Des. Codes Cryptogr. (2016) 79:237–259
DOI 10.1007/s10623-015-0047-9

Improved indifferentiability security bound for the JH mode

Dustin Moody · Souradyuti Paul · Daniel Smith-Tone

Received: 22 May 2014 / Revised: 30 January 2015 / Accepted: 1 February 2015 /
Published online: 19 February 2015

Abstract Indifferentiability security of a hash mode of operation guarantees the mode's resistance against *all* generic attacks. It is also useful to establish the security of protocols that use hash functions as random functions. The JH hash function was one of the five finalists in the National Institute of Standards and Technology SHA-3 hash function competition. Despite several years of analysis, the indifferentiability security of the JH mode has remained remarkably low, only at $n/3$ bits, while the two finalist modes Keccak and Grøstl offer a security guarantee of $n/2$ bits. Note all these three modes operate with n-bit digest and $2n$-bit permutations. In this paper, we improve the indifferentiability security bound for the JH mode to $n/2$ bits (e.g. from approximately 171 to 256 bits when $n = 512$). To put this into perspective, our result guarantees the absence of (non-trivial) attacks on both the JH-256 and JH-512 hash functions with time less than approximately 2^{256} computations of the underlying 1024-bit permutation, under the assumption that the underlying permutations can be modeled as an ideal permutation. Our bounds are optimal for JH-256, and the best known for JH-512. We obtain this improved bound by establishing an isomorphism of certain query-response graphs through a careful design of the simulators and bad events. Our experimental data strongly supports the theoretically obtained results.

Keywords Indifferentiability · Security · Hash functions · JH mode of operation

Communicated by L. R. Knudsen.

D. Moody (✉) · D. Smith-Tone
National Institute of Standards and Technology, Gaithersburg, MD 20877, USA
e-mail: dustin.moody@nist.gov

S. Paul
Computer Science and Engineering, Indian Institute of Technology (IIT) Gandhinagar, Chandkheda, Ahmedabad 382424, India
e-mail: souradyutip@iitgn.ac.in

Present Address:
D. Smith-Tone
Department of Mathematics, University of Louisville, Louisville, KY 40292, USA

Mathematics Subject Classification 94A60 · 68W40 · 68Q25

1 Introduction

1.1 Generic attacks

In a generic attack, an adversary attempts to break a property of the target crypto-algorithm assuming that one or more of its smaller components are ideal objects, such as random oracles, ideal permutations, or ideal ciphers. For example, suppose that the target crypto-algorithm is a hash function $H : \{0, 1\}* \rightarrow \{0, 1\}^n$. Assume that for a given input $M \in \{0, 1\}^*$, H invokes an ideal object, say a random oracle $\mathsf{ro} : \{0, 1\}^m \rightarrow \{0, 1\}^n$, one or multiple times, to compute $H(M)$. Informally, a generic attack breaks a property of the hash function H utilizing less resources than would be required to break the same property of the big random oracle $\mathsf{RO} : \{0, 1\}^* \rightarrow \{0, 1\}^n$.

Generic attacks against hash functions are plentiful in the literature. See, for example, Joux's multi-collision attack [19], the Kelsey–Schneier expandable-message second pre-image attack[1] [21], and the Kelsey–Kohno herding attack [20], all on the popular Merkle–Damgård hash mode. Generic attacks have also been reported on hash modes other than the plain Merkle–Damgård mode. A few of these are the 2nd pre-image attacks on the dithered variants of the Merkle–Damgård construction [1], a pre-image attack on the JH mode [8], pre-image, second pre-image, and multi-collision attacks on the Sponge construction when the state-size is small [7], collision attacks on some concatenated hash functions [19], second pre-image, multi-collision and herding attacks on some hash functions based on checksums [13,16], multi-collisions in iterated concatenated and expanded hash functions [18], and multi-collisions on some generalized sequential hash functions [24]. See also [14,15,17], which analyze generic attacks on randomized hashing (a variant of Merkle–Damgård).

In each of the above examples, a common assumption was that the underlying basic primitive of the hash function is an ideal object. Therefore, all of these attacks fit the definition of a generic attack. Generic attacks have changed the outlook on the security of a cryptographic hash function over the last few years. One naturally asks how to design a hash mode secure against *all* generic attacks.

1.2 Indifferentiability security

The indifferentiability security framework was introduced by Maurer et al. [23] in 2004, and was first applied to analyze hash modes of operation by Coron et al. [10] in 2005. A hash mode proven secure in this framework is able to resist *all* generic attacks. More technically, the indifferentiability framework measures the extent to which a hash function behaves as a random oracle under the assumption that the underlying small compression function is an ideal object. The class of indifferentiability attacks includes more attacks [4,8,9] than just useful generic attacks as above. Thus in some sense, an indifferentiable hash function can be viewed as eliminating potential future attacks. We note the security of many cryptographic protocols rely on the indifferentiability security of the underlying hash functions that the protocols use as random oracles. In such a case, security of the hash functions against selected specialized attacks—such as collision, pre-image attacks and second pre-image attacks—are inadequate

[1] This generic attack is a generalized version of the second pre-image attack of Dean [11] which works on Merkle–Damgård hash functions based on compression functions with fixed points.

Table 1 The resistance of the JH mode against several attacks

Mode of operation	Message block-length	Permutation size	First preimage	Second preimage	Collision resistance	Indiff. (old)	Indiff. (new)
JH-n	n	$2n$	$n/2$ [3]	$n/2$ [3]	$n/2$* [22]	$n/3$ [8]	$n/2$
JH-512	512	1024	256	256	256*	171	256
JH-256	512	1024	256*	256*	128*	171	256*

Each number is in bits. The asterisk indicates the optimality of the bound

to guarantee the security of the overlying protocol. The notion of indifferentiability security has also been applied to compression functions. See [5] for example. Some limitations of the indifferentiability framework have recently been discovered in [12] and [26]: the first paper shows how some hash modes can be attacked in the indifferentiability setting, even if they are "practically secure"; the second paper points out how a certain protocol allows for an attack by a multi-stage adversary when its underlying random oracle is replaced by an indifferentiable hash function. These limitations do not apply to the indifferentiability security of compression functions (i.e., [5]). Still, indifferentiability security has gradually become a de-facto requirement for the adoption of a hash mode as a standard because it guarantees security for hash function modes of operation against generic attacks.

1.3 Previous analysis of the JH mode

The JH hash function was one of the five finalist algorithms in the NIST SHA-3 hash function competition [25]. The hash function uses an iterative mode which is novel in the sense that it is based on a permutation [28]. Several popular hash functions—such as SHA-1 and SHA-2—are constructed instead using a block cipher. Since its publication in 2007, the JH mode of operation has undergone an extensive security analysis. The first published analysis of the JH mode was done by Bhattacharyya, Mandal and Nandi, who showed that the indifferentiability security of the basic version of the JH mode up to $n/3$ bits [8];[2] they have also shown a preimage attack on the JH-512 mode with approximately 2^{507} calls to the underlying permutation. A year later, in [22], it was shown that the JH mode achieves the optimal collision resistance of up to $n/2$ bits. Recently Andreeva, Mennink and Preneel have improved the first and second pre-image resistance of the JH mode from $n/3$ to $n/2$ bits [3]. However, the improvement of the indifferentiability security of the JH mode beyond $n/3$ bits has remained elusive. Table 1 gives an overview of the main results on the JH mode.

1.4 Our contribution

The usage of an ideal permutation, instead of a random oracle, in the JH mode allows the adversary to use reverse queries in addition to forward queries. One of the main obstacles for an improved indifferentiability security analysis of the JH mode is how to handle these reverse queries. This additional privilege of the adversary makes challenging the construction of an efficient simulator able to withstand adversaries using up to (approximately) $2^{n/2}$ queries. It is important to note that these adversaries (working in the indifferentiability framework) are *distinguishing* adversaries telling apart the pair of the JH mode and the underlying permutation from the pair of a random oracle and a simulator. Since the framework involves two pairs of algorithms, the security guarantee obtained in this framework is at least as much as—

[2] The basic version uses a $2n$-bit permutation and n-bit digest. The chopped versions use a smaller digest.

and most likely even better than—the guarantee obtained in a framework that distinguishes output of one algorithm (e.g. JH mode) from that of the other (e.g random oracle) [23]. Another major challenge, which turns out to be quite hard, is to estimate the probability of the events when a current query submitted by an arbitrary adversary matches an old but unknown query. A somewhat easier task is to show that the probability of a node-collision on the graph constructed by an efficient simulator, is at most $\frac{\sigma^2}{2^n}$, where σ is the total number of submitted queries. We overcome these hurdles by carefully designing a set of bad events. Our construction is such that the absence of the bad events, (1) eliminates the possibility of a reverse query being attached to the simulator graph, (2) allows the graph to grow only linearly in the number of submitted queries, and most importantly (3) ensures the isomorphism of the simulator graphs in two different games. Using this isomorphism result and the linear bound on the number of nodes in the isomorphic graphs, we are able to improve the indifferentiability security bound of JH to $n/2$ bits. Another feature of our work, which may be of independent interest, is that the proof of our main theorem Theorem 1 requires *only* three games. The smaller number of games (in stark comparison with the usual practice of tackling such problems using a sequence of a large number of games) makes third-party verification of the proof a great deal easier and also allows the application of probabilistic tools to find practical security bounds [27] .

Our indifferentiability bound guarantees the absence of generic attacks on the JH hash function (using a $2n$-bit permutation) with work less than $2^{n/2}$. When the digest-size is 256 or 512 bits, the hash mode is resistant to all generic attacks up to (approximately) 2^{256} computations of the underlying 1024-bit permutation. This bound is optimal for JH-256 and the best known for JH-512. Furthermore, we have performed a series of experiments with the JH mode studying the effects of the bad events in our framework. Our experiments verify the theoretically obtained results, and also exhibit optimal adversarial strategies. See Sect. 6 for more on the experiments. We caution the reader that our result on the JH mode says nothing about the security of the underlying 1024-bit permutation, which is assumed to be free from all structural weaknesses throughout the paper.

