For each i in $[1, q_0]$ we write $\mathsf{Pad}_{\mathsf{Fg}}(M_i)$ as

$$\mathsf{Pad}_{\mathsf{Fg}}(M_i) = m_1^i \| \cdots \| m_{\ell_i}^i.$$

By P_i we denote the set of prefixes of $\mathsf{Pad}_{\mathsf{Fg}}(M_i)$. $EP_i \subseteq P_i$ be the set of evaluatable prefixes of $\mathsf{Pad}_{\mathsf{Fg}}(M_i)$, in other words $x \in EP_i$ if and only if $\mathsf{MD}^{g^{\pi_1}}(x)$ is evaluatable by the relation (9) and $x \in P_i$. $\mathcal{V}_{\mathsf{NC}}$ being a $\mathsf{NCFugue}$-Irreducible Good view either $\mathsf{Pad}_{\mathsf{Fg}}(M_i) \notin EP_i$ or $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i)) \notin \{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$.

For each $i \in [1, q_0]$, if $\mathsf{Pad}_{\mathsf{Fg}}(M_i) \notin EP_i$, we start assigning the output of $\mathsf{MD}^{g^{\pi_1}}(x)$ for each $x \in P_i \backslash EP_i$ in a sequential manner, such that $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i)) \notin \{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$ and $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i))$ values are different.[7] Initially by relation (9), π_1 is already defined at $q^1 + q^{-1}$ points. Also initially, for each $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i))$ there is one fixed *starting input point*.[8] Each time, we assign a output value of $\mathsf{MD}^{g^{\pi_1}}(x)$, π_1 gets defined at one more point.

When assigning output of $\mathsf{MD}^{g^{\pi_1}}(x)$ we are actually free to choose any nt-bit value outside the set of currently defined output values of π_1. However, we need to be careful. For some $x \in P_i \setminus EP_i$ and $x \neq \mathsf{Pad}_{\mathsf{Fg}}(M_i)$, if we assign the output of $\mathsf{MD}^{g^{\pi_1}}(x)$ in a way such that the element $\mathsf{MI}_{\mathsf{Fg}}(\mathsf{MD}^{g^{\pi_1}}(x)\|m)$ is already a defined input point of π_1 and $x\|m \in P_i \setminus EP_i$, we are in a problematic scenario. We won't be able to assign the value of $\mathsf{MD}^{g^{\pi_1}}(x\|m)$ in the next step, failing to ensure $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i)) \notin \{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$. Not only this, one should also ensure $\mathsf{MI}_{\mathsf{Fg}}(\mathsf{MD}^{g^{\pi_1}}(x)\|m)$ do not collide with the possible q_0 many starting input points and $\mathsf{MD}^{g^{\pi_1}}(x)$ do not collide with $\{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$.[9]

Let us recall, for any $z \in \{0,1\}^{nt}, m \in \{0,1\}^n$ there exist at most 2^n many $y \in \{0,1\}^{nt}$ such that,

$$\mathsf{MI}_{\mathsf{Fg}}(y\|m) = z.$$

Hence, at any stage if π_1 is already defined at k many input/output points, while assigning the output of $\mathsf{MD}^{g^{\pi_1}}(x)$ we are forbidden to choose the output value from

- the currently defined k output points, the set $\{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$ and
- If $x \neq \mathsf{Pad}_{\mathsf{Fg}}(M_i)$ for any $i \in [1, q_0]$: at most $(k + q_0 + 1)2^n$ many possible points which might lead to the previously described problem.[10]

This means, while assigning the output of $\mathsf{MD}^{g^{\pi_1}}(x)$, if π_1 is currently defined at k-many input/output points; we have at least

$$(2^{nt} - \overline{k + q^2 + q^{-2} + (q_0 + k + 1)2^n}) = (2^{nt} - k)(1 - \frac{q^2 + q^{-2} + (q_0 + k + 1)2^n}{2^{nt} - k})$$

[7] Sequentially means, if $x_1, x_2 \in P_i \setminus EP_i$ and x_1 is a suffix of x_2 we assign output of $\mathsf{MD}^{g^{\pi_1}}(x_1)$ before $\mathsf{MD}^{g^{\pi_1}}(x_2)$.

