Improved Indifferentiability Security Bound for the JH Mode *

Dustin Moody

National Institute of
Standards and Technology
Gaithersburg, MD, USA
dustin.moody@nist.gov

Souradyuti Paul

National Institute of
Standards and Technology
Gaithersburg, MD, USA
&
K.U.Leuven, Belgium
souradyuti.paul@nist.gov

Daniel Smith-Tone

National Institute of
Standards and Technology
Gaithersburg, MD, USA
daniel.smith@nist.gov

Abstract

The JH hash function is one of the five finalists of the ongoing NIST SHA3 hash function competition. Despite several earlier attempts, and years of analysis, the indifferentiability security bound of the JH mode has so far remained remarkably low, only up to $n/3$ bits [7]. Using a recent technique introduced by Moody, Paul, and Smith-Tone in [23], we improve the bound of JH to $n/2$ bits. We also performed experiments which verify the theoretically obtained results.

1 Introduction

Iterative hash functions are usually composed of two parts: (1) a basic primitive C with finite domain and range, and (2) an iterative mode of operation H to extend the domain of the hash function. We denote a hash function using the mode H and the primitive C by H^C. In studying the security of a hash function, both the security of the primitive C, as well as the security of the mode of operation H need to be examined separately, since one can be attacked independently of the other.

The most popular hash mode of operation is the classical Merkle-Damgård MD mode [12, 22]. It has the desirable property that if C is collision resistant, then MD^C will also be collision resistant. Many practical hash functions, such as MD4 [28], MD5 [29], SHA-0/1/2 [25, 26] use the MD mode of operation. However, several recent attacks have greatly undermined the security of any hash function based on the MD mode. Examples of such attacks include the length-extension attack, Joux's multi-collision attack [16], the Kelsey-Kohno herding attack [17], and the Kelsey-Schneier preimage attack [18]. Each of these attacks targets the mode of operation, and they work no matter how secure the underlying primitive C is. A telltale sign of the demise of the MD mode is that none of the 64 submissions to the ongoing SHA-3 competition used the MD mode.

In search of a secure replacement for the Merkle-Damgård mode, we have broadly witnessed several stages of improvement: (1) additional postprocessing and/or counters injected into the MD mode to eliminate the length-adjustment related attacks (e.g., HAIFA [8], EMD [4], MDP [15]); (2) widening of the output length of the primitive C to eliminate Joux's multi-collision type attacks (e.g., Widepipe-MD [20], JH [30], Grøstl [14], Sponge [5], Shabal [10], Parazoa [3]); (3) multiple applications of the primitive C on the same message-block (e.g., Doublepipe MD [20]).

Another research direction, motivated by the innovative attacks on hash modes of operation, has been the development of new security frameworks which cover the above attacks, as well as unforeseen attacks. The indifferentiability framework was introduced by Maurer *et al.* [21] in 2004, and was first applied to analyze

*The extended version of this paper is available at `http://www.esat.kuleuven.be/scd/person.php?persid=111`

hash modes of operation by Coron *et al.* [11] in 2005. Indifferentiability measures the extent to which a hash function is behaving as a random oracle under the assumption that the underlying compression function is ideal (e.g. a random oracle, ideal permutation, or ideal cipher). A hash mode proven secure in the indifferentiability framework guarantees resistance to the attacks specifically mentioned above. Most new proposals for hash modes of operation include indifferentiability proofs of security. We note that some limitations of the indifferentiability framework have recently been discovered in [13] and [27]. Nevertheless, the framework still covers most known attack scenarios, in addition to guaranteeing resistance to many generic attacks. In this work we prove an indifferentiability bound for JH, one of the five finalists in the SHA-3 hash competition.

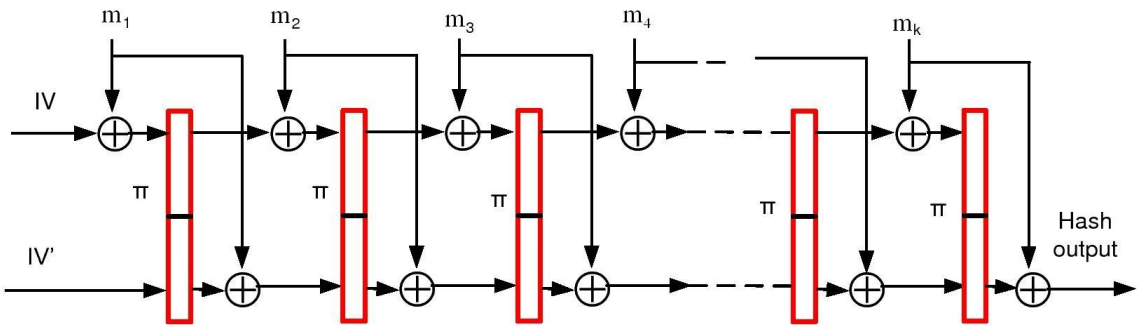


Figure 1: JH mode diagram: all wires are n bits.

Related Work: Since its publication, the JH mode of operation has undergone rigorous cryptanalysis. Previous indifferentiability security proofs for the JH mode were only able to obtain a bound of $n/3$ bits [2, 7]. In [19], it was shown that the JH mode achieves optimal collision resistance. However, improvement of the indifferentiability security of the JH mode beyond $n/3$ bits has remained elusive.

Our Contribution: We improve the indifferentiability security bound of the JH mode from $n/3$ to $n/2$ bits. This result guarantees the absence of generic attacks on the JH hash function with work $\Omega(2^{n/2})$. Furthermore, we have performed a series of experiments with the JH mode using several sets of 'bad' events. These bad events will be described later on. Our experiments verify the theoretically obtained results.

Notation and Convention. Throughout the paper we let n be a fixed integer. We shall use the little-endian bit-ordering system. The symbol $|x|$ denotes the bit-length of the bit-string x, or sometimes the size of the set x. Let $x \xrightarrow{parse} x_1||x_2$ denote parsing x into x_1 and x_2 such that $|x_1| = n$ and $|x_2| = |x| - n$. Let S_X denote the sample space of the discrete random variable X. The relation $A \sim B$ is satisfied if and only if $\Pr[A = X] = \Pr[B = X]$ for all $X \in S$, where $S = S_A = S_B$. Other notation which we will use is included in Table 1.

Table 1: Notation

$\mathcal{A}^B$	Algorithm $\mathcal{A}$ with oracle access to B
$Dom(T)$	The set of indices I in table T such that $T[i] \neq \perp\ \forall i \in I$
ab	$a\|\|b$
$[x, y]$	The set of integers between (and including) x and y
$a[x, y]$	The bit-string between the x^{th} and y^{th} bit-positions of a
$\mathcal{U}[0, N]$	The uniform distribution over the integers between 0 and N

1.1 Description of the JH Mode

JH(M)
01. $M \xrightarrow{pad} m_1 m_2 \dots m_{k-1} m_k$;
02. $y_0 = IV$, $y'_0 = IV'$;
03. for$(i = 1, 2, \dots k)$
$y_i y'_i = \pi\Big(y_{i-1} || (y'_{i-1} \oplus m_i)\Big) \oplus m_i || 0$;
04. return y_k;

Figure 2: Pseudocode for the JH hash function

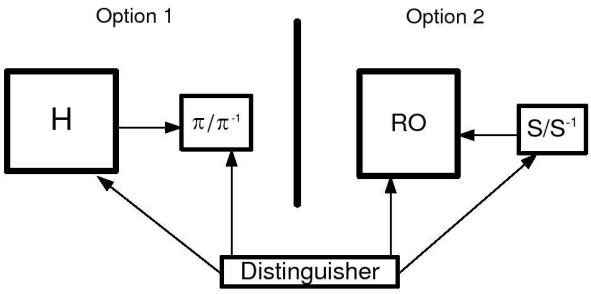


Figure 3: Indifferentiability framework for a hash function based on an ideal permutation. An arrow indicates the direction in which a query is submitted.