1.5 Notation and convention

Throughout the paper we let n be a fixed integer. We shall use the little-endian bit-ordering system. The symbol $|\cdot|$ is used for both length of a message and the cardinality of a set. For concatenation of fixed length strings a and b, we use $a||b$, or just ab if the meaning is clear. Let S_X denote the sample space of the discrete random variable X. The relation $A \sim B$ is satisfied if and only if $\Pr\big[A = X\big] = \Pr\big[B = X\big]$ for all $X \in S$, where $S = S_A = S_B$. Let T be an array or a table. Then $Dom(T) = \{i \mid T[i] \neq \perp\}$ and $Rng(T) = \{T[i] \mid T[i] \neq \perp\}$. We write $\mathscr{A}^B \Rightarrow b$ to denote an algorithm $\mathscr{A}$ with oracle access to B outputting b. Finally, let $[c, d]$ be the set of integers between c and d inclusive, and $a[x, y]$ the bit-string between the x-th and y-th bit-positions of a. In algorithm descriptions, '=' is used to denote the assignment operation.

2 Indifferentiability framework for JH

2.1 Description of the JH mode

Suppose $n \geq 1$. Let $\pi : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n}$ be a $2n$-bit ideal permutation used to build the JH hash function $\mathrm{JH}^{\pi} : \{0, 1\}^* \rightarrow \{0, 1\}^n$. A pictorial description of the JH transform is

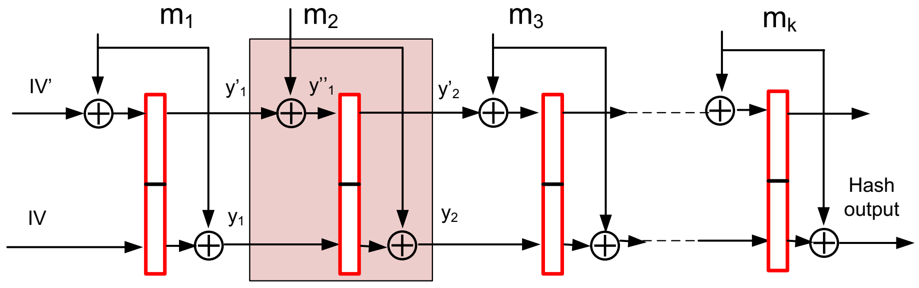

Fig. 1 Diagram for the JH mode. All *rectangles* denote the ideal permutation π on $\{0,1\}^{2n}$. JH takes as input a message $M \in \{0,1\}^*$, and performs the following four steps: $M \overset{pad}{\rightarrow} m_1 m_2 \ldots m_{k-1} m_k$; $y_0 = IV$, $y'_0 = IV'$; $y_i y'_i = \pi\left(y_{i-1} || (y'_{i-1} \oplus m_i)\right) \oplus m_i || 0$ for all $i \in \{1, 2, \ldots k\}$; return y_k

given in Fig. 1. The semantics for the notation $M \overset{pad}{\rightarrow} m_1 \cdots m_{k-1} m_k$ is as follows: Using an injective function $\mathsf{pad} : \{0,1\}^* \rightarrow \cup_{i \geq 1} \{0,1\}^{ni}$, M is mapped into a string $m_1 \cdots m_{k-1} m_k$ such that $k = \left\lceil \frac{|M|}{n} \right\rceil + 1$, $|m_i| = n$ for $1 \leq i \leq k$. The injective function $\mathsf{pad}(\cdot)$ ensures that distinct messages remain distinct after padding. 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 $\mathsf{dePad} \colon \cup_{i \geq 1} \{0,1\}^{in} \rightarrow \{\perp\} \cup \{0,1\}^*$ is defined as follows: $\mathsf{dePad}(\mathsf{pad}(M)) = M$, for all $M \in \{0,1\}^*$, and otherwise $\mathsf{dePad}(\cdot)$ returns a special symbol $\perp$ denoting that the padded message was not generated from a valid message. We note that the padding rules of all practical hash functions have the above properties. For more details, the reader is referred to the original specification written by the JH designer [28].

2.2 Introduction to the indifferentiability framework

We will frequently refer to the use of a random oracle. A random oracle is a function $\mathsf{RO} : X \rightarrow Y$ chosen uniformly at random from the set of all $|Y|^{|X|}$ functions that map $X \rightarrow Y$. In other words, a function $\mathsf{RO} : X \rightarrow Y$ 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.

We now define the indifferentiability security notion which is a slightly modified version of the original definition provided in [10,23].

Indifferentiability security [10] An interactive Turing machine (ITM) T with oracle access to an ideal primitive $\mathscr{F}$ is said to be $(t_{\mathscr{A}}, t_S, q, \varepsilon)$-indifferentiable from an ideal primitive $\mathscr{G}$ if there exists a simulator S such that, for any distinguisher $\mathscr{A}$, the following equation is satisfied:

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

The simulator S is an ITM which has oracle access to $\mathscr{G}$ and runs in time at most t_S. The distinguisher $\mathscr{A}$ – also known as an *indifferentiability* adversary—runs in time at most $t_{\mathscr{A}}$. The number of queries used by $\mathscr{A}$ is at most q. Here ε is a real number in (0, 1).

Suppose, an ideal primitive $\mathscr{G}$ (e.g. a variable-input-length random oracle) is indifferentiable from an algorithm T based on another ideal primitive $\mathscr{F}$ (e.g. a fixed-input-length random oracle). Then any cryptographic system $\mathscr{P}$ based on $\mathscr{G}$ is as secure as $\mathscr{P}$ based on $T^{\mathscr{F}}$ (i.e., $\mathscr{G}$ replaces $T^{\mathscr{F}}$ in $\mathscr{P}$) [23]. For a more detailed explanation, we refer the reader to [23].

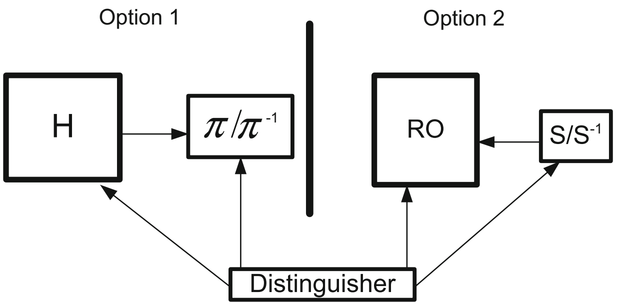

Fig. 2 Indifferentiability framework for a hash function based on an ideal permutation. An *arrow* indicates the direction in which a query is submitted

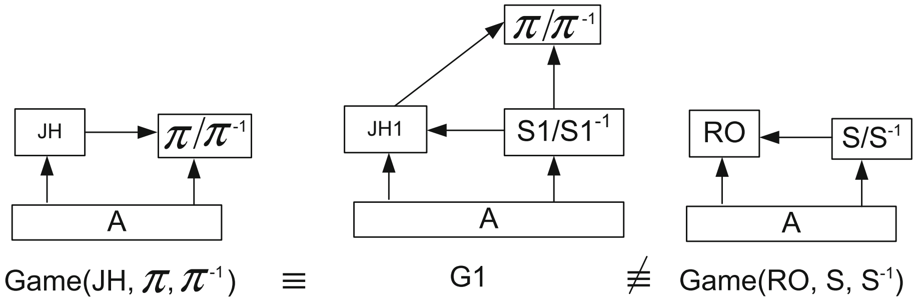

Fig. 3 Schematic diagrams of the security games used in the indifferentiability framework for JH. The *arrows* show the directions in which the queries are submitted

Pictorial description of indifferentiability security framework In Fig. 2, the five algorithms involved in the definition of indifferentiability security are shown: T, $\mathscr{F}$, $\mathscr{G}$ 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 based on the ideal permutation π. In this setting, the definition of indifferentiability addresses the degree to which any computationally bounded adversary is unable to distinguish between Option 1 and Option 2.

2.3 JH indifferentiability

To study the indifferentiability security of the JH mode, we use the ideal permutation π/π^{-1} : $\{0,1\}^{2n} \to \{0,1\}^{2n}$ as the basic primitive of JH. To obtain the indifferentiability security bound, we follow the usual game-playing techniques [2,6]. The schematic diagrams of the two games Option 1 and Option 2 (of Fig. 2) are Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}) as illustrated in Fig. 3. The other game $G1$ is an intermediate step, allowing us to more easily compare pairs of games. The pseudocode for all the games are provided in Sect. 3. One of the major challenges of the indifferentiability security analysis of JH is the construction of a simulator-pair S/S^{-1} being able to withstand attacks by all adversaries limited by a total of $2^{n/2}$ queries to the underlying permutations. The construction techniques for designing such a simulator-pair and their effectiveness are described in detail in Sect. 3.

A necessary part of our analysis is determining equivalences between pairs of games. We define this notion formally.

Equivalence of games 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. Let (x_i, y_i) denote the i-th query and response pair from the game G. The view of the game G after j queries (with respect to the adversary $\mathscr{A}$), is the sequence $\{(x_1, y_1), \ldots, (x_j, y_j)\}$.

Denote the views of the games $G1$ and $G2$ after i queries by V_1^i and V_2^i. The games $G1$ and $G2$ are said to be equivalent (with respect to the adversary $\mathscr{A}$) if and only if $V_1^i \sim V_2^i$ for all $i > 0$. Equivalence between the games $G1$ and $G2$ is denoted by $G_1 \overset{\mathscr{A}}{\equiv} G2$, or simply $G1 \equiv G2$, when the adversary is clear from the context.

3 Description of the security games for JH

In this section, we elaborate on the games Game(JH, π, π^{-1}), $G1$, and Game(RO, S, S^{-1}) that are schematically presented in Fig. 3. The pseudocode for all the games is given in Figs. 4 and 6.

JH, JH1, and RO are mappings from $\{0, 1\}*$ to $\{0, 1\}^n$. S is a mapping from $\{0, 1\}^{2n}$ to $\{0, 1\}^{2n}$. Also, π, π^{-1}, $\mathsf{S1}$, and $\mathsf{S1}^{-1}$ are all permutations on $\{0, 1\}^{2n}$, while S^{-1} is a mapping from $\{0, 1\}^{2n}$ to $\{0, 1\}^{2n} \cup \{\text{“INVALID”}\}$. The mapping S^{-1} returns a special string “INVALID” if it is not behaving like a permutation; more precisely, on input r, S^{-1} returns “INVALID” if there exist at least two distinct images x_1 and x_2 such that $\mathsf{S}(x_1) = \mathsf{S}(x_2) = r$. A query submitted to JH, or JH1, or RO is called an l-query, short for long query. Likewise, a query submitted to π of Game(JH, π, π^{-1}), or to $\mathsf{S1}$ of $G1$, or to S of Game(RO, S, S^{-1}), is called an s-query. A query submitted to π^{-1} of Game(JH, π, π^{-1}), or to $\mathsf{S1}^{-1}$ of $G1$, or to S^{-1} of Game(RO, S, S^{-1}), is called an s^{-1}-query. An s-, s^{-1}-, π-, or π^{-1}-query is also called a short query.