[8] If $x\|m$ is the smallest element of $P_i \setminus EP_i$, $\mathsf{MI}_{\mathsf{Fg}}(\mathsf{MD}^{g^{\pi_1}}(x)\|m)$ is the *starting input point* for $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i))$. If there is a collision between the starting input points (e.g. if padded messages have common suffixes), after assignment of $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i))$, starting input point corresponding to $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_j))$ will get changed. However, as we are dealing with Consistent views that is not a problem.

[9] Normally, we do not want a collision between $\mathsf{MD}^{g^{\pi_1}}(x)$ and $\{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$ only when $x = \mathsf{Pad}_{\mathsf{Fg}}(M_i)$. However, it might happen $\mathsf{Pad}_{\mathsf{Fg}}(M_j)$ is a prefix of $\mathsf{Pad}_{\mathsf{Fg}}(M_i)$ for some $i < j$. Hence to play safe, we always prevent collision between $\mathsf{MD}^{g^{\pi_1}}(x)$ and $\{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$.

[10] $(k + q_0 + 1)$, because there are previously defined k input points, at most q_0 many starting input points and the current input point.

many choices. As $\mathcal{V}_{\mathsf{NC}}$ is a Consistent view, the above assignment strategy always make sure after completion of π_1 assignment for all $i \in [1, q_0]$,

$$\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i)) \notin \{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$$

and $\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i))$ values are different. Also, $\mathcal{V}_{\mathsf{NC}}$ being a Good view, $r_1, \cdots, r_{q_0}$ are all different and they are different from already defined π_2 output values $y_1^2, \cdots, y_{q^2+q^{-2}}^2$. Hence, we can assign $\pi_2(\mathsf{MD}^{g^{\pi_1}}(\mathsf{Pad}_{\mathsf{Fg}}(M_i)))$ output as r_i without any problem. If σ is the total number of message blocks present in $\mathsf{NCFugue}$ queries and $q = q_0 + q^1 + q^{-1} + q^2 + q^{-2}$ is the total number of queries we have at least

$$(2^{nt}!)^2(1 - \frac{2\sigma(q+\sigma)2^n}{2^{nt}})\frac{1}{2^{ntq_0}} \prod_{i=1}^{q^1+q^{-1}} \prod_{j=1}^{q^2+q^{-2}} \frac{1}{(2^{nt}-i+1)(2^{nt}-j+1)}$$

many permutation pairs (π_1, π_2) such that relations (8), (9) and (10) are satisfied (assuming $q + \sigma < 2^{nt-1}$). As, there are $(2^{nt}!)^2$ many permutation pairs without any constrain we get the lower bound as claimed in 3.

B Proof of Theorem 4

Now, we would like to have a lower bound on number of $\mathsf{NCFugue}$-Irreducible, Consistent Good views $\mathcal{V}_{\mathsf{NC}}$ (as in (6)), corresponding to a Fugue-Irreducible Consistent view $\mathcal{V}$ (as in, (3) with $t = 2$) such that the corresponding output view satisfy equation (5). We know, n_h-many bits of r_i should be same as h_i, for the rest of the bits we have 2^{nt-n_h} many possible choices. However, we are forbidden to choose r_i values from $\{x_1^2, \cdots, x_{q^2+q^{-2}}^2\}$ and r_i's should be collision free. Hence for each r_i we have at least

$$(2^{nt-n_h} - (q_0 + q^2 + q^{-2}))$$

many choices. Hence we get the result as claimed in the theorem.