Suppose $n \geq 1$. Let $\pi : \{0,1\}^{2n} \to \{0,1\}^{2n}$ be a $2n$-bit ideal permutation used to build the JH hash function $\mathrm{JH}^{\pi} : \{0,1\}^* \to \{0,1\}^n$. The diagram and the description of the JH transform are given in Figures 1 and 2. The semantics for the notation $M \xrightarrow{pad} m_1 \cdots m_{k-1} m_k$ is as follows: Using an injective function $\mathsf{pad} : \{0,1\}^* \to \cup_{i \geq 1} \{0,1\}^{ni}$, M is mapped into a string $m_1 \cdots m_{k-1} m_k$ such that $k \geq \left\lceil \frac{|M|}{n} \right\rceil + 1$, $|m_i| = n$ for $1 \leq i \leq k$. In addition to the injectivity of $\mathsf{pad}(\cdot)$, we will also require that there exists a function $\mathsf{dePad}(\cdot)$ that can efficiently compute M, given $\mathsf{pad}(M)$. Formally, the function $\mathsf{dePad}\colon \cup_{i \geq 1} \{0,1\}^{in} \to \{\lambda\} \cup \{0,1\}^*$ computes $\mathsf{dePad}(\mathsf{pad}(M)) = M$, for all $M \in \{0,1\}^*$, and otherwise $\mathsf{dePad}(\cdot)$ returns λ. We note that the padding rules of all practical hash functions have the above properties.

1.2 Preliminaries: Introduction to the Indifferentiability Framework

We begin with the definition of a random oracle. This useful object will be used frequently in the subsequent discussion.

Definition 1.1 (Random oracle) *A random oracle is a function $\mathsf{RO} : X \to Y$ chosen uniformly at random from the set of all $|Y|^{|X|}$ functions that map $X \to Y$. In other words, a function $\mathsf{RO} : X \to Y$ is a random oracle if and only if, for each $x \in X$, the value of $\mathsf{RO}(x)$ is chosen uniformly at random from Y.*

Corollary 1.2 *If a function $\mathsf{RO} : X \to Y$ is a random oracle, then*

$$\Pr\big[\mathsf{RO}(x) = y | \mathsf{RO}(x_1) = y_1,\ \mathsf{RO}(x_2) = y_2,\ \dots,\ \mathsf{RO}(x_q) = y_q\ = \frac{1}{|Y|}$$

where $x \notin \{x_1, x_2, \dots, x_q\}$, $y \in Y$ and $q \in \mathbb{Z}$.

Next we introduce the indifferentiability framework and briefly discuss its significance. The definition we give is a slightly modified version of the original definition provided in [11, 21].

Definition 1.3 (Indifferentiability framework) *[11] An interactive Turing machine (ITM) T with oracle access to an ideal primitive $\mathcal{F}$ is said to be $(t_{\mathcal{A}}, t_S, q, \varepsilon)$-indifferentiable from an ideal primitive $\mathcal{G}$ if there exists a simulator S such that, for any distinguisher $\mathcal{A}$, the following equation is satisfied:*

$$|\Pr[\mathcal{A}^{T,\mathcal{F}} = 1] - \Pr[\mathcal{A}^{\mathcal{G},S} = 1]| \leq \varepsilon.$$

The simulator S is an ITM which has oracle access to $\mathcal{G}$ and runs in time at most t_S. The distinguisher $\mathcal{A}$ runs in time at most $t_{\mathcal{A}}$. The number of queries used by $\mathcal{A}$ is at most q. Here ε is a negligible function in the security parameter of T.

We define $\mathbf{Adv}^{\mathcal{G},S}_{T,\mathcal{F}} = \max_{\mathcal{A}} |\Pr[\mathcal{A}^{T,\mathcal{F}} = 1] - \Pr[\mathcal{A}^{\mathcal{G},S} = 1]|$, so that by definition $\mathbf{Adv}^{\mathcal{G},S}_{T,\mathcal{F}} \leq \varepsilon$. The significance of the framework is as follows. Suppose, an ideal primitive $\mathcal{G}$ (e.g. a *variable-input-length* random oracle) is indifferentiable from an algorithm T based on another ideal primitive $\mathcal{F}$ (e.g. a *fixed-input-length* random oracle). In such a case, any cryptographic system $\mathcal{P}$ based on $\mathcal{G}$ is as secure as $\mathcal{P}$ based on $T^{\mathcal{F}}$ (i.e., $\mathcal{G}$ replaces $T^{\mathcal{F}}$ in $\mathcal{P}$). For a more detailed explanation, we refer the reader to [21].

Pictorial Description of Definition 1.3(Figure 3). In the figure, the five entities involved in Definition 1.3 are shown: T, $\mathcal{F}$, $\mathcal{G}$ and S have been replaced by a hash mode H, an ideal permutation π/π^{-1}, a random oracle RO, and a pair of simulators S/S^{-1}. For the purposes of our paper, H is the JH hash mode described in Figure 2, based on the permutation π (see Figure 1). In this setting, Definition 1.3 addresses the degree to which any computationally bounded adversary is unable to distinguish between Option 1 and Option 2.

2 Indifferentiability Framework for JH: Definitions

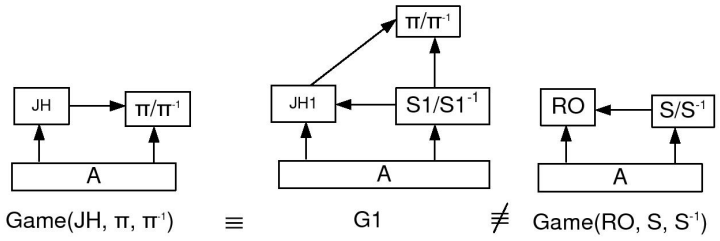


Figure 4: Schematic diagrams of the security games used in the indifferentiability framework for JH. The arrows show the directions in which the queries are submitted.

To study the indifferentiability security of the JH mode, we use the ideal permutation $\pi/\pi^{-1} : \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$ as the basic primitive of JH. To obtain the indifferentiability security bound for the JH mode, we follow the usual game-playing techniques [1, 4]. The schematic diagrams of the two cryptographic systems Option 1 and Option 2 (of Figure 3) are Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}) illustrated in Figure 4. The other game G_1 is an intermediate step, allowing us to more easily compare the games. The pseudocode for each game is provided in Section 3. Informally, a game takes an adversarial query as input and produces an output. A simple example is the description of Game(JH, π, π^{-1}) which is provided in Figure 5(a). We now formally define a game.

Definition 2.1 (Game) *A game is a stateful probabilistic algorithm that takes an adversary-generated query as input, updates the current state, and produces an output to the adversary.*

The notion of *equivalence of games* will play a central role in the security reduction processes in the coming sections. To put it loosely, two games are *equivalent* if their input-output distributions are identical. To make the definition simpler we restrict ourselves *only* to games that expose identical interfaces to their adversaries. This allows us to define *equivalence of a pair of games* when they interact with the same adversary. Before we give the formal definition of equivalent games, we introduce the notion of the *view of a game.*

Definition 2.2 (View of a game) *Let $(x_i y_i)$ denote the i^{th} query and response pair from the game G, when it interacts with the adversary $\mathcal{A}$. The view of the game G after i queries with respect to the adversary $\mathcal{A}$, is defined to be the sequence $\{(x_1 y_1), \ldots, (x_i y_i)\}$.*

Definition 2.3 (Equivalence of games) *Denote the views of the games G_1 and G_2 after i queries by $V_{1,i}$ and $V_{2,i}$, when they are interacting with the adversary $\mathcal{A}$. The games G_1 and G_2 are said to be equivalent*

with respect to the adversary $\mathcal{A}$ if and only if $V_{1,i} \sim V_{2,i}$, *for all* $i > 0$. *Equivalence between the games* G_1 *and* G_2 *with respect to the adversary* $\mathcal{A}$ *is denoted by* $G_1 \overset{\mathcal{A}}{\equiv} G_2$, *or simply* $G_1 \equiv G_2$, *when the adversary is clear from the context.*

Next we state an important lemma relating the *equivalence of games* and the adversarial outputs.