The games will use several global and local variables. The global variables D_l and D_s are two tables used to store query-response pairs: D_l for l-queries and responses, and D_s for s/s^{-1}-queries and responses. The table D_π contains all π/π^{-1}-queries and responses. The tables D_l, D_s and D_π, and all local variables are initialized with $\perp$. The graphs T_π and T_s—built using elements of D_π and D_s—are also global variables which initially contain only a 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 queries.

The queries can also be divided into types according to the time of submission and the location in the tables. The *current* query is the one that is submitted by the adversary at the current time. A *c*urrent query can be of two types: it is an *old* query if already present in the query history; it is a *fresh* query if *not* present in the query history. Table 2 formally defines an old and a fresh query in the various security games.

We assume that the adversary does not submit two identical l-queries, s-queries, or s^{-1} queries. This implies that every current l-query is fresh in all the games; however, every current s-, s^{-1}-, π-, or π^{-1}-query is not necessarily fresh in all games. For example, the current s-query in Game(JH, π, π^{-1}) may accidentally match a query in $Dom(D_\pi)$ that was generated as an intermediate π-query from a previously submitted l-query. Later on, we shall collect these accidents as BAD events, and use them to bound the indifferentiability security of the JH mode (see Sects. 4 and 5).

Description of Game (JH, π, π^{-1}) The pseudocode for this game is given in Fig. 4(a). Following the definition provided in Sect. 2.3, the game Game(JH, π, π^{-1}) implements the JH hash function using the permutations π and π^{-1}. The ideal permutation π/π^{-1} has been implemented through lazy sampling. Lazy sampling is the postponement of sampling the

(a)

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

π(x)
11. if $x \notin Dom(D_\pi)$ then
        $D_\pi[x] \overset{\$}{\leftarrow} \{0,1\}^{2n} \setminus Rng(D_\pi)$;
12. return $D_\pi[x]$;

π^{-1}(y)
21. if $y \notin Rng(D_\pi)$ then
        $D_\pi^{-1}[y] \overset{\$}{\leftarrow} \{0,1\}^{2n} \setminus Dom(D_\pi)$;
22. return $D_\pi^{-1}[y]$;
```

(b)

```
RO(M)
001. if $M \in Dom(D_l)$ then
        return $D_l[M]$;
002. $h \overset{\$}{\leftarrow} \{0,1\}^n$; $D_l[M] = h$;
003. return $h$;

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

S(x)
101. $r \overset{\$}{\leftarrow} \{0,1\}^{2n}$;
102. $\mathscr{M} = \mathsf{MessageRecon}(x, T_s)$;
103. if $|\mathscr{M}| = 1$ then $r[0, n-1] = D_l[M] \oplus z$;
104. $D_s[x] = r$;
105. $\mathsf{FullGraph}(D_s)$;
106. return $r$;

S^{-1}(r)
300. If $\exists x_1, x_2 \in Dom(D_s)$ s.t. $D_s[x_1] = D_s[x_2] = r$
301.        then return "INVALID";
302. If $r \in Rng(D_s)$ then return $D_s^{-1}[r]$;
303. If $r \notin Rng(D_s)$ then $x \overset{\$}{\leftarrow} \{0,1\}^{2n}$;
304. If $x \notin Dom(D_s)$ then $D_s[x] = r$;
305. return $x$;
```

Fig. 4 The main games Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}). (**a**) Game(JH, π, π^{-1}): global variable is the table D_π ; (**b**) Game(RO, S, S^{-1}): Global variables are the tables D_l and D_s, and the graph T_s

Table 2 Definitions of *old* and *fresh* queries in various security games for JH

Current query	Game(JH, π, π^{-1})		G1		Game(RO, S, S^{-1})	
	Old	Fresh	Old	Fresh	Old	Fresh
l-query	$\in Dom(D_l)$	$\notin Dom(D_l)$	$\in Dom(D_l)$	$\notin Dom(D_l)$	$\in Dom(D_l)$	$\notin Dom(D_l)$
s-query	$\in Dom(D_\pi)$	$\notin Dom(D_\pi)$	$\in Dom(D_\pi)$	$\notin Dom(D_\pi)$	$\in Dom(D_s)$	$\notin Dom(D_s)$
s^{-1}-query	$\in Rng(D_\pi)$	$\notin Rng(D_\pi)$	$\in Rng(D_\pi)$	$\notin Rng(D_\pi)$	$\in Rng(D_s)$	$\notin Rng(D_s)$
π-query	$\in Dom(D_\pi)$	$\notin Dom(D_\pi)$	$\in Dom(D_\pi)$	$\notin Dom(D_\pi)$	–	–
π^{-1}-query	$\in Rng(D_\pi)$	$\notin Rng(D_\pi)$	$\in Rng(D_\pi)$	$\notin Rng(D_\pi)$	–	–

Suppose, the current l-query in Game(RO, S, S^{-1}) is *fresh*; then the entry in the table corresponding to "current", "l-query", and "Game(RO, S, S^{-1})" is "$\notin Dom(D_l)$"

random values until they are actually used for the first time. The query-response pairs for π/π^{-1} are stored in the table D_π.

Description of Game (**RO**, **S**, $\mathbf{S}^{-1}$) The pseudocode for this game is give in Fig. 4b. The functions S and S^{-1} are the simulators of the indifferentiability framework for JH. Construction of effective simulators is the most important part of the analysis of 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 that are indistinguishable from the output from 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

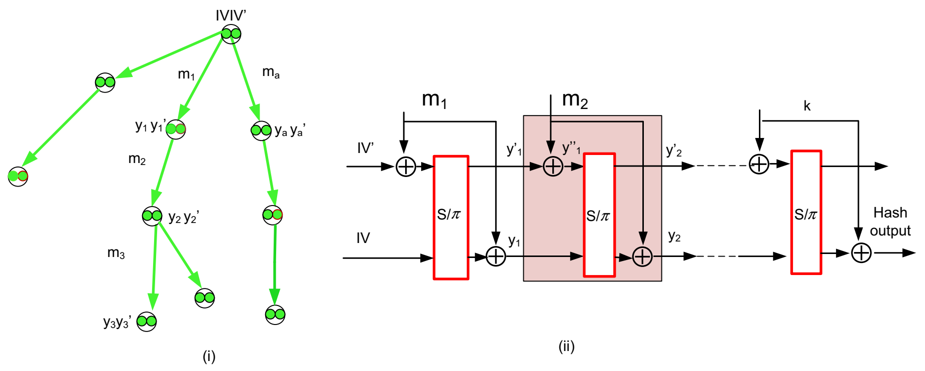

Fig. 5 All *arrows* and *dots* are n bits each. (**i**) The directed graph T_s (or T_π) which is updated by the subroutine $\mathsf{FullGraph}$ of Game(RO, S, S^{-1})(or $\mathsf{PartialGraph}$ of $G1$) (see Figs. 4b and 6). Example: The edge (y_1y_1', m_2, y_2y_2') is composed of the head node y_1y_1', the *arrow* m_2, and the tail node y_2y_2'. The left and right coordinates of a node (y_ay_a') is y_a and y_a'. (**ii**) JH mode with $M \overset{pad}{\rightarrow} m_1m_2\cdots m_k$. The *shaded region* shows the generation of the edge (y_1y_1', m_2, y_2y_2') in T_s using S (or in T_π using π);

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: S maintains a graph T_s using s-queries and responses, such that every path in T_s represents the execution of JH on some message. Correspondingly, any "JH-mode-compatible" message that can be reconstructed from the s-queries and responses is represented by some path on T_s. This helps S keep track of all "JH-mode-compatible" messages at all times. This is accomplished by a special subroutine $\mathsf{FullGraph}$. The pictorial representation of T_s is given in Fig. 5. In addition, whenever a new message M is found on T_s, S makes a crucial adjustment using a subroutine $\mathsf{MessageRecon}$, so that the distributions of $JH^\pi(M)$ and $\mathsf{RO}(M)$ are close. The complete description of $\mathsf{S}/\mathsf{S}^{-1}$ is as follows.

Description of the simulator-pair $\mathsf{S}/\mathsf{S}^{-1}$ We first describe two important subroutines used by the simulator-pair.

- $\mathsf{FullGraph}$ This routine updates the graph T_s using the elements in D_s in such a way that each path originating from the root (IV, IV') represents the execution of $\mathrm{JH}^{\mathsf{S}}(\cdot)$ on a prefix of some message. Additionally and more importantly, the graph T_s contains all possible paths derived from the elements in D_s; hence the name $\mathsf{FullGraph}$. See Fig. 5 for the pictorial description of how several components of the graph T_s are built. For example, suppose $M \overset{pad}{\rightarrow} m_1m_2M'$. Then the path $IVIV' \overset{m_1}{\rightarrow} y_1y_1' \overset{m_2}{\rightarrow} y_2y_2'$ represents the first two-block execution of $\mathrm{JH}^{\mathsf{S}}(M)$ where, $y_1y_1' = \mathsf{S}(IV||IV' \oplus 0||m_1) \oplus m_1||0$ and $y_2y_2' = \mathsf{S}(y_1||y_1' \oplus 0||m_2) \oplus m_2||0$.
- $\mathsf{MessageRecon}(x, T_s)$ The purpose of this routine is to reconstruct all messages M such that the final input to S in $\mathrm{JH}^{\mathsf{S}}(M)$ is the current s-query x. Hence $\mathrm{JH}^{\mathsf{S}}(M) = \mathsf{S}(x)[0, n-1] \oplus z$, where z is the final message-block of M after padding. The subroutine uses T_s to find all such M, by first calling the subroutine $\mathsf{FindNode}(y = x[0, n-1])$ to check whether there exist nodes in T_s with left-coordinate y. If present, then the subroutine $\mathsf{FindBranch}(y)$ collects all paths between the root (IV, IV') and the nodes yz'. A set $\mathscr{M}$ is returned, containing all the sequences of arrows on those paths—denoted

```
JH1($M$)
001. $M \overset{pad}{\rightarrow} m_1 m_2 \cdots m_{k-1} m_k$;
002. $y_0 = IV, y'_0 = IV'$;
003. for($i = 1, \cdots, k-1$){
004.     $y''_{i-1} = y'_{i-1} \oplus m_i$;
005.     $r = \pi(y_{i-1} y''_{i-1})$;
006.     $y_i y'_i = r \oplus m_i || 0$;
007.     if $y_{i-1} y''_{i-1}$ is fresh then
             PartialGraph($y_{i-1} y''_{i-1}, r$);}
008. $y''_{k-1} = y'_{k-1} \oplus m_k$;
009. **if Type3 then BAD := True;**
010. $r = \pi(y_{k-1} y''_{k-1})$;
011. if $y_{k-1} y''_{k-1}$ is fresh then
         PartialGraph($y_{k-1} y''_{k-1}, r$);
012. $D_l[M] = r[0, n-1] \oplus m_k$;
013. return $D_l[M]$;