C Proof of Lemma 1

We start with the observation that each query to the simulator can introduce at most one node to the graph G. So before i^{th} query there is at most i nodes in G. For each existing node Y', there can be at most 2^n many values of $Y' + T^{-1}Am$. For a forward finalization query, total number of possible outputs of Y, conditioned on $R(M)$, is $2^{(t-1)n} - q$. For any other query, total number of possible outputs of Y is $2^{tn} - q$. As, to answer a forward query, Y is chosen uniformly at random from $\{0,1\}^{tn} \setminus \mathcal{Y}(\pi^*)$ (conditioned on $R(M)$ for finalization queries), $\Pr[\mathbf{Bad}_{\mathsf{Lf}}^{+1}] = \Pr[Y = Y' + T^{-1}Am] \leq \frac{i2^n}{2^{(t-1)n}-q}$. Similarly, for each $x \in \mathcal{X}(\pi^*)$, there can be at most 2^n values of $x + Am$. So at the i^{th} query

$\Pr[\mathbf{Bad}^{+2}_{\mathsf{Lf}}] = \Pr[Y = x + Am | x \in \mathcal{X}(\pi^*), m \in \{0,1\}^n] \leq \frac{i2^n}{2^{(t-1)n}-q}$. Finally for the inverse query, out of $2^{tn} - q$ possible outputs, there can be at most $i2^n$ values which sets $\mathbf{Bad}^{-1}_{\mathsf{Lf}}$ true.

Hence $\Pr[\mathbf{Bad}_{\mathsf{Lf}}] \leq \sum_i (\frac{\mathcal{O}(i2^n)}{2^{(t-1)n}-q}) = \frac{\mathcal{O}(q^2)}{2^{(t-2)n}}$, assuming $q \leq 2^n$.

D Proof of Lemma 2

Let $\overline{BAD_i}$ denote the event that **Bad** has happened in one of the first i queries and $\mathbf{Bad}_i$ denotes the event that **Bad** happens in the i^{th} query. Clearly $\Pr[\mathbf{Bad}] \leq \sum_i \Pr[\mathbf{Bad}_i | \neg\overline{BAD_{i-1}}]$.

First, we consider the case when i^{th} query is a forward query and $\overline{BAD_{(i-1)}}$ is false. Hence before the i^{th} query to the simulator, all the properties of simulator graph was satisfied.

Note that, for any two node Y, Y', for all $1 \leq j \leq t$, $T_jY \neq T_jY'$ implies $|\Gamma_0| \leq 1$. Moreover If x was a final input of the finalization phase with $\Gamma_0 = (Y, m, Count)$ and its value was not yet set, then one can show that while creating the node Y, a **Bad** event was set. As $\overline{BAD_{(i-1)}}$ is true, this situation can never happen. Hence in Algorithm 3 $\mathbf{Bad}_i$ can only be set in Step 31 when inserting the final node or in Step 5 in Algorithm 2.

First we find the probability of a new node Y' to break the properties of the tree. Let σ_i be the number of nodes in the graph before i^{th} query. As T is a linear transformation, probability that for some m, $TY' + Am$ is the inputs of a previous state is at most $\frac{\sigma_i}{2^n-q}$. Probability for partial collision can also be bounded by $\frac{\binom{t}{2}\sigma_i}{2^n-q}$. Finally, probability of the breaking the free final input is at most $\frac{t\sigma_i}{2^n-q}$. For the other **Bad** events, probability that for some m and for some $1 \leq j, k \leq t$, $T_iY' + A_im \in \mathcal{X}(\pi^*_j)$ and $T_kY' + A_km \in \mathcal{X}(\pi^*_k)$ is at most $\frac{\binom{t}{2}q^2}{2^n-q}$. Finally probability of future final input to collide with previous query is at most $\frac{q}{2^n-q}$.

By the property of no pseudo-collision, one can argue that each of x and x_j in Algorithm 3 can be input of at most q states. Hence $|\Gamma_1| \leq q$ and $|\Gamma^j_1| \leq q$. So the probability of **Bad** in Algorithm 2 is at most $\frac{2q(q+\binom{t}{2}q^2+(1+t+\binom{t}{2})\sigma_i)}{2^n-q}$.