Lemma 2.4 *If* $G_1 \overset{\mathcal{A}}{\equiv} G_2$ *then* $\Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right] = \Pr\left[\mathcal{A}^{G_2} \Rightarrow 1\right]$. *The probabilities are taken over all coin tosses in the games and the adversary.*

Proof. The result follows immediately from Definition 2.3. □

Roadmap. From Section 1.2, we have

$$\mathbf{Adv}^{\mathsf{RO},S/S^{-1}}_{JH,\pi/\pi^{-1}} = \max_{\mathcal{A}} \left| \Pr\left[\mathcal{A}^{\text{Game(JH, } \pi, \pi^{-1})} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\text{Game(}\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})} \Rightarrow 1\right] \right|.$$

In the following sections we shall estimate $\mathbf{Adv}^{\mathsf{RO},S/S^{-1}}_{JH,\pi/\pi^{-1}}$ using the following approach:

- In Section 3 and 4 we describe the security games presented in Figure 4; an important part of the description is designing a simulator-pair $\mathsf{S}/\mathsf{S}^{-1}$ for the indifferentiability framework for JH, and computing an upper bound of $\mathcal{O}(\sigma^5)$ on the running time of the simulator-pair, where σ is the maximum number of invocations to the ideal permutation π/π^{-1}. Additionally, we shall show that Game(JH, π, π^{-1})$\equiv G_1$, which implies by Lemma 2.4, $\Pr\left[\mathcal{A}^{\text{Game(JH, } \pi, \pi^{-1})} \Rightarrow 1\right] = \Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right]$. This, in turn, implies:

$$\mathbf{Adv}^{\mathsf{RO},S/S^{-1}}_{JH,\pi/\pi^{-1}} = \max_{\mathcal{A}} \left| \Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\text{Game(}\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})} \Rightarrow 1\right] \right|.$$

- Then in Section 5 we shall show

$$\max_{\mathcal{A}} \left| \Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\text{Game(}\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})} \Rightarrow 1\right] \right| = \mathcal{O}\left(\frac{\sigma^2}{2^n}\right).$$

The $n/2$ bit indifferentiability security bound follows from this last equation.

3 Description of the Security Games for JH

In this section, we elaborate on the games Game(JH, π, π^{-1}), G_1, and Game(RO, S, S^{-1}) that are schematically presented in Figure 4. The pseudocode for all the games is given in Figure 5. The functions JH, JH1, and RO are mappings from $\{0,1\}^* \to \{0,1\}^n$. We use the generic term *long oracle* (or l-oracle) to identify any of them and call any query submitted to a long oracle an l-query. Also, π, and $\mathsf{S1}$ are permutations on $\{0,1\}^{2n}$. The oracle S is a mapping from $\{0,1\}^{2n} \to \{0,1\}^{2n}$. These three oracles are called *forward short oracles* (or forward s-oracles). The corresponding queries are called forward s-queries. Similarly, π^{-1}, $\mathsf{S1}^{-1}$, and S^{-1} are also permutations on $\{0,1\}^{2n}$ which are called *reverse short oracles* (or reverse s-oracles). We assume that identical queries are not submitted more than once. The games will use several global and local variables. The tables D_l, D_s, and all local variables are initialized with $\perp$. The graph T_s initially contains only the root node (IV, IV'). The local variables are re-initialized every new invocation of the game, while the global data-structures maintain their states across the queries.

Description of Game(JH, π, π^{-1}) (Figure 5(a)). The game Game(JH, π, π^{-1}) implements three functions: the l-oracle JH^{π}, the forward s-oracle π, and the reverse oracle π^{-1}. The definition of the function JH^{π} was given in Section 2. The ideal permutation π/π^{-1} has been implemented through lazy sampling. The query-response pairs for π/π^{-1} are stored in the table D_s. ∎

$\underline{\mathrm{JH}(M)}$

01. $M \xrightarrow{pad} m_1 m_2 \dots m_{k-1} m_k$;
02. $y_0 = IV$, $y'_0 = IV'$;
03. for$(i = 1, 2, \dots k)$
    $y_i y'_i = \pi\Big(y_{i-1} || (y'_{i-1} \oplus m_i) \quad \oplus m_i || 0$;
04. return y_k;

$\underline{\pi(x)}$

11. if $x \notin Dom(D_s)$ then $D_s[x] \xleftarrow{\$} \{0,1\}^{2n} \setminus Rng(D_s)$;
12. return $D_s[x]$;

$\underline{\pi^{-1}(y)}$

21. if $y \notin Rng(D_s)$, $D_s^{-1}[y] \xleftarrow{\$} \{0,1\}^{2n} \setminus Dom(D_s)$;
22. return $D_s^{-1}[y]$;

(a) Game(JH, π, π^{-1})

$\underline{\mathsf{JH1}(M)}$

000. if Type3 BAD=True;
001. if $M \in Dom(D_l)$ return $D_l[M][0, n-1]$;
002. $M \xrightarrow{pad} m_1 m_2 \cdots m_{k-1} m_k$;
003. $y_0 = IV$, $y'_0 = IV'$, $b = 0$;
004. for$(i = 1, \cdots, k-1)$
005. $\quad y''_{i-1} = y'_{i-1} \oplus m_i$;
006. $\quad$ if $y_{i-1} y''_{i-1} \notin Dom(D_s)$ then $b = 1$;
007. $\quad r = \pi(y_{i-1} y''_{i-1})$; $y_i y'_i = r \oplus m_i || 0$;
008. $\quad$ if $b = 1$ $\mathsf{PartialGraph}(y_{i-1} y''_{i-1}, r)$; $b = 0$;
009. $y''_{k-1} = y'_{k-1} \oplus m_k$;
010. if $y_{k-1} y''_{k-1} \notin Dom(D_s)$ then $b = 1$;
011. $r = \pi(y_{k-1} y''_{k-1})$; $D_l[M] = r \oplus m_k || 0$;
012. return $D_l[M][0, n-1]$;
013. if $b = 1$ $\mathsf{PartialGraph}(y_{k-1} y''_{k-1}, r)$;

$\underline{\mathsf{MessageRecon}(x, T'_s)}$

201. $x \xrightarrow{parse} yy'$;
202. if $\mathsf{FindNode}(y) = 0$ return $\mathcal{M} = \emptyset$;
203. $\mathsf{FindBranch}(y) = \mathcal{M}'$;
204. $\mathcal{M} = \{\mathsf{dePad}(Xz) \mid Xz' \in \mathcal{M}', z = z' \oplus y'\}$;
205. return $\mathcal{M}$;

$\underline{\pi(x)}$

301. if $x \notin Dom(D_s)$ $D_s[x] \xleftarrow{\$} \{0,1\}^{2n} \setminus Rng(D_s)$;
302. return $D_s[x]$;

$\underline{\mathsf{S1}(x)}$

100. if Type2-a BAD =True;
101. if $x \in Dom(D_s)$ return $D_s[x]$ else $r = \pi(x)$;
102. if Type2-b BAD =True;
103. $\mathcal{M} = \mathsf{MessageRecon}(x, T'_s)$;
104. if $|\mathcal{M}| = 1 \wedge M \notin Dom(D_l)$ $D_l[M] = r \oplus (z||0)$;
105. return r; $\mathsf{PartialGraph}(x, r)$;

$\underline{\mathsf{PartialGraph}(x, r)}$

401. $x \xrightarrow{parse} y_c y''_c$; $r \xrightarrow{parse} y^* y'$;
402. $\mathsf{Coset} = \mathsf{CreateCoset}(y_c)$;
403. $\mathsf{EdgeNew} = \{(y_c y'_c, m, yy') \mid$
     $y_c y'_c \in \mathsf{Coset}, m = y''_c \oplus y'_c, y = y^* \oplus m\}$;
404. for $(y_c y'_c, m, yy') \in \mathsf{EdgeNew}$ $\{\mathsf{AddEdge}(y_c y'_c, m, yy')$;
405. $\quad$ if Type1-a $\vee$ Type1-b BAD=True; $\}$

$\underline{\mathsf{S1}^{-1}(r)}$

601. if Type4 BAD =True;
602. if $r \in Rng(D_s)$ return $D_s^{-1}[r]$ else $x = \pi^{-1}(r)$;
603. if Type1-c BAD =True;
604. return x;

$\underline{\pi^{-1}(r)}$

501. if $r \notin Rng(D_s)$ $D_s^{-1}[r] \xleftarrow{\$} \{0,1\}^{2n} \setminus Dom(D_s)$;
502. return $D_s^{-1}[r]$;

(b) Game G_1

$\underline{\mathsf{RO}(M)}$

001. if $M \in Dom(D_l)$ return $D_l[M][0, n-1]$;
002. $h \xleftarrow{\$} \{0,1\}^{2n}$; $D_l[M] = h$;
003. return $h[0, n-1]$;