MessageRecon($x, T_s$)
201. $x \overset{parse}{\rightarrow} yy'$;
202. if FindNode($y$) = 0 then return $\mathscr{M} = \emptyset$;
203. $\mathscr{M}'$ = FindBranch($y$);
204. $\mathscr{M} = \{\text{dePad}(Xz) \mid Xz' \in \mathscr{M}', z = z' \oplus y'\}$;
205. return $\mathscr{M}$;

$\pi(x)$
301. if $x \notin Dom(D_\pi)$ then
         $D_\pi[x] \overset{\$}{\leftarrow} \{0,1\}^{2n} \setminus Rng(D_\pi)$;
302. return $D_\pi[x]$;

S1($x$)
100. **if Type2 then BAD = True;**
101. $r = \pi(x)$;
102. $\mathscr{M}$=MessageRecon($x, T_s$);
103. if $|\mathscr{M}| = 1 \wedge M \notin Dom(D_l)$ then
         $D_l[M] = r[0, n-1] \oplus z$;
104. $D_s[x] = r$;
105. if $x$ is fresh then PartialGraph($x, r$);
106. return $r$;

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

S1$^{-1}$($r$)
601. **if Type4 then BAD = True;**
602. $x = \pi^{-1}(r)$;
603. **if Type1-c then BAD = True;**
604. $D_s[x] = r$;
605. return $x$;