If the i^{th} query is an inverse query, one can bound the probability of **Bad** by $\frac{q\sigma_i}{2^n-q}$. As each query can introduce at most q states in the graph, $\sigma_i \leq q^2$. Hence $\Pr[\mathbf{Bad}_i | \neg\overline{BAD_{i-1}}] \leq \frac{\mathcal{O}(q^3)}{2^n-q}$. Taking sum over i, we get the desired bound. □

E Algorithms for Luffa Simulator

Algorithm 1 FindState(x, i): Find the states in the graph such that x can be the next query of a chain to permutation π_i

$\Gamma_0 = \emptyset, \Gamma_1 = \emptyset$.
for all vertex Y in the graph **do**
  Compute $m = {A_i}^{-1}(x + T_i Y)$
  Count=0;
  for $k = 1$ to t; $i \neq k$ **do**
    Compute $x'_k = T_k Y + A_k m$
    if $x'_k \in \mathcal{X}(\pi^*_k)$ **then**
      Count+=1;
    end if
  end for
  if $Count \geq t - 2$ and $m = 0^n$ **then**
    Add $(Y, m, Count)$ in List Γ_0.
  end if
  if $Count = t - 1$ **then**
    $x'_i = x$
    Add $(y, m, x'_1, \cdots, x'_t)$ in List Γ_1.
  end if
end for
return (Γ_0, Γ_1).

Algorithm 2 CheckOther(Γ,j,y_j): Check whether y_j sets any **Bad** event true

for each $(Y', m', x'_1, \cdots, x'_t) \in \Gamma$ **do**
  For all $i = 1$ to t, $i \neq j$, Retrieve y'_i such that $(x'_i, y'_i) \in \pi^*_i$.
  $y'_j = y_j$.
  Construct $Y_1 = (y'_1, \cdots, y'_t)$.
  if Y_1 sets **Bad** True **then**
    return 1.
  else
    Create node Y_1 in the graph
    Add edge (Y', Y_1) with label m'.
  end if
end for
return 0.

Algorithm 3 $S^{\mathcal{R}}(i,+,x)$: Forward query simulator for Luffa

if $(x,y) \in \pi_i^*$ for some y **then**
  return y.
end if
(Γ_0, Γ_1)=FindState(x,i).
if $|\Gamma_0| > 1$ **then** {/*For two nodes one of the final output matched*/}
  Bad=True
  return $\perp$
end if
if $|\Gamma_0| = 1$ **then**
  $(Y, m, Count) \leftarrow \Gamma_0$
  if Count=t-1 **then** {/*For finalization all but one output is already set*/}
    Bad=True
    return $\perp$
  end if
  Retrieve M from the path from IV to Y.
  $z = \mathcal{R}(M)$.
  for each $k = 1$ to t, $k \neq i$ **do**
    $x'_k = T_k Y$
    if $(x'_k, y'_k) \in \pi_k^*$ for some y'_k **then**
      $z = z + y'_k$.
    else {The input which is not yet queried}
      $x_j = x'_k$ and $j = k'$
    end if
  end for
  repeat
    $y'_j \leftarrow \{0,1\}^n \setminus \mathcal{Y}(\pi_j^*)$
  until $z + y'_j \notin \mathcal{Y}(\pi_i^*)$
  $y = z + y_j$.
  $y'_i = y$.
  $Y_1 = (y'_1, \cdots, y'_t)$.
  if Y_1 sets **Bad** true **then**
    return $\perp$
  else
    Create node Y_1 in the graph.
    Add edge (Y, Y_1) with label 0^n.
  end if
  {/*Check whether y_j induces any Bad event by creating new states*/}
  (Γ_0^j, Γ_1^j)=FindState(x_j, j).
  temp= CheckOther(Γ_1^j, j, y_j);
  if temp=1 **then**
    return $\perp$
  end if
  Add (x_j, y_j) to π_j^*.
  Add (x,y) to π_i^*.
  return y.
else
  $y \leftarrow \{0,1\}^n \setminus \mathcal{Y}(\pi_i^*)$.
end if
if $|\Gamma_1| \geq 0$ **then** {/*Check whether y sets any Bad event by creating new node*/}