$\underline{\mathsf{MessageRecon}(x, T_s)}$

201. $x \xrightarrow{parse} yy'$;
202. if $\mathsf{FindNode}(y) = 0$ return $\mathcal{M} = \emptyset$;
203. $\mathsf{FindBranch}(y) = \mathcal{M}'$;
204. $\mathcal{M} = \{\mathsf{dePad}(Xz) \mid Xz' \in \mathcal{M}', z = z' \oplus y'\}$;
205. return $\mathcal{M}$;

$\underline{\mathsf{S}(x)}$

101. $r \xleftarrow{\$} \{0,1\}^{2n}$;
102. $\mathcal{M} = \mathsf{MessageRecon}(x, T_s)$;
103. if $|\mathcal{M}| = 1 \wedge M \notin Dom(D_l)$ $D_l[M] = r \oplus (z||0)$;
104. if $|\mathcal{M}| = 1 \wedge M \in Dom(D_l)$ $r = D_l[M] \oplus (z||0)$;
105. $D_s[x] = r$;
106. return r; $\mathsf{FullGraph}(D_s)$;

$\underline{\mathsf{S}^{-1}(r)}$

301. if $r \notin Rng(D_s)$ $D_s^{-1}[r] \xleftarrow{\$} \{0,1\}^{2n} \setminus Dom(D_s)$;
302. return $D_s^{-1}[r]$;

(c) Game(RO, S, S^{-1})

Figure 5: The Games Game(JH, π, π^{-1}), G_1, and Game(RO, S, S^{-1})

Description of Game(RO, S, S^{-1}) (Figure 5(c)). In this game, the pair of s-oracles $\mathsf{S}/\mathsf{S}^{-1}$ is the simulator pair for the indifferentiability framework of JH. The l-oracle of this game implements the random oracle RO. Construction of an effective simulator pair is perhaps the most important part in the analysis of the indifferentiability security for a hash mode of operation. The purpose of the simulator pair $\mathsf{S}/\mathsf{S}^{-1}$ is two-fold: (1) to output values which are indistinguishable from the output of the ideal permutation π/π^{-1}, and (2) to respond in such a way that $JH^{\pi}(M)$ and $\mathsf{RO}(M)$ are identically distributed. It will easily follow that as long as the simulator pair $\mathsf{S}/\mathsf{S}^{-1}$ is able to output values satisfying the above conditions, no adversary can distinguish between Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}).

Our design strategy for $\mathsf{S}/\mathsf{S}^{-1}$ is fairly intuitive and simple. However, before jumping into a detailed description, we define the data structures and the subroutines. The game uses three global data structures: the table D_s, which stores all s-queries and their responses, the table D_l, which stores all l-queries and their responses, and finally a specially designed directed graph T_s.

- Subroutine FullGraph(): This subroutine updates the directed graph T_s using the elements stored in D_s. The graph T_s is built in such a way that each path originating from the root-node (IV, IV') represents the execution of JH$(\cdot)$ on a prefix of some message. For instance, suppose $M \xrightarrow{pad} m_1 m_2 M'$. Then the path $IVIV' \xrightarrow{m_1} y_1 y_1' \xrightarrow{m_2} y_2 y_2'$ represents the first two-block execution of JH(M) where, $y_1 y_1' = \pi(IV||IV \oplus m_1) \oplus m_1||0$ and $y_2 y_2' = \pi(y_1||y_1' \oplus m_2) \oplus m_2||0$. See Figure 6 for a pictorial description. Additionally and more importantly, the graph T_s contains all possible paths derived from the elements in D_s; hence the name FullGraph.

- Subroutine MessageRecon(x, T_s): The purpose of this subroutine is to reconstruct all messages M, such that JH$^{\pi}(M)$ can be computed using the elements in D_s, and $\pi(x)$. As the final input to π in JH$^{\pi}(M)$ is x, we see JH$^{\pi}(M) = \pi(x)[0, n-1] \oplus m_k$, where m_k is the final block of the reconstructed message M.

  In order to find the messages, the subroutine FindNode$(y = x[0, n-1])$ first checks whether there exists a node in the graph T_s with left-coordinate y (line 202). If present, then the subroutine FindBranch(y) collects all paths from the root node (IV, IV') to these nodes of the form yz', and returns a set $\mathcal{M}$. The set $\mathcal{M}$ contains the sequences of arrows on the paths found, each concatenated with $z = z' \oplus x[n, 2n-1]$ (lines 203 and 204). If there are no nodes with left coordinate y, then the empty set is returned (line 202).

On a new forward s-query x, the simulator S assigns a uniformly sampled $2n$-bit value to r (line 101). The subroutine MessageRecon(x, T_s) is then invoked, returning a set of messages $\mathcal{M}$ (line 102). If the size of $\mathcal{M}$ is 1 and M (such that $M \xrightarrow{pad} Xz \in \mathcal{M}$) is not an old l-query, then $D_l[M]$ is assigned the value of x (line 103). If M is an old l-query, then r is assigned the value of $D_l[M]$ (line 104). In lines 105 and 106 the table D_s is updated and the value of r is returned. Finally, the graph T_s is updated. We will see very shortly that the worst-case running time of S on the i^{th} query is $\mathcal{O}(i^4)$.

On a new reverse s-query r, S^{-1} returns a uniformly sampled $2n$-bit value chosen from the set $\{0,1\}^{2n} \setminus Dom(D_s)$ (lines 301 and 302).

For a new l-query M, the random oracle RO first checks whether M has already been reconstructed using forward s-queries. In such a case, M already belongs to $Dom(D_l)$ and RO returns the least-significant n bits of $D_l[M]$, i.e. $D_l[M][0, n-1]$ (line 001). Otherwise, the value for $D_l[M]$ is assigned a uniformly sampled $2n$-bit value (line 002), and the least-significant n bits are returned (line 003). ∎

Time Complexity of the Simulator S Since there are i queries after i rounds, the maximum number of nodes in T_s is i^2. Therefore, to construct the fullgraph at ith round, the amount of time required is $\mathcal{O}(i^4)$. Now if the adversary submits σ queries, then the time complexity is $\mathcal{O}(\sigma^5)$. Since the fullgraph construction dominates the simulator time, the simulator time complexity is also $\mathcal{O}(\sigma^5)$.

Description of G_1 (Fig. 5(b)). The description of the game G_1 might look a bit artificial in the sense

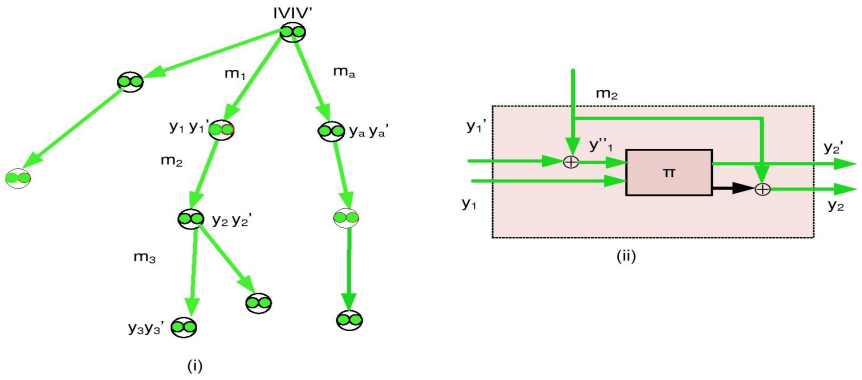


Figure 6: All wires are n bits. (i) The directed graph T_s is updated by the subroutines FullGraph or PartialGraph (see Sect. 3). Example: The edge (y_1y_1', m_2, y_2y_2') is composed of the head node y_1y_1', the arrow m_2, and the tail node y_2y_2'. (ii) Generation of the edge (y_1y_1', m_2, y_2y_2') of T_s using π; the shaded rectangle is viewed as the compression function of JH with $y_1y_1'm_2$ being the input and y_2y_2' as the output.

that it was constructed as a hybridization of the games Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}). The purpose of this game is to be a transition point from Game(JH, π, π^{-1}) to Game(RO, S, S^{-1}) so that their difference in execution can be easily examined. As in Game(RO, S, S^{-1}), the global variables are the tables D_s and D_l, and the graph T_s.