$\pi^{-1}(r)$
501. if $r \notin Rng(D_\pi)$ then
         $D_\pi^{-1}[r] \overset{\$}{\leftarrow} \{0,1\}^{2n} \setminus Dom(D_\pi)$;
502. return $D_\pi^{-1}[r]$;
```

Fig. 6 Game $G1$: Global variables are the tables D_l, D_s and D_π, and the graphs T_π and T_s

by X—concatenated with $z = z' \oplus x[n, 2n-1]$. Notice that $\mathsf{dePad}(X||z) = M$. If no such $M \neq \perp$ is found, then the subroutine returns the empty set.

Using the above two subroutines, the simulator-pair $\mathsf{S}/\mathsf{S}^{-1}$ works as follows.

– For an s-query x, S assigns a uniformly sampled $2n$-bit value to r. The subroutine $\mathsf{MessageRecon}(x, T_s)$ is then invoked, which returns a set of messages $\mathscr{M}$. If $|\mathscr{M}| = 1$ then $r[0, n-1]$ is re-assigned the n-bit string $\mathsf{RO}(M) \oplus z$, where $M \in \mathscr{M}$ and $M \overset{pad}{\rightarrow} m_1 m_2 \cdots m_k = X||z$. Finally, D_s and T_s are updated using $\mathsf{FullGraph}$, and the value of r is returned.
– For an s^{-1}-query r, if there exist $x_1 \neq x_2$ such that $D_s[x_1] = D_s[x_2] = r$, then a special string "INVALID" is returned. If instead there exists a unique $x \in Dom(D_s)$ such that $D_s[x] = r$, then x is returned. The last possible case is if $r \notin Rng(D_s)$, and then x is assigned a $2n$-bit integer chosen according to the uniform distribution on $[0, 2^{2n} - 1]$. If $x \notin Dom(D_s)$ then $D_s[x]$ is assigned r. Finally x is returned.

Description of RO The oracle RO works as follows. Given an l-query M, RO first checks whether M has already been queried by S. In such a case, M already belongs to $Dom(D_l)$ and the RO returns $D_l[M]$. Otherwise, $D_l[M]$ is assigned a uniformly sampled n-bit value, which is eventually returned.
Description of G1 The pseudocode for this game is given in Fig. 6. The description of $G1$ apparently looks a bit artificial in the sense that it was constructed as a hybridization of the previous two games Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}). The purpose of this game is to be a transit point from Game(JH, π, π^{-1}) to Game(RO, S, S^{-1}) so that their difference in execution can be understood.

In the description of this game, we will ignore the bolded statements where the variable BAD is set, 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 $\left|\Pr\left[\mathscr{A}^{G1} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]\right|$ in Sect. 5.

Description of the simulator-pair $\mathsf{S1}/\mathsf{S1}^{-1}$ The intuition behind the design of this simulator-pair is similar to that of $\mathsf{S}/\mathsf{S}^{-1}$ of Game(RO, S, S^{-1}). The key difference is that rather than building a graph representing all "JH-compatible-messages", the graph in this game contains a *partial set* of "JH-compatible-messages". We will eventually see in Lemma 1, that if BAD events do not occur, then both the graphs are isomorphic. We now describe $\mathsf{S1}/\mathsf{S1}^{-1}$ in full detail using the following two subroutines.

- $\mathsf{PartialGraph}(x, r)$ The subroutine builds the graph T_π in such a way that each directed path originating from the root (IV, IV') represents the execution of $\mathrm{JH}^\pi(\cdot)$ on a prefix of some message (see Fig. 5). In constrast with $\mathsf{FullGraph}$, rather than building all possible paths using the fresh pair (x, r) and the old pairs in D_π, the $\mathsf{PartialGraph}$ augments the T_π by only one level; hence the name $\mathsf{PartialGraph}$. The details are as follows.
  First, the subroutine $\mathsf{CreateCoset}(y_c = x[0, n-1])$ is invoked, which returns a set Coset containing all nodes in T_π whose left-coordinate is y_c. The size of Coset determines the number of fresh nodes to be added to T_π in the the current iteration. Using the members of Coset and the new pair (x, r), new edges are constructed, stored in $\mathsf{EdgeNew}$, and added to T_π using the subroutine $\mathsf{AddEdge}$.
- $\mathsf{MessageRecon}(x, T_s)$. The current s-query is x, and the graph T_s is the maximally connected subgraph (of T_π) with the root-node (IV, IV'), generated by the s/s^{-1}-queries and responses stored in D_s. This subroutine has been described already in game Game(RO, S, S^{-1}).

Using the above two subroutines, the simulator-pair $\mathsf{S1}/\mathsf{S1}^{-1}$ works as follows.

- For an s-query x, r is assigned the value of $\pi(x)$. The ideal permutation π is implemented through lazy sampling. $\mathsf{MessageRecon}$ is called with (x, T_s), which returns a set of messages $\mathscr{M}$. If $|\mathscr{M}| = 1$, and if $M \in \mathscr{M}$ is not a previous l-query then $D_l[M]$ is re-assigned the value of $r[0, n-1] \oplus z$, where $M \overset{pad}{\rightarrow} Xz$. Then the table D_s is updated. If x is *fresh* then the routine $\mathsf{PartialGraph}$ is invoked on (x, r) to update the graph T_π. Finally, r is returned.
- For an s^{-1} query r, x is assigned the value of $\pi^{-1}(r)$, $D_s[x]$ is updated, and x is returned.

Description of $\mathit{JH1}$ If an l-query M has already been queried by $\mathsf{S1}$, then $D_l[M]$ is returned. Otherwise, JH1 mimics JH, in addition to updating the graph T_s whenever a fresh intermediate input is generated. Afterwards, $D_l[M]$ is assigned the value of $r[0, n-1] \oplus m_k$. Finally, $D_l[M]$ is returned.

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

Proposition 1 *For any distinguishing adversary* $\mathscr{A}$, *Game(JH,* π, π^{-1}*)* $\equiv G1$.

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 $\mathsf{S1}^{-1}(x) = \pi^{-1}(x)$. Likewise, from the descriptions of JH1 and JH, for all $M \in \{0, 1\}^*$, $\mathrm{JH1}(M) = \mathrm{JH}(M)$. □

A round of $G1$ *and Game*(RO, S, S^{-1}) A round is defined based on the type of the submitted query.

- s- QUERY In the game $G1$, a round spans the lines 100 through 106 (Fig. 6). For the game Game(RO, S, S^{-1}), a round spans the lines 101 through 106 (Fig. 4b).
- s^{-1}- QUERY In the game $G1$, a round spans the lines 601 through 605. For Game(RO, S, S^{-1}) a round spans the lines 300 through 305.
- l- QUERY Let $M \overset{pad}{\rightarrow} m_1 m_2 \cdots m_k$. For the game $G1$, the lines 004 through 007 form a round for the message-blocks m_1, m_2, ... and m_{k-1}. For the last block, m_k, the round is between the lines 008 and 013. For the Game(RO, S, S^{-1}), it is not specified how the random oracle $\mathsf{RO}(\cdot)$ processes the individual message-blocks m_j $(1 \le j \le k)$ internally. We assume that it processes the message-blocks sequentially and the time taken to process each block is equal.

Note that the sum of the numbers of message-blocks, s-queries and s^{-1}-queries before the $i+1$st round is i.

Time complexity of the simulator-pair $\mathsf{S}/\mathsf{S}^{-1}$ *in Game*(RO, S, S^{-1}) Since there are at most i short queries and responses after i rounds, the maximum number of distinct edges (or distinct nodes) in T_s is i^2 after i rounds. This follows because one edge (or one node) can be constructed from a pair of short queries (and responses), and that there are at most i^2 pairs of short queries. Therefore, to construct T_s at the i-th round, the amount of time required is $\mathcal{O}(i^4)$, since the maximum number of distinct edges in a path of T_s is i^2 and the maximum number of distinct paths in T_s is also i^2 (after i rounds). Now, if the adversary submits σ queries, then the time complexity to construct T_s is $\mathcal{O}(\sigma^5)$, as $\sum_1^{\sigma} i^4 = \mathcal{O}(\sigma^5)$. Since the time to construct T_s dominates over all the other steps, the simulator time complexity is also $\mathcal{O}(\sigma^5)$ in the worst case.

The events Type1, Type2, Type3, and Type4 of $G1$ are still not defined. These events are used to tell apart the game $G1$ from the game Game(RO, S, S^{-1}). We describe them in the next section.

4 Definition of the BAD events

Events GOOD_i *and* BAD_i BAD_i denotes the event when the variable BAD is set during round i of $G1$. Let the symbol GOOD_i denote the event $\neg \bigvee_{j=1}^{i} \mathsf{BAD}_i$. The symbol GOOD_0 denotes the event when no queries are submitted.

From a high level, the intuition behind the construction of the BAD_i event is straightforward: we will show that if BAD_i does not occur, and if GOOD_{i-1} did occur, then the views of $G1$ and Game(RO, S, S^{-1}) (after i rounds) are identically distributed for *any* attacker $\mathscr{A}$. Using the above facts, we will show

$$\left|\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]\right| \le \sum_{i=1}^{\sigma} \Pr\left[\mathsf{BAD}_i \mid \mathsf{GOOD}_{i-1}\right]. \tag{1}$$

We will then establish a concrete upper bound for this inequality in Sect. 5. In the next few subsections, we define the Type1, Type2, Type3 and Type4 events of the game $G1$ (see Fig. 6).

4.1 Type1 events: collision on T_π

Let (x, r) be a fresh pair of π-query and response generated at round i. Observe that such a fresh pair *always* invokes the subroutine $\mathsf{PartialGraph}$. Type1 events—which are due to π-

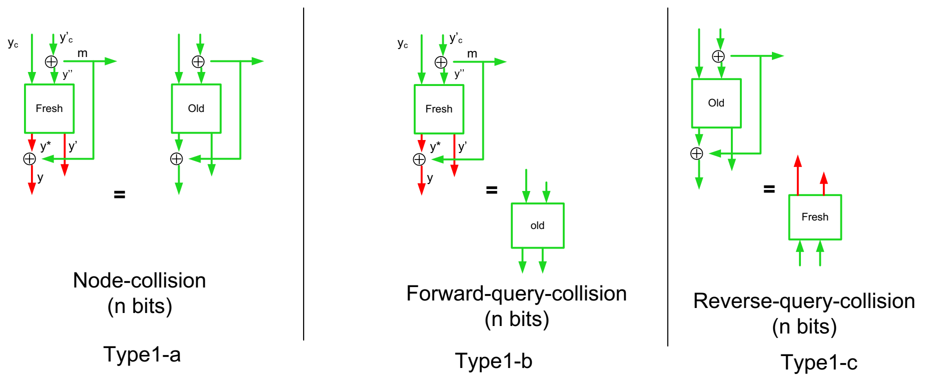

Fig. 7 Type1 events of game $G1$ defined in Fig. 6. All *arrows* are n bits each. *Red arrow* denotes fresh n bits of output from the ideal permutation π/π^{-1}. The *symbol* "=" denotes n-bit equality (Color figure online)

query and its response—are shown in Fig. 7. We divide this type into two subcases. Suppose $(y_c y'_c, m, yy')$ is a new edge generated from a new π-query/response (x, r).

- *Event Type1-a* (Fig. 7(Type1-a)) This event occurs if y collides with the least-significant n bits (or the left-coordinate) of a node already in T_π.
- *Event Type1-b* (Fig. 7(Type1-b)) This event occurs if y collides with the least-significant n bits of a query already in D_π.
- *Event Type1-c* (Fig. 7(Type1-c)) This event occurs, if the least-significant n bits of output of a fresh π^{-1}-query matches with the left-coordinate of a node already in T_s.

4.2 Events Type2, 3 and 4: current short query is old