Algorithm 4 $S^{\mathcal{R}}(i,-,y)$

if $(x,y) \in \pi_i^*$ for some x **then**
  return x.
end if
$\Gamma_2 = \emptyset$.
for all node Y in the graph **do**
  for all $j = 1$ to t, $j \neq i$, for all $x_j \in \mathcal{X}(\pi_j^*)$ **do**
    $m = m = A_j^{-1}(x_j + T_j Y)$.
    $x' = T_i Y + A_i m$.
    Put x' in Γ_2.
  end for
end for
$x \leftarrow_R \{0,1\}^n \setminus \mathcal{X}(\pi_i^*)$.
if $x \in \Gamma_2$ **then**
  Bad=True
  return $\perp$
else
  Add (x,y) to π_i^*.
  return x.
end if

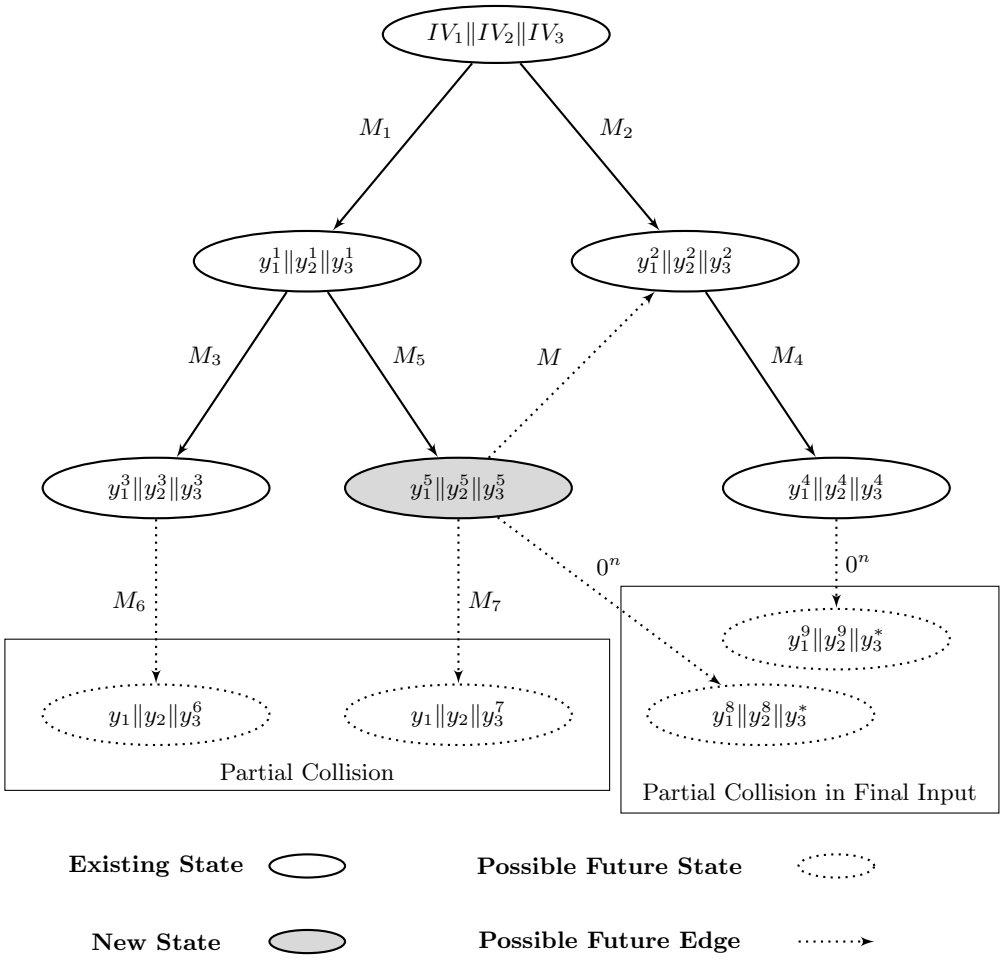

Fig. 3. The Simulator graph for Luffa with **Bad** events