In our initial description of this game, we omit the statements where the variable BAD is set (lines 000, 100, 102, 405, 601, and 603), since they do not impact the output and the global data structures. The variable BAD is set when certain events occur in the global data structures. Those events will be discussed, when we compute $\Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]$ in Section 5. The subroutines in this game are similar to the ones for Game(RO, S, S^{-1}):

- SUBROUTINE PartialGraph(x, r): Like FullGraph, this subroutine also builds the graph T_s in such a way that each directed path originating from the root-node (IV, IV') represents the execution of JH$(\cdot)$ on a prefix of some message (depicted in Figure 6). However, there are important differences. Rather than building all possible paths using the new pair (x, r) and the old pairs in D_s, PartialGraph augments T_s in at most one phase; hence its name.

  PartialGraph invokes the subroutine CreateCoset$(y_c = x[0, n-1])$, which returns a set Coset containing all nodes in T_s, whose left-coordinate is y_c (lines 401 and 402). The size of Coset determines the number of nodes to be added to T_s in the the current iteration. Using the members of Coset and the new pair (x, r), new edges are constructed, stored in EdgeNew, and added to T_s using the subroutine AddEdge.

- SUBROUTINE MessageRecon(x, T_s'): This subroutine is virtually the same as described in game Game(RO, S, S^{-1}). The only difference is that instead of the graph T_s, the second argument to this subroutine is T_s', which is the maximally connected subgraph of T_s with root-node (IV, IV'), generated by a part of the table D_s which contains only the forward s-queries and their responses. Note that in addition to forward s-query/responses, D_s may also contains reverse s-query/responses, as well as π-query/response pairs generated during the execution of l-queries. This is a significant difference between this game and Game(RO, S, S^{-1}), which does not have any π-queries arising from the processing of an l-query. This difference will be quantified in Section 5.

Given a forward s-query x, if $x \in Dom(D_s)$ then $D_s[x]$ is returned by S1. Otherwise, $\pi(x)$ is computed (line 101). We see the ideal permutation π is implemented through lazy sampling (lines 301 and 302). The subroutine MessageRecon is called with input (x, T_s'), which returns a set of reconstructed messages $\mathcal{M}$ (line 103). If the size of $\mathcal{M}$ is 1, and if the M is not a previous l-query (such that $M \stackrel{pad}{\rightarrow} Xz \in \mathcal{M}$), then $D_l[M]$ is assigned the value of $\pi(x) \oplus z||0$ (line 104). After finally returning r (line 105), the subroutine PartialGraph is called on the input (x, r) to update the existing graph T_s.

The reverse s-oracle $\mathsf{S1}^{-1}$ is identical to the oracle S^{-1} of Game(RO, S, S^{-1}).

If an l-query M has already been reconstructed by $\mathsf{S1}$ on some previous forward s-query, then $D_l[M][0, n-1]$ is returned (line 001). Otherwise, (lines 002-012) JH1 mimics JH. It also updates the graph T_s (lines 008 and 013), whenever a new intermediate input is generated (lines 006 and 010). Afterwards, the value of $D_l[M]$ is assigned to be $r \oplus m_k||0$ (line 011). Finally, $D_l[M][0, n-1]$ is returned (line 012). The graph T_s is also updated, if necessary (line 013). ∎

Remark 1 *The difference in how the subroutines* FullGraph *and* PartialGraph *construct the directed graph T_s forms the first non-trivial step towards improving the bound for the indifferentiability security of JH. The* PartialGraph *augments T_s in at most one phase every invocation. It is easy to see that the graph T_s arising from* PartialGraph *is a connected subgraph of the T_s constructed by the* FullGraph*. We will show that if the events in lines 000, 001, 100, 102, 405, 601, and 603 do not occur, then both subroutines build identical graphs. As a consequence, the probability of occurrence of these "BAD" events constitutes a significant fraction of JH's overall indifferentiability bound of $\mathcal{O}(\frac{\sigma^2}{2^n})$. It seems possible that if* PartialGraph *were to augment T_s in more than one phase, the indifferentiability bound could be further improved. However, in this case, the determination of a non-trivial upper-bound on the probability of the BAD events seems to be hard. For more details, see the extended version of this paper.*

With the above description of the games at our disposal, we are well equipped to state and prove an easy, but important, result.

Proposition 3.1 *For any distinguishing adversary $\mathcal{A}$, Game(RO, S, S^{-1})$\equiv G_1$.*

Proof. From the description of $\mathsf{S1}$, and $\mathsf{S1}^{-1}$, we observe that, for all $x \in \{0,1\}^{2n}$, $\mathsf{S1}(x) = \pi(x)$, and $\mathsf{S1}^{-1}(x) = \pi^{-1}(x)$ (lines 101, and 602). Likewise, from the descriptions of JH1 and JH, for all $M \in \{0,1\}^*$, JH1(M) =JH(M). □

The events Type1, Type2, Type3, and Type4 mentioned in Figure 5(b) are still undefined. These events are used to distinguish game G_1 from game Game(RO, S, S^{-1}). We describe them in the next section.

4 Definition of the events BAD_i, and GOOD_i

For the remainder of the paper, by a query we will mean either a message-block in an l-query, or a forward or reverse s-query. With this convention, we see that prior to the i^{th} query, the sum of the s-queries and message-blocks contained in l-queries is thus $i-1$. The purpose of this section is to concretely define the Type1, Type2, Type3 and Type4 events mentioned in lines 405, 603, 001, 100, 102, 000, and 601 of game G_1 (Figure 5(b)). Before doing so, we define a few additional events that can be defined using the Type1, Type2, Type3, and Type4 events.

- Let the symbol BAD_i denote the event when the variable BAD is set (in lines 405, 603, 001, 100, 102, 000, and 601) on the i^{th} query in game G_1. In other words, BAD_i occurs when one or more of Type1, Type2, Type3, or Type4 events occur on the i^{th} query in game G_1.
- The symbol GOOD_i denotes $\neg \vee_{j=1}^{i} \mathsf{BAD}_i$b for all $i > 0$.
- We use the symbol GOOD_0 to denote the event when no queries have been submitted in game G_1.

From a high level, the intuition behind the construction of the BAD_i events in the game G_1 is rather straightforward. We make sure that if BAD_i does not occur, and if GOOD_{i-1} did occur, then the views of G_1 and Game(RO, S, S^{-1}) (up to the i^{th} query) are identically distributed for *any* attacker $\mathcal{A}$. Hence, the BAD_i events will establish the following lemma.

Lemma 4.1 (Computational Paradigm) *For the games G_1 and Game(RO, S, S^{-1}), interacting with an indifferentiability adversary $\mathcal{A}$ limited by σ queries, we have*

$$\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1\ \ - \Pr\big[\mathcal{A}^{Game(RO,\ S,\ S^{-1})} \Rightarrow 1\ \ \leq \Pr\big[\mathcal{A}\ sets\ BAD\ in\ G_1\ \ \leq \sum_{i=0}^{\sigma-1} \Pr\big[BAD_{i+1} \mid GOOD_i\ .$$

The computational paradigm in Lemma 4.1 has been used in most well-known constructions [1, 4, 5, 6, 24]. However, it is worth remembering that the BAD_i events designed in the earlier attempts led to indifferentiability bounds which did not go beyond $n/3$ bits [7]. In our present case, we had to use deeper insight to construct a different set of BAD_i events to extend the bound up to $n/2$ bits. This primarily involved tricks to overcome the interleaving of branches in successive π calls in the JH mode. As mentioned in Remark 1, these events are an important contribution of this paper.

Proof. [Proof Sketch of Lemma 4.1] The second part of the lemma is easy. To prove the first part, we need to show

$$\begin{aligned}\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1 \mid \mathcal{A} \text{ never sets } \mathsf{BAD} \text{ in } G_1\ \ &= \Pr\big[\mathcal{A}^{\mathrm{Game}(\mathsf{RO},\ \mathsf{S},\ \mathsf{S}^{-1})} \Rightarrow 1 \mid \mathcal{A} \text{ never sets } \mathsf{BAD} \text{ in } G_1\ ,\\ \text{or, equivalently, } \Pr\big[\mathcal{A}^{G_1} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-1}\ \ &= \Pr\big[\mathcal{A}^{\mathrm{Game}(\mathsf{RO},\ \mathsf{S},\ \mathsf{S}^{-1})} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-1}\ . \qquad (1)\end{aligned}$$

Let $V_{1,i}$ and $V_{2,i}$ denote the views of the games G_1, and Game(RO, S, S^{-1}), after i queries have been processed. To prove equation (1), it is sufficient to show given $\mathsf{GOOD}_{\sigma-1}$, that $V_{1,\sigma}$ and $V_{2,\sigma}$ are identically distributed. We use induction on the number of rounds.