Before we define these events, we first classify all the query-response pairs stored in D_π prior to the submission of the current query, according to their known and unknown parts. The known part of a query-response pair is the part that is present in the view of the game $G1$, while the unknown part is not present in the view. We observe that there are seven types of such a pair, and we denote them by Q0, Q1, Q2, Q3, Q4, Q5 and Q6 in Fig. 8a(i) and (ii); the head and tail nodes in each type denote the input and output, each of size $2n$ bits. Down, up and two-sided arrows indicate π-query, π^{-1}-query and any of the two, respectively. The red and green circles denote the unknown and the known parts of size n bits each. The queries of type Q0 have no red circles, since they are s/s^{-1} queries. The remaining six types are generated due to the intermediate π calls during the processing of l-queries; these queries have *at least* one *red* circle. The Q5 type can be further divided into two subtypes Q5-1 and Q5-2 according to its position in the graph T_π. If all the query-response pairs preceding the Q5 query are of type Q0 then it is Q5-1, otherwise it is type Q5-2. We cannot have any other type beyond Q0, Q1, Q2, Q3, Q4, Q5, Q6 because in any given node we cannot have the leftmost n bits of input or output be unknown while the rightmost n bits are known; this fact is readily clear by observing the XOR operations between the message-blocks, queries and responses occurring in the JH mode. Note that the message-blocks are always known.

- *Event Type2* (Fig. 8a) A Type2 event occurs when the current s-query is equal to an old query of type Q1, Q2, Q3, or Q4.
- *Event Type3* (Fig. 8b(i)–(iii)) Let M be the current l-query such that $M \overset{pad}{\rightarrow} m_1 m_2 \cdots m_k$ was already present as a branch in T_π, but not in T_s; such a branch is called a red branch

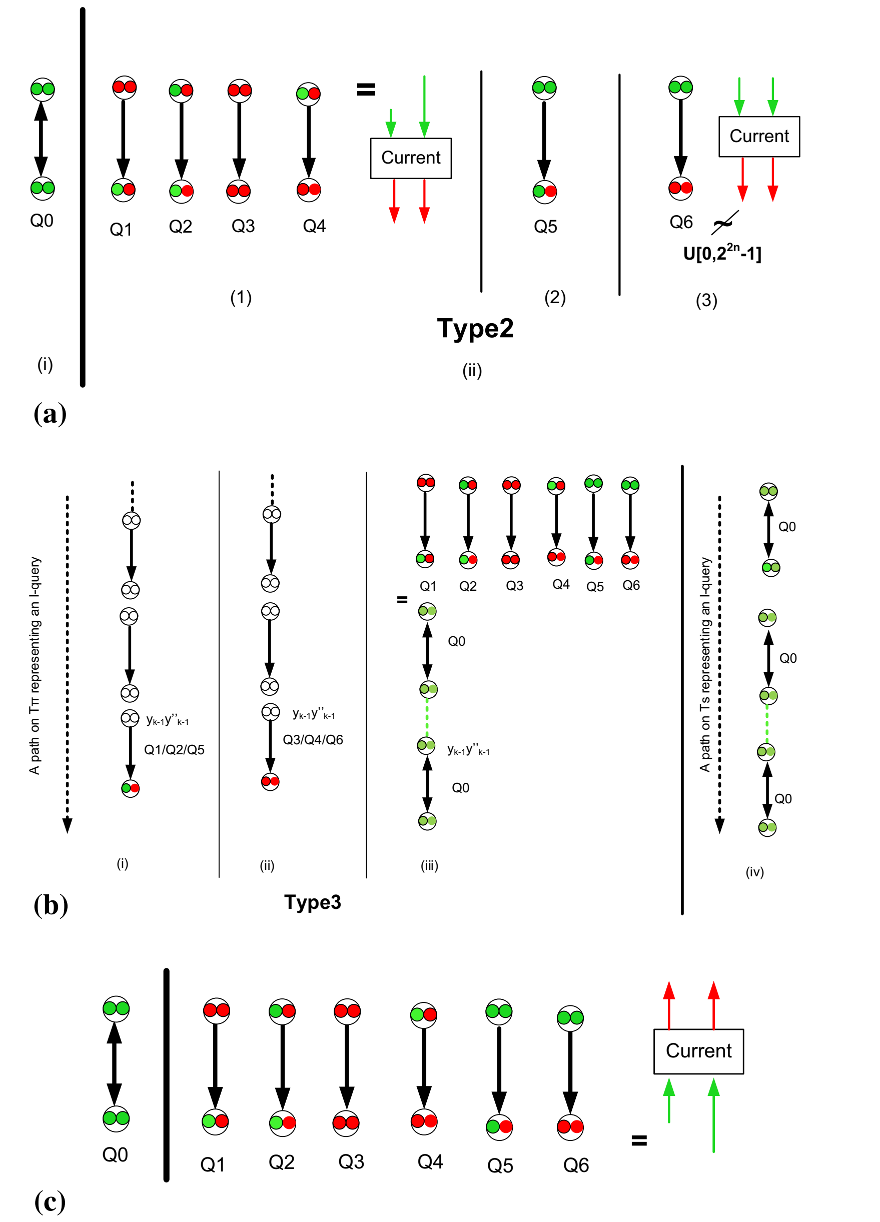

Fig. 8 Pictorial description of Type2, Type3 and Type4 events of the game $G1$ (Fig. 6). *Green circle*, or *green arrow* denotes n bits of information present in the view of the game. *Red circle* or *red arrow* denotes n bits of information not present in the view. *Black arrow* is not used to denote any information; it denotes the transition from input to output. The *symbol* "=" and "==" denote events representing n-bit and $2n$-bit equality respectively. (**a**) (*i*) and (*ii*): *Q0*, *Q1*, *Q2*, *Q3*, *Q4*, *Q5* and *Q6* denote seven types of π/π^{-1}-query and response; Type Q5 has further been divided into Q5-1 and Q5-2. The corresponding Type2 events are also shown; (**b**) Different types of a branch in the graph T_π. (*i*)—(*iii*) are called *red* branches since they exist in T_π, but not in T_s; the corresponding Type3 events associated with *red* branches are described in Sect. 4.2. (iv) A *green* branch is a branch in the graph T_s. The final input to π is denoted by $y_{k-1}y''_{k-1}$ in all cases; (**c**) Type4 events of game $G1$ (Color figure online)

since it has at least n bits of unknown part. A Type3 event occurs if the current π-query is the final π-query of a red branch.

- *Event Type4* (Fig. 8c) The Type4 event occurs, if the current s^{-1}-query is equal to an old query of type Q1, Q2, Q3, Q4, Q5, or Q6.

4.3 Computational paradigm

To prove the inequality 1, we will need the following lemma.

Lemma 1 (Graph Isomorphism Lemma) *Given GOOD_i and $V_1^i = V_2^i$, the graphs T_s for the games $G1$ and Game$(\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})$ are isomorphic after i rounds.*

Proof For each fresh π/π^{-1}-query, the graph T_π for game $G1$ is augmented in one phase (see the subroutine PartialGraph of Fig. 6). In that phase, all possible nodes generated from a fresh π-query are added to the graph T_π. A straightforward analysis of the Type1-a, b and c events shows that if these events do not occur then no nodes can be added beyond this phase. In other words, if Type1-a , b and c events do not occur in i rounds then the graph T_π contains all possible paths generated from all elements stored in the table D_π in i rounds with root (IV, IV'). Note that the graph T_s is the maximally connected subgraph of T_π rooted at (IV, IV'), generated *only* by the s-queries and responses stored in D_s. Also recall that, due to absence of a Type-c event, no s^{-1} query can be added to the graph T_π. This implies that the graph T_s of the game $G1$ contains all paths generated from all s/s^{-1}-queries and responses with root (IV, IV').
Observe that the graph T_s for Game$(\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})$ also contains all paths generated from all s/s^{-1}-queries and responses with root (IV, IV'). Since $V_1^i = V_2^i$, the graphs T_s for $G1$ and Game$(\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})$ are isomorphic after i rounds. □

With the help of the events described in Sects. 4.1 and 4.2 we are equipped to prove

Theorem 1 *Let $\mathscr{A}$ be an indifferentiability adversary interacting with the games $G1$ and Game$(\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})$. If $\mathscr{A}$ is limited by σ queries, then*

$$\left|\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]\right| \le \sum_{i=1}^{\sigma} \Pr\left[\mathsf{BAD}_i \mid \mathsf{GOOD}_{i-1}\right].$$

Proof The event $\mathsf{GOOD}_{i-\frac{1}{2}}$ is defined as GOOD_{i-1} in addition to the BAD events Type2, Type3, and Type 4 having not occured in the i-th round. For brevity, $\mathsf{GOOD}_{(i+1)-\frac{1}{2}}$ will be denoted by $\mathsf{GOOD}_{i+\frac{1}{2}}$. We need to show two things:

$$\left|\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]\right| \le \Pr\left[\neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right], \tag{2}$$

$$\Pr\left[\neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right] \le \sum_{i=1}^{\sigma} \Pr\left[\mathsf{BAD}_i \mid \mathsf{GOOD}_{i-1}\right]. \tag{3}$$

The proof of 3 is straight-forward. To prove 2, we proceed in the following way. Observe

$$\begin{aligned}
&\left|\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]\right| \\
&= \left|\left(\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-\frac{1}{2}}\right]\right.\right. \\
&\quad \left. - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-\frac{1}{2}}\right]\right) \cdot \Pr\left[\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right] \\
&\quad + \left(\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1 \mid \neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right]\right. \\
&\quad \left.\left. - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1 \mid \neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right]\right) \cdot \Pr\left[\neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right]\right|. \qquad (4)
\end{aligned}$$

If we can show that

$$\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-\frac{1}{2}}\right] = \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-\frac{1}{2}}\right], \qquad (5)$$

then 4 reduces to 2, since $\left|\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1 \mid \neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right] - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1 \mid \neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right]\right| \leq 1$. As a result, we focus on establishing 5.

Let V_1^i and V_2^i denote the views of the games $G1$ and Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$) respectively, after i queries have been processed. To prove 5, it suffices to show that given $\mathsf{GOOD}_{\sigma-\frac{1}{2}}$, the views V_1^σ and V_2^σ are identically distributed. We do this by induction on the number of queries $i = \sigma$. When $i = 0$, then no query has been made; therefore the views are identical. We now assume the induction hypothesis holds, where the hypothesis is: given $\mathsf{GOOD}_{i-\frac{1}{2}}$, then V_1^i and V_2^i are identically distributed. We have to show that if $\mathsf{GOOD}_{i+\frac{1}{2}}$ occurred, then V_1^{i+1} and V_2^{i+1} are identically distributed. We do so by examining all possible cases based on a set of conditions for the game $G1$. The details are quite technical, however the main idea is that if no BAD events have occurred, then the graphs T_s in $G1$ and Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$) are isomorphic, from the Graph Isomorphism Lemma 1. The identical distribution of the views is an easy consequence of the isomorphism.

Let (I_1^{i+1}, O_1^{i+1}) and (I_2^{i+1}, O_2^{i+1}) denote the input-output pairs for the games $G1$ and Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$) respectively in the $i+1$st round.

Notice that if $V_1^i = V_2^i$, then the input views I_1^{i+1} and I_2^{i+1} are identically distributed. We also have Lemma 1 which shows that the graphs T_s in two games are isomorphic.

A little reflection shows that proving the induction step is now equivalent to showing that if $I_1^{i+1} = I_2^{i+1}$ then the output-views O_1^{i+1} and O_2^{i+1} are identically distributed. Let I^{i+1} denote the shared query input $I_1^{i+1} = I_2^{i+1}$.

We continue by considering all possible cases based on a set of conditions for the game $G1$ in the $i+1$st round; cases 1 through 9 consider when I_{i+1} is an s-query, cases 10 and 11 consider I_{i+1} to be an s^{-1}-query, while cases 12 through 17 consider when I_{i+1} is part of an l-query. Our decision tree produced the above 17 cases, which have been derived from a sequence of questions (see Fig. 9). The reader is invited to verify that all cases are considered.

Case 1 *s- query, $|\mathscr{M}| = 0$, and Fresh*

Implication The condition implies that O_1^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_\pi)$. (Fig. 6. Since the graphs T_s are isomorphic in both games $G1$ and Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$) by Lemma 1, $|\mathscr{M}| = 0$ for Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$) (Fig. 4b). This implies that O_2^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_s)$ (Fig. 4b).

Case 2 *s-query, $|\mathscr{M}| = 0$, not Fresh, and type Q6*

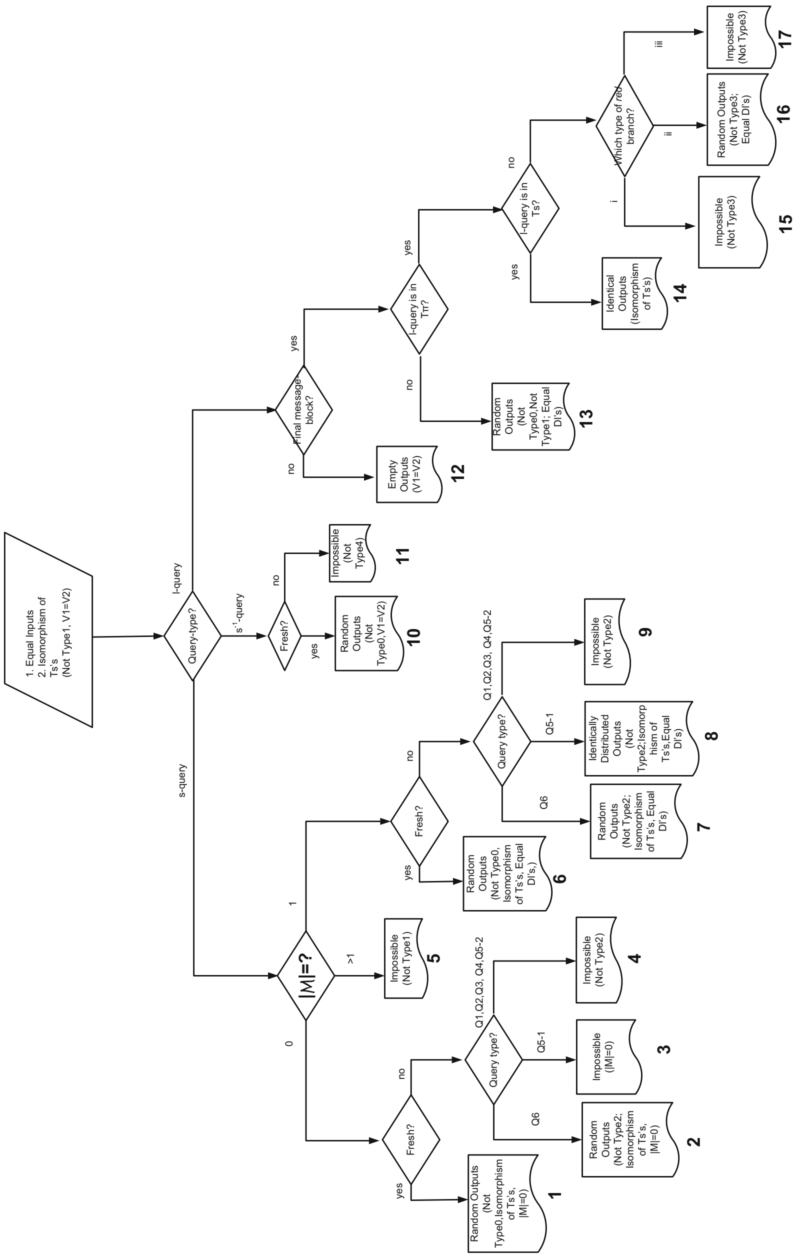
1. Equal Inputs
2. Isomorphism of Ts's
(Not Type1, V1=V2)
Query-type?
s-query
l-query
s⁻¹-query
|M|=?
0
1
>1
Fresh?
yes
no
Query type?
Q6
Q5-1
Q1,Q2,Q3, Q4,Q5-2
Random Outputs (Not Type0,Isomorphism of Ts's, |M|=0)
1
Random Outputs (Not Type2; Isomorphism of Ts's, |M|=0)
2
Impossible (|M|=0)
3
Impossible (Not Type2)
4
Impossible (Not Type1)
5
Random Outputs (Not Type0, Isomorphism of Ts's, Equal DI's,)
6
Random Outputs (Not Type2; Isomorphism of Ts's, Equal DI's)
7
Identically Distributed Outputs (Not Type2;isomorphism of Ts's,Equal DI's)
8
Impossible (Not Type2)
9
Random Outputs (Not Type0,V1=V2)
10
Impossible (Not Type4)
11
Final message block?
Empty Outputs (V1=V2)
12
l-query is in Tπ?
Random Outputs (Not Type0,Not Type1; Equal DI's)
13
l-query is in Ts?
Identical Outputs (Isomorphism of Ts's)
14
Which type of red branch?
i
ii
iii
Impossible (Not Type3)
15
Random Outputs (Not Type3; Equal DI's)
16
Impossible (Not Type3)
17

Fig. 9 The decision tree for the proof of the induction step of Theorem 1. The conditions for the game $G1$ are shown inside the *diamonds* of the decision tree. The *text* in each leaf-node shows the implications of the conditions to the outputs of games $G1$ and Game(RO, S, S^{-1}), while the reasons for such implications are described in brief inside the *bracket*

Implication The event $\mathsf{GOOD}_{i+\frac{1}{2}}$ implies that Type2 event did not occur for $G1$ in the current $i+1$th round; therefore, since $|\mathscr{M}| = 0$, O_1^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_\pi)$. As the graphs T_s of the games $G1$ and Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$) are isomorphic by Lemma 1, $|\mathscr{M}| = 0$ for Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$). This implies that $O_2^{i+1} = r$ follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_s)$.

Case 3 *s-query, $|\mathscr{M}| = 0$, not Fresh, and type Q5-1*

Implication This case is impossible since $|\mathscr{M}| = 0$ and I^{i+1} being of type Q5-1 contradict each other.

Case 4 *s-query, $|\mathscr{M}| = 0$, not Fresh, and type Q1, Q2, Q3, Q4, or Q5-2*

Implication This case is impossible since $\mathsf{GOOD}_{i+\frac{1}{2}}$ implies that Type2 event did not occur for $G1$ in the current $i+1$st round. The given conditions create a Type2 event.

Case 5 *s-query, $|\mathscr{M}| > 1$*

Implication If $|\mathscr{M}| > 1$ then we would have a node-collision in T_s. However, this is impossible since $\mathsf{GOOD}_{i+\frac{1}{2}}$ ensures that a Type1 event did not occur for $G1$ in the previous i rounds, and a node-collision in T_s is a Type1 event.

Case 6 *s-query, $|\mathscr{M}| = 1$, and Fresh*

Implication Since I^{i+1} is fresh, O_1^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_\pi)$. Now, for $G1$, $M \in \mathscr{M}$ implies that $M \notin Dom(D_l)$ in the first i rounds, since the current s-query I^{i+1} is fresh. Also note that, because $V_i^1 = V_i^2$ and the T_s's are isomorphic, we have that the D_l's in both games are identical. Therefore, for Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$), $M \notin Dom(D_l)$ in the first i rounds. This implies that O_2^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_s)$.

Case 7 *s-query, $|\mathscr{M}| = 1$, not Fresh, and type Q6*

Implication The event $\mathsf{GOOD}_{i+\frac{1}{2}}$ implies that a Type2 event did not occur in the $i+1$st round of $G1$; therefore, O_1^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_\pi)$. In $G1$, $M \in \mathscr{M}$ implies that $M \notin Dom(D_l)$ in the first i rounds, since the current s-query I^{i+1} is either of type Q3 or Q4, while the final π-query of any l-query *cannot* be of type Q3 or Q4. As in the previous case, $V_i^1 = V_i^2$ and the isomorphic T_s's together imply that the D_l in both games are identical. Therefore, for Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$), $M \notin Dom(D_l)$ in the first i rounds. This implies that O_2^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_s)$.

Case 8 *s-query, $|\mathscr{M}| = 1$, not Fresh, and type Q5-1*

Implication The event $\mathsf{GOOD}_{i+\frac{1}{2}}$ implies that Type2 event did not occur in the $i+1$st round of $G1$; therefore, $O_1^{i+1}[n, 2n-1]$ follows the uniform distribution on $[0, 2^n - 1] \setminus Rng(D_\pi)$, and $O_1^{i+1}[0, n-1]$ is a fixed value. Now, for $G1$, $M \in \mathscr{M}$ implies that $M \in Dom(D_l)$ after the first i rounds, since the current s-query I^{i+1} is of type Q5-1; also note that $O_1^{i+1}[0, n-1] = D_l[M] \oplus z$, where z is final block of M after padding. As in the previous case, $V_i^1 = V_i^2$ and the isomorphism of T_s's together imply that the D_l are identical in both games. Therefore, $O_2^{i+1}[0, n-1] = D_l[M] \oplus z$ (line 103 of Fig. 4b); also note that $O_2^{i+1}[n, 2n-1]$ follows the uniform distribution on $[0, 2^n - 1] \setminus Rng(D_s)$. In conclusion, O_1^{i+1} and O_2^{i+1} are identically distributed.

Case 9 *s-query, $|\mathscr{M}| = 0$, not Fresh, and type Q1, Q2, Q3, Q4, or Q5-2*

Implication This case is impossible since event Type2 did not occur in the current $i+1$st round, and, therefore, I^{i+1} cannot be of type Q1, Q2, Q3, Q4.

Case 10 *s^{-1}-query and Fresh*

Implication The condition implies that O_1^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_\pi)$. Because $V_1^i = V_2^{i+1}$, we have that the s^{-1}-query is also a fresh query for

Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$). Also note that the tables D_s of both games are an identical permutation.Therefore, O_2^{i+1} follows the uniform distribution on $[0, 2^{2n} - 1] \setminus Rng(D_s)$.
Case 11 *s^{-1}-query and not Fresh*
Implication A Type4 event and the above condition contradict each other.
Case 12 *l-query and not Final Block*
Implication If $V_{i+1}^1 = V_{i+1}^2$ then $O_1^{i+1} = O_2^{i+1} = \lambda$, where λ is the empty string.
Case 13 *l-query, Final Block, l-query not in T_π*
Implication Let M be the l-query in question. Since the event $\mathsf{GOOD}_{i+\frac{1}{2}}$ implies that a Type1 event did not occur in the previous i rounds of $G1$, there are no node-collisions in the graph T_π. Therefore, the final π-query is fresh, and so O_1^{i+1} follows the uniform distribution on $[0, 2^n - 1] \setminus Rng(D_\pi)$. Now notice, the tables D_l in both games were identical when the l-query M was submitted; therefore, at that time of submission, $M \notin Dom(D_l)$ for both games. This ensures that $O_2^{i+1} = \mathsf{RO}(M)$ follows the uniform distribution on $[0, 2^n - 1] \setminus Rng(D_s)$.
Case 14 *l-query, Final Block, l-query in T_π, l-query in T_s*
Implication The graphs T_s in both games are isomorphic by Lemma 1. It follows that $O_1^{i+1} = O_2^{i+1}$.
Cases 15, 16 and 17 *l-query, Final Block, l-query in T_π, l-query not in T_s* I^{i+1} is the final message-block of the current l-query (denoted by M) which forms a *red* branch. Let the final π-query while processing the l-query M be denoted by $y_{k-1}y''_{k-1}$.
Case 15 Final *π-query is type Q1, Q2, or Q5*
Implication The above condition implies the occurrence of Type3-1 event in the $i+1$st round; therefore, we arrive at a contradiction.
Case 16 Final *π-query is type Q3, Q4, or Q6*
Implication Since a Type3-2 event did not occur in the $i+1$st round, O_1^{i+1} follows the uniform distribution on $[0, 2^n - 1] \setminus Rng(D_\pi)$. Also observe, for $G1$, the l-query M did not belong to $Dom(D_l)$ (when M was submitted), since the final π-query of any l-query cannot be of type Q3, Q4 or Q6. As the tables D_l of both games are identical, then for Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$) we have that $M \notin Dom(D_l)$ (when M was submitted). Therefore, $O_2^{i+1} = \mathsf{RO}(M)$, which follows the uniform distribution on $[0, 2^n - 1] \setminus Rng(D_s)$.
Case 17 Final *π-query is type Q0 and an intermediate query is type Q1, Q2, Q3, Q4, Q5, or Q6*
Implication This case is impossible since Type3-3 in the $i + 1$st round did not occur.
In summary, for each of the 17 cases above we have shown that the outputs O_1^{i+1} and O_2^{i+1} are identically distributed if the variable BAD is not set. This completes the proof of the induction step of Theorem 1. $\square$

5 Estimation of $\left|\Pr\left[\mathscr{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathscr{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]\right|$

We individually compute the probabilities of each of the events described in Sects. 4.1 and 4.2. We need the help of the following lemma to provide a rigorous analysis for the upper-bounds we compute in this section.

Lemma 2 (Correction factor) *If the advantage of an indifferentiability adversary $\mathscr{A}$ distinguishing between the games $G1$ and Game (RO,S), when limited by σ queries, is bounded by ε,* *then*

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

for some constant $C > 0$*, for all* $0 \leq i \leq \sigma$.

Proof Since $\varepsilon < 1$, $\Pr\big[\mathscr{A}$ sets BAD in $G1\big] \leq \varepsilon \leq 1 - \frac{1}{C}$ for some constant $C > 0$. Noting that $\Pr\big[\mathsf{GOOD}_i\big]$ is a decreasing function in i, the result follows. □

The Type1-a event guarantees that if T_π is GOOD_{i-1}, then it has at most i nodes. Assuming $i \leq 2^{n/2}$, from Fig. 7 we obtain,

$$\begin{aligned}\Pr\Big[\text{Type1}_i \mid \mathsf{GOOD}_{i-1}\Big] &\leq 3i/(2^n - i),\\ &= \frac{6Ci}{2^n},\end{aligned} \tag{6}$$

since for $i \leq 2^{n/2}$, then $(2^n - i) \geq \frac{1}{2}2^n$.

Using the definition of Type2, Type3, and Type4 events in Sect. 4, it is straightforward to deduce:

$$\begin{aligned}\Pr\big[\text{Type2}_i \mid \mathsf{GOOD}_{i-1}\big] &\leq \frac{2Ci}{2^n},\\ \Pr\big[\text{Type3}_i \mid \mathsf{GOOD}_{i-1}\big] &\leq \frac{2Ci}{2^n},\\ \Pr\big[\text{Type4}_i \mid \mathsf{GOOD}_{i-1}\big] &\leq \frac{2Ci}{2^n}.\end{aligned}$$

We conclude by combining the above bounds into the following inequality which holds for $1 \leq i \leq \sigma$:

$$\begin{aligned}\Pr\big[\mathsf{BAD}_i \mid \mathsf{GOOD}_{i-1}\big] \leq\ & \Pr\big[\text{Type1}_i \mid \mathsf{GOOD}_{i-1}\big]\\ &+ \Pr\big[\text{Type2}_i \mid \mathsf{GOOD}_{i-1}\big]\\ &+ \Pr\big[\text{Type3}_i \mid \mathsf{GOOD}_{i-1}\big]\\ &+ \Pr\big[\text{Type4}_i \mid \mathsf{GOOD}_{i-1}\big]\\ \leq\ & \frac{12Ci}{2^n}.\end{aligned} \tag{7}$$

Therefore, by Theorem 1, for all $\mathscr{A}$,

$$\begin{aligned}\Big|\Pr\big[\mathscr{A}^{G1} \Rightarrow 1\big] - \Pr\big[\mathscr{A}^{\text{Game}(\mathsf{RO},\mathsf{S},\mathsf{S}^{-1})} \Rightarrow 1\big]\Big| &\leq \sum_{i=1}^{\sigma} \Pr\big[\mathsf{BAD}_i \mid \mathsf{GOOD}_{i-1}\big]\\ &\leq 12C \sum_{i=1}^{\sigma} \frac{i}{2^n}\\ &= \mathscr{O}\Big(\frac{\sigma^2}{2^n}\Big).\end{aligned} \tag{8}$$

Using 8 and that the advantage ϵ is less than 1, we see that the adversary must use at least $2^{n/2}$ queries to distinguish between the games $G1$ and Game(RO, S, S^{-1}) (or between the games Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}), since $G1 \equiv$ Game(JH, π, π^{-1}) by Proposition 1). This yields the indifferentiability bound of $n/2$ bits for the JH mode.

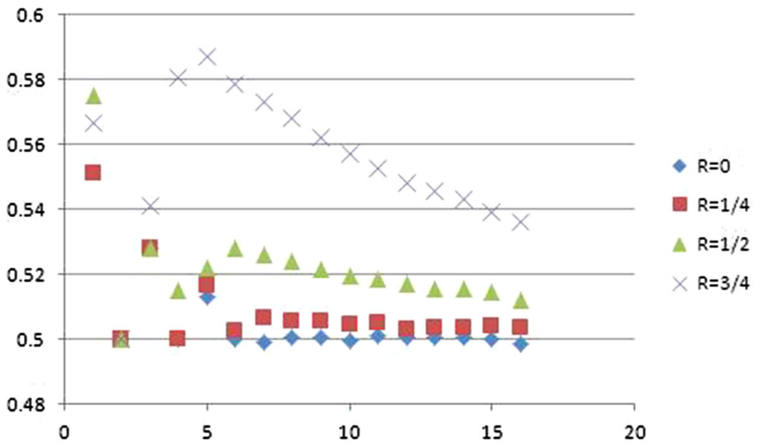

Fig. 10 Plot of experimental data of value of n versus the normalized logarithm of σ, $\log_2(\sigma)/n$, for the game $G1$ with various values of R, the proportion of reverse queries allowed

6 Experimental results

We performed a series of experiments verifying our theoretical framework. The motivation for doing experiments was three-fold: the first was to do a sanity check that the probability computed from experiments did not cross the theoretically obtained upper bound. Second, we wanted to identify the likely adversarial strategies that made our analysis tight. The final goal was to explore the possibility of the existence of a shorter proof for the bound obtained in Eq. 8. On all counts, we obtained useful results.

Our simple C implementation of the game $G1$ simulated the ideal permutation, π, with randomness supplied by cstdlib>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 of the probabilities of Type1 events, $\Pr\big[\text{Type1}_i \mid \mathsf{GOOD}_{i-1}\big]$, described in Sect. 4. Our experiments included as a parameter the proportion of reverse queries, R, allowed in the hopes that if an optimal adversarial strategy including reverse queries uses a positive proportion of reverse queries that we may discover a spike in performance near this proportion. Compiling these data we conclude that, as one would expect, when the proportion, R, approaches zero, the Type1-a event becomes dominant; whereas, when R approaches 1, the Type1-c event dominates.

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 $\sum_{i=1}^{\sigma} \Pr\big[\text{Type1}_i \mid \mathsf{GOOD}_{i-1}\big] \geq 0.5$.

We did not consider the Type2, Type3, and Type4 events, since, their probabilities are bounded by that of the Type1 events, for any efficient adversary. We found that choosing the values at which to place the first query uniformly at random from among all possible nodes was the most advantageous strategy for an adversary.

The results of the experiments following this method are summarized in Fig. 10. The data support the theoretically obtained bound of $\sigma = \Omega(2^{n/2})$ (see 8). 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 $1/2$.

The data indicate that the optimal adversarial strategy does not include the use of reverse queries. For each fixed $R < 1$, however, we observe that the data asymptotically approach $1/2$. Although it is the case that for $R = 1$, σ has an expected value of 2^{n-1}, the data support our result that, for our definition of Type1 bad events and any fixed value of $R < 1$, $\sigma = \Theta(2^{n/2})$.

Based on the experimental results, it seems likely that removing the Type2 to Type4 events, as well as the reverse queries from the JH indifferentiability framework may lead to the same upper bound. The only difference in such a case would be that the proof becomes much shorter. However, a theoretical argument to accomplish this is still an open problem.

7 Conclusion and open problems

The JH hash function was one of the finalist algorithms in the NIST SHA-3 hash function competition. In this paper we improve the indifferentiability security bound of the JH hash mode of operation from $n/3$ bits to $n/2$ bits, when it is used with a $2n$-bit permutation. This bound is optimal for JH-256, and the best, so far, for JH-512. We believe that the bound could be further improved, likely closer to n bits.

Our work leaves room for more research into the JH mode. It is somewhat remarkable that despite the absence of generic attacks with work-factor significantly lower than n bits, the proven pre-image, second pre-image and indifferentiability bounds for the JH mode are *only* up to $n/2$ bits. In future work we plan to use the proof technique from this paper to narrow the exponential gap between the upper and lower bounds of JH's indifferentiability security. It would be very interesting to find an attack that matches the indifferentiability bounds derived in this paper. Also, the complexity for the simulator could be improved.

References

1. Andreeva E., Bouillaguet C., Fouque P.A., Hoch J.J., Kelsey J., Shamir A., Zimmer S.: Second preimage attacks on dithered hash functions. In: Smart N.P. (ed.) EUROCRYPT 2008. Lecture Notes in Computer Science, vol. 4965, pp. 270–288. Springer, Berlin (2008).
2. Andreeva E., Mennink B., Preneel B.: On the indifferentiability of the Grøstl hash function. In: Garay J.A., Prisco R.D. (eds.) SCN. Lecture Notes in Computer Science, vol. 6280, pp. 88–105. Springer, Berlin (2010).
3. Andreeva E., Mennink B., Preneel B.: Security reductions of the second round SHA-3 candidates. In: Burmester M., Tsudik G., Magliveras S.S., Ilic I. (eds.) ISC. Lecture Notes in Computer Science, vol. 6531, pp. 39–53. Springer, Berlin (2010).
4. Andreeva E., Luykx A., Mennink B.: Provable security of BLAKE with non-ideal compression function. In: 3rd SHA-3 Candidate Conference (2012).
5. Bagheri N., Gauravaram P., Knudsen L.R.: Building indifferentiable compression functions from the PGV compression functions. Des. Codes Cryptogr. (2014). doi:10.1007/s10623-014-0020-z.
6. Bellare M., Ristenpart T.: Multi-property-preserving hash domain extension and the EMD transform. In: Lai X., Chen K. (eds.) ASIACRYPT 2006. Lecture Notes in Computer Science, vol. 4284, pp. 299–314. Springer, Berlin (2006).
7. Bernstein D.: CubeHash attack analysis (2.B.5). Online. http://cubehash.cr.yp.to/submission2/attacks. Accessed Feb 2012.
8. Bhattacharyya R., Mandal A., Nandi M.: Security analysis of the mode of JH hash function. In: Hong S., Iwata T. (eds.) FSE. Lecture Notes in Computer Science, vol. 6147, pp. 168–191. Springer, Berlin (2010).
9. Chang D., Nandi M., Yung M.: Indifferentiability of the hash algorithm BLAKE. Cryptology ePrint Archive, Report 2011/623. http://eprint.iacr.org/ (2011). Accessed Feb 2012.

10. Coron J.S., Dodis Y., Malinaud C., Puniya P.: Merkle–Damgård revisited: how to construct a hash function. In: Shoup V. (ed.) CRYPTO 2005. Lecture Notes in Computer Science, vol. 3621, pp. 430–448. Springer, Berlin (2005).
11. Dean R.D.: Formal aspects of mobile code security. PhD thesis, Princeton University (1999).
12. Fleischmann E., Gorski M., Lucks S.: Some observations on indifferentiability. In: Steinfeld R., Hawkes P. (eds.) ACISP. Lecture Notes in Computer Science, vol. 6168, pp. 117–134. Springer, Berlin (2010).
13. Gauravaram P., Kelsey J.: Linear-XOR and additive checksums don't protect Damgrd–Merkle hashes from generic attacks. In: Malkin T. (ed.) CT-RSA 2008. Lecture Notes in Computer Science, vol. 4964, pp. 36–51. Springer, Heidelberg (2008).
14. Gauravaram P., Knudsen L.R.: On randomizing hash functions to strengthen the security of digital signatures. In: Joux A. (ed.) EUROCRYPT 2009. Lecture Notes in Computer Science, vol. 5479, pp. 88–105. Springer, Berlin (2009).
15. Gauravaram P., Knudsen L.R.: Security analysis of randomize-hash-then-sign digital signatures. J. Cryptol. **25**(4), 748–779 (2012).
16. Gauravaram P., Kelsey J., Knudsen L.R., Thomsen S.: On hash functions using checksums. Int. J. Inf. Secur. **9**(2), 137–151 (2010).
17. Halevi S., Krawczyk H.: Strengthening digital signatures via randomized hashing. In: Dwork C. (ed.) CRYPTO 2006. Lecture Notes in Computer Science, vol. 4117, pp. 41–59. Springer, Berlin (2006).
18. Hoch J.J., Shamir A.: Breaking the ICE—finding multicollisions in iterated concatenated and expanded (ICE) hash functions. In: Robshaw M.J.B. (ed.) FSE. Lecture Notes in Computer Science, vol. 4047, pp. 179–194. Springer, Berlin (2006).
19. Joux A.: Multicollisions in iterated hash functions: application to cascaded constructions. In: Franklin M.K. (ed.) CRYPTO 2004. Lecture Notes in Computer Science, vol. 3152, pp. 306–316. Springer, Heidelberg (2004).
20. Kelsey J., Kohno T.: Herding hash functions and the Nostradamus attack. In: Vaudenay S. (ed.) EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 183–200. Springer, Berlin (2006).
21. Kelsey J., Schneier B.: Second preimages on n-bit hash functions for much less than 2^n work. In: Cramer R. (ed.) EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494, pp. 474–490. Springer, Heidelberg (2005).
22. Lee J., Hong D.: Collision resistance of the JH hash function. Cryptology ePrint Archive, Report 2011/019. http://eprint.iacr.org/ (2011). Accessed Feb 2012.
23. Maurer U.M., Renner R., Holenstein C.: Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In: Naor M. (ed.) Theory of Cryptography—TCC 2004. Lecture Notes in Computer Science, vol. 2951, pp. 21–39 Springer, Heidelberg (2004).
24. Nandi M., Stinson D.R.: Multicollision attacks on some generalized sequential hash functions. IEEE Trans. Inf. Theory **53**(2), 759–767 (2007).
25. NIST: Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family. Federal Register, 72(212), 2007. http://www.nist.gov/hash-competition. Accessed Feb 2012.
26. Ristenpart T., Shacham H., Shrimpton T.: Careful with composition: limitations of the indifferentiability framework. In: Paterson K.G. (ed.) EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 487–506. Springer, Berlin (2011).
27. Smith-Tone D., Tone C.: A measure of dependence for cryptographic primitives relative to ideal functions. Rocky Mountain J. Math (2015).
28. Wu H