Induction Hypothesis: Given GOOD_{i-1}, then $V_{1,i}$ and $V_{2,i}$ are identically distributed. Note the hypothesis is clearly true when $i = 1$.

To show: Given GOOD_i, we need that $V_{1,i+1}$ and $V_{2,i+1}$ are identically distributed. The proof is established by observing the BAD_{i+1} events (or, equivalently the Type1, 2, 3 and 4 events occurring in the $(i+1)^{th}$ round of the game G_1) described below. As already mentioned, the Type1, 2, 3 and 4 events were custom-designed so that the lemma would hold. Observe that, given GOOD_{i-1}, and given $V_{1,i} = V_{2,i}$:

1. Any l-queries submitted by $\mathcal{A}$ are identically distributed for the games G_1, and Game(RO, S, S^{-1}). This is easy to see since $V_{1,i} = V_{2,i}$.
2. Any s-queries submitted by $\mathcal{A}$ are identically distributed for the games G_1, and Game(RO, S, S^{-1}). This again follows easily ince $V_{1,i} = V_{2,i}$.
3. The graphs T'_s, and T_s constructed in the games G_1, and Game(RO, S, S^{-1}) are identical. This can be shown by observing the Type1, 2, 3 and 4 events described later in this section.
4. The outputs of the l-queries returned by RO, and JH1 follow the uniform distribution $\mathcal{U}[0, 2^n - 1]$, given that BAD_{i+1} did not occur. This is established using point 3, along with the description of the events in Section 4.
5. The outputs of the forward s-queries returned by S, and S1 also follow the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$, if BAD_{i+1} did not occur. Similarly, this can be shown using point 3, and the description of the events to follow.
6. Finally, the outputs of the reverse s-queries returned by S^{-1}, and $\mathsf{S1}^{-1}$ follow an identical distribution, assuming BAD_{i+1} did not occur. This also will follow from point 3, and the description of the events.

The only remaining part is now to define the Type1, Type2, Type3, and Type4 events of the game G_1, so that the above claims are true. This will finish the proof. For more details than provided in this sketch, we encourage the reader to read the full version of this paper. □

4.1 Type1 Event

Type1 events occur in lines 405, and 603 in Figure 5(b). We divide them into three subcases. Suppose $(y_c y'_c, m, yy')$ is a new edge generated from a new query/response (x, r) (lines 401 through 404).

- **Type1-a event** (Figure 7(Type1-a)): This event occurs if y collides with the left-coordinate of a node already in T_s.
- **Type1-b event** (Figure 7(Type1-b)): This event occurs if y collides with the least-significant n bits of a query already in D_s.
- **Type1-c event** (Figure 7(Type1-c)): This event occurs if the least-significant n bits of output of a reverse s-query match with the left-coordinate of a node already in T_s.

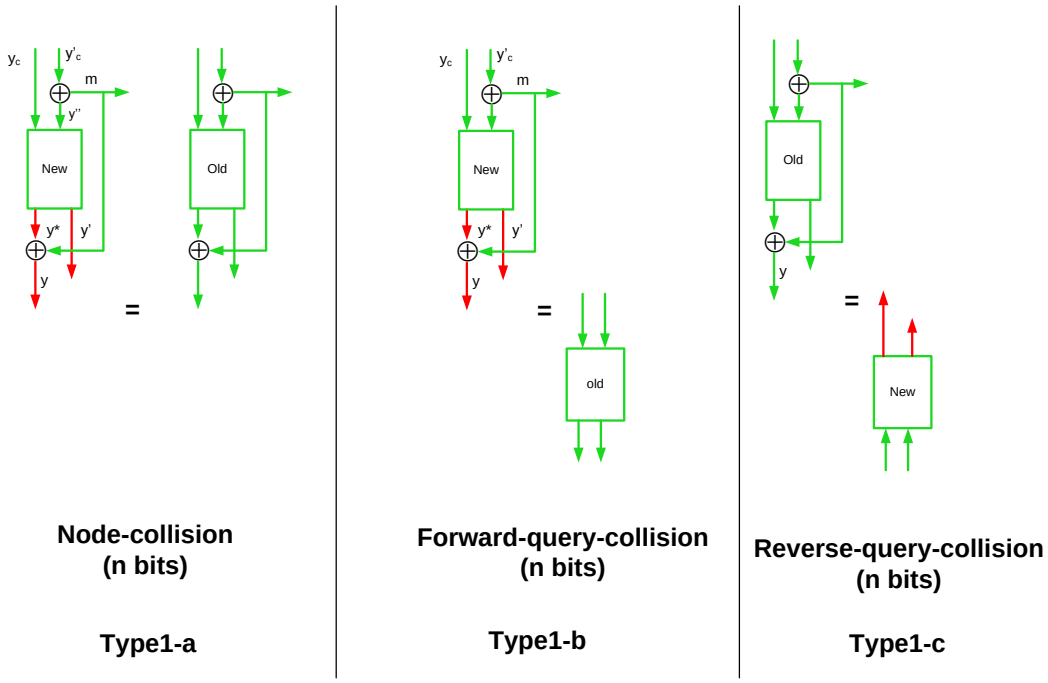
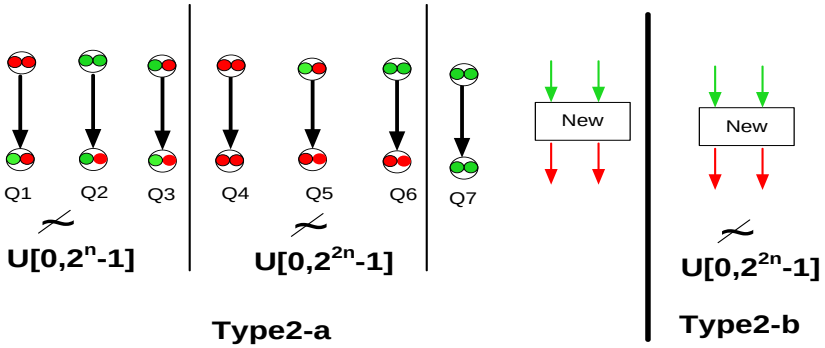


Figure 7: Type1 events when BAD is set in lines 405 and 603 of game G_1 (Figure 5(b)). All arrows are n bits. A red arrow denotes new n bits of output from the ideal permutation π/π^{-1}. The symbol "=" denotes n-bit equality.

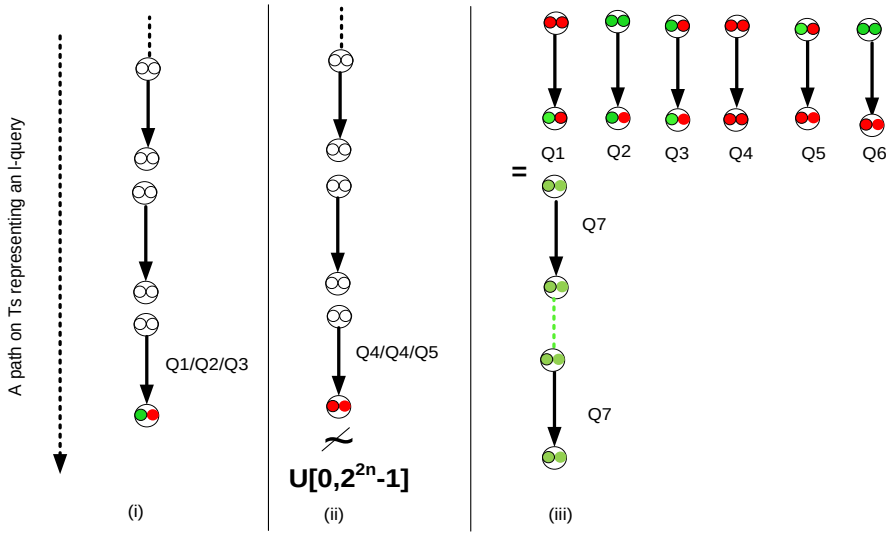
4.2 Type2 Event

This event is mentioned in lines 100 and 102 of game G_1. In order to define Type2 events, we need to classify all old query-response pairs to the oracles π/π^{-1} stored in D_s, according to their known and unknown parts. The known part of a query-response pair is the part which is present in the view of the game G_1, while the unknown part is not present. We note that there are seven types of such pairs. The first six types are generated due to the intermediate π calls made during l-queries. The type Q7 queries are the old s-queries. See Figure 8(a).

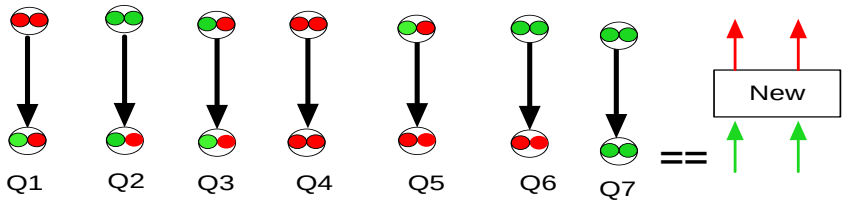
A Type2-a event occurs, if the output of a forward s-query is one of the types Q1, Q2, Q3, Q4, Q5, and Q6, and the output can be distinguished from the uniform distribution. The Type2-b event occurs, if the output of a new forward s-query can be distinguished from the uniform distribution.



(a) Seven types of input-output pairs for π/π^{-1}-query; Type2 events for which BAD is set in lines 100 and 102 of game G_1.



(b) Type3 events for which BAD is set in line 000 of game G_1.



(c) Type4 events for which BAD is set in line 601 of game G_1.

Figure 8: Pictorial description of Type2, Type3, and Type4 events of the game G_1 (Figure 5(b)). Green circles and green arrows denote n bits present in the view of the game. Red circles denote n bits not present in the view. A red arrow denotes new n bits of output from the ideal permutation π/π^{-1}. The symbols "=", and "==" denote n-bit and $2n$-bit equality respectively.

4.3 Type3 Event

This event is mentioned in line 000 of game G_1. A Type3 event occurs when an l-query M is submitted in the game G_1, and $\mathrm{JH}^{\pi}(M)$ can be computed from the entries already stored in D_s. We divide this case into three subcases, according to the final π-query. The first subase is when the final π-query is of type Q1, or Q2, or Q3. A simple observation shows that this case cannot happen, since this would imply a left-coordinate-collision in T_s, forbidden by the occurrence of GOOD_i. The second subcase is if the final π-query is of type Q4, or Q5, or Q6. The Type3 event occurs, if an adversary distinguishes the $2n$ bits of final output from the uniform distribution $\mathcal{U}[2^{2n}-1]$. Finally, the third subcase is when the final π-query is of type Q7. In this situation, the Type3 event occurs if $\mathrm{JH}^{\pi}(M)$ can be computed on an existing path of the graph T_s, and if the path has an π-query/response whose type is one of Q1, Q2, Q3, Q4, Q5 and Q6. See Figure 8(b) for a graphical description.

4.4 Type4 Event

The final event is mentioned in line 601 of game G_1. A Type4 event occurs, if the input of a fresh reverse s-query matches with a query of type Q1, Q2, Q3, Q4, Q5, or Q6. See Figure 8(c).

5 Estimation of $\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1\ - \Pr\big[\mathcal{A}^{\mathsf{RO},\mathsf{S}} \Rightarrow 1$

We individually compute the probabilities of each of the events described in Section 4. We need the help of the following lemma to provide a rigorous analysis for the upper-bounds we compute in this section.

Lemma 5.1 (Correction Factor) *If the advantage of an indifferentiable adversary $\mathcal{A}$ for the games G_1 and Game(RO, S, S^{-1}), limited by σ queries, is bounded by ε, which is a negligible function in the security parameter n, then*

$$\Pr\big[\mathsf{GOOD}_i\ \geq \frac{1}{C}$$

for some constant $C > 0$, for all $0 \leq i \leq \sigma - 1$, and for all $n > 0$.

Proof. Since $\varepsilon < 1$ for all $n > 0$, $\Pr\big[\mathcal{A}$ sets BAD in $G_1\ \leq \varepsilon \leq 1 - \frac{1}{C}$ for some constant $C > 0$, and for all $n > 0$. Also note that $\Pr\big[\mathsf{GOOD}_i\ $ is a decreasing function in i. Hence the result. □

The Type1-a event guarantees that that if T_s is GOOD_i, then it has $\mathcal{O}(i)$ nodes, thus from Figure 7, we obtain,

$$\begin{aligned}\Pr\Big[\mathrm{Type1}_{i+1} \mid \mathsf{GOOD}_i\Big] &\leq 3i/2^n, \\ &= \mathcal{O}\Big(\frac{i}{2^n}\ . \end{aligned} \tag{2}$$

Using the definition of Type2 events in Section 4, and their representation in Figure 8(a), it is easy to deduce:

$$\Pr\big[\mathrm{Type2}_{i+1} \mid \mathsf{GOOD}_i\ = \mathcal{O}\Big(\frac{i}{2^n}\ .$$

Similarly, with the definition of Type3 events in Section 4, and Figure 8(b), we find:

$$\Pr\big[\mathrm{Type3}_{i+1} \mid \mathsf{GOOD}_i\ = \mathcal{O}\Big(\frac{1}{2^n}\ .$$

Finally, using the definition of Type4 events, it follows that:

$$\Pr\big[\mathrm{Type4}_{i+1} \mid \mathsf{GOOD}_i\ = \mathcal{O}\Big(\frac{i}{2^n}\ .$$

We conclude with the following inequality which holds for $0 \leq i \leq \sigma - 1$. By the upper-bounds computed above, we obtain

$$\begin{aligned} \Pr\big[\mathsf{BAD}_{i+1} \mid \mathsf{GOOD}_i \big] &\leq \Pr\big[\mathrm{Type1}_{i+1} \mid \mathsf{GOOD}_i \big] + \Pr\big[\mathrm{Type2}_{i+1} \mid \mathsf{GOOD}_i \big] \\ &\quad + \Pr\big[\mathrm{Type3}_{i+1} \mid \mathsf{GOOD}_i \big] + \Pr\big[\mathrm{Type4}_{i+1} \mid \mathsf{GOOD}_i \big] \\ &= \mathcal{O}\Big(\frac{i}{2^n}\Big) . \end{aligned} \tag{3}$$

Therefore, by Lemma 4.1, for all $\mathcal{A}$,

$$\begin{aligned} \Pr\big[\mathcal{A}^{G_1} \Rightarrow 1\big] - \Pr\big[\mathcal{A}^{\mathrm{Game}(\mathsf{RO},\, \mathsf{S},\, \mathsf{S}^{-1})} \Rightarrow 1\big] &\leq \sum_{i=0}^{\sigma-1} \Pr\big[\mathsf{BAD}_{i+1} \mid \mathsf{GOOD}_i \big] \\ &\leq \sum_{i=0}^{\sigma-1} \mathcal{O}\Big(\frac{i}{2^n}\Big) \\ &= \mathcal{O}\Big(\frac{\sigma^2}{2^n}\Big) . \end{aligned} \tag{4}$$

This last equation (4) yields the bound of $n/2$.

6 Experimental Results

We performed a series of experiments verifying our theoretical framework. Our simple C implementation of the game G_1 simulated the ideal permutation, π, with randomness supplied by $\mathsf{rand}()$, by maintaining a database of input/output pairs, assuring that π is a permutation. The experiments were performed allowing varying proportions of reverse queries to determine the optimal adversarial strategy.

For each of these experiments, we collected data providing accurate estimates for the values $\Pr\big[\mathrm{Type1}_{i+1} \mid \mathsf{GOOD}_i$ described in Section 4. Compiling these data we computed the relative percentage of Type1 events when the proportion, R, of reverse queries were 0, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$, and 1. The results are provided in Table 6.

		n						
R	Event	6	7	8	9	10	11	12
$\frac{1}{4}$	Type1-a	62.1%	61.7%	61.0%	60.8%	60.4%	60.3%	60.4%
	Type1-b	17.3%	18.0%	18.5%	19.1%	19.4%	19.4%	19.5%
	Type1-c	20.6%	20.4%	20.4%	20.3%	20.2%	20.2%	20.0%
$\frac{1}{2}$	Type1-a	35.6%	35.0%	34.7%	34.0%	33.9%	34.1%	33.7%
	Type1-b	28.9%	30.1%	30.6%	31.9%	32.1%	32.3%	32.8%
	Type1-c	35.6%	34.9%	34.7%	34.1%	34.0%	33.6%	33.5%
$\frac{3}{4}$	Type1-a	16.3%	15.7%	15.3%	15.1%	14.8%	14.5%	14.5%
	Type1-b	35.2%	37.6%	38.8%	40.3%	40.8%	41.6%	42.2%
	Type1-c	48.4%	46.8%	45.9%	44.7%	44.4%	43.9%	43.3%

Table 2: The relative percentage of Type1 bad events for various values of n and R, the proportion of reverse queries allowed. Trivially, for $R = 0$, all bad events are Type1-a, and for $r = 1$, all bad events are Type1-c. The values were computed using 100000 simulations.

In addition to these event probabilities, we calculated security bounds for several values of n and R. The computation was achieved by randomly generating a large number of graphs T_s, and determining the number of queries, σ, required to cause $\sum_{i=0}^{\sigma-1} \Pr\big[\mathrm{Type1}_{i+1} \mid \mathsf{GOOD}_i \big] \geq 0.5$.

We did not consider the Type2 and Type3 events, since, intuitively, their probabilities are dominated by those of the Type1 events, for any efficient adversary. However, a theoretical proof of the validity of this

assumption is still an open problem. We found that choosing the values at which to place the first query uniformly at random from among all possible maximal left-cosets was the most advantageous strategy for an adversary.

The results of the experiments following this method are summarized in Figure 9. The data supports the theoretically obtained bound of $\sigma = \Omega(2^{n/2})$ (see Equation 4). Some of the values in the graph are slightly lower than 1/2, due to the effect of constants. We expect the data to asymptotically approach 1/2.

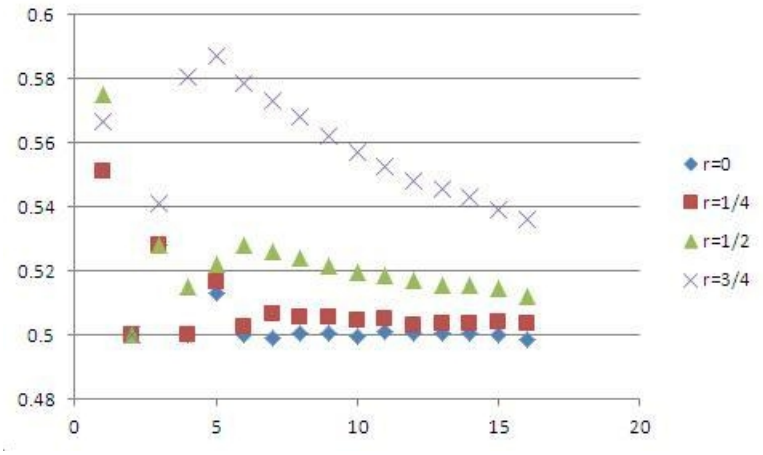


Figure 9: Plot of experimental data of value of n versus the normalized logarithm of σ, $\log_2(\sigma)/n$, for the game G_1 with various values of r, the proportion of reverse queries allowed.

The data indicates that the optimal adversarial strategy for game G_1 is to disallow reverse queries. For each fixed $R < 1$, however, we observe that the data asymptotically approaches 1/2. Although it is the case that for $R = 1$, σ has an expected value of 2^{n-1}, we conjecture that for any fixed value of $R < 1$, $\sigma = \Theta(2^{n/2})$.

7 Conclusion

In this paper we improved the indifferentiability security bound of the JH hash mode of operation from $n/3$ bits to $n/2$ bits. We are currently experimenting to see if the bound could be further improved, by employing various techniques, including the allowance of n-bit collision a certain number of times or the utilization of higher phase rounds in which previous queries may allow T_s to grow more rapidly. In addition, our very straight forward proof technique, which is strikingly different from most approaches in the literature, may be of independent interest.

References

[1] Elena Andreeva, Bart Mennink, and Bart Preneel. On the Indifferentiability of the Grøstl Hash Function. In Juan A. Garay and Roberto De Prisco, editors, *SCN*, volume 6280 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2010. (Cited on pages 4 and 10.)

[2] Elena Andreeva, Bart Mennink, and Bart Preneel. Security Reductions of the Second Round SHA-3 Candidates. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC*, volume 6531 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2010. (Cited on page 2.)

[3] Elena Andreeva, Bart Mennink, and Bart Preneel. The Parazoa Family: Generalizing the Sponge Hash Functions. *IACR Cryptology ePrint Archive*, 2011:28, 2011. (Cited on page 1.)

[4] Mihir Bellare and Thomas Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2006. (Cited on pages 1, 4 and 10.)

[5] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008. (Cited on pages 1 and 10.)

[6] Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Indifferentiability characterization of hash functions and optimal bounds of popular domain extensions. In Bimal K. Roy and Nicolas Sendrier, editors, *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 199–218. Springer, 2009. (Cited on page 10.)

[7] Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Security Analysis of the Mode of JH Hash Function. In Seokhie Hong and Tetsu Iwata, editors, *FSE*, volume 6147 of *Lecture Notes in Computer Science*, pages 168–191. Springer, 2010. (Cited on pages 1, 2 and 10.)

[8] Eli Biham and Orr Dunkelman. A framework for iterative hash functions – HAIFA. Second NIST Cryptographic Hash Workshop, 2006, 2006. (Cited on page 1.)

[9] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990. (Cited on page 16.)

[10] E. Bresson, A. Canteaut, B. Chevallier-Mames, C .Clavier, T. Fuhr, A. Gouget, T. Icart, J.-F. Misarsky, M. Naya-Plasencia, P. Paillier, T. Pornin, J.-R. Reinhard, C. Thuillet, and M. Videau. SHABAL. The 1st SHA-3 Candidate Conference. (Cited on page 1.)

[11] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005. (Cited on pages 2 and 3.)

[12] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [9], pages 416–427. (Cited on page 1.)

[13] Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Some Observations on Indifferentiability. In Ron Steinfeld and Philip Hawkes, editors, *ACISP*, volume 6168 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2010. (Cited on page 2.)

[14] P. Gauravaram, L. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schlaffer, and S. Thomsen. Groestl - a SHA-3 candidate. The 1st SHA-3 Candidate Conference. (Cited on page 1.)

[15] Shoichi Hirose, Je Hong Park, and Aaram Yun. A Simple Variant of the Merkle-Damgård Scheme with a Permutation. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2007. (Cited on page 1.)

[16] Antoine Joux. Multicollisions in Iterated Hash Functions: Application to Cascaded Constructions. In *CRYPTO 2004*, pages 306–316, 2004. (Cited on page 1.)

[17] John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006. (Cited on page 1.)

[18] John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005. (Cited on page 1.)

[19] Jooyoung Lee and Deukjo Hong. Collision resistance of the jh hash function. Cryptology ePrint Archive, Report 2011/019, 2011. `http://eprint.iacr.org/`. (Cited on page 2.)

[20] Stefan Lucks. A failure-friendly design principle for hash functions. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005. (Cited on page 1.)

[21] 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. (Cited on pages 1, 3 and 4.)

[22] Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [9], pages 428–446. (Cited on page 1.)

[23] Dustin Moody, Souradyuti Paul, and Daniel Smith-Tone. Indifferentiability security of the fast widepipe hash: Breaking the birthday barrier. Cryptology ePrint Archive, Report 2011/630, 2011. `http://eprint.iacr.org/`. (Cited on page 1.)

[24] Mridul Nandi and Souradyuti Paul. Speeding up the wide-pipe: Secure and fast hashing. In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 144–162. Springer, 2010. (Cited on page 10.)

[25] NIST. Secure hash standard. In *Federal Information Processing Standard, FIPS-180*, 1993. (Cited on page 1.)

[26] NIST. Secure hash standard. In *Federal Information Processing Standard, FIPS 180-1*, April 1995. (Cited on page 1.)

[27] Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with Composition: Limitations of the Indifferentiability Framework. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer, 2011. (Cited on page 2.)

[28] R. Rivest. The MD4 message-digest algorithm. In A. J. Menezes and S. Vanstone, editors, *CRYPTO*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer, 1990. (Cited on page 1.)

[29] R. Rivest. The MD5 message-digest algorithm. In *IETF RFC 1321*, 1992. (Cited on page 1.)

[30] Hongjun Wu. The JH Hash Function. The 1st SHA-3 Candidate Conference. (Cited on page 1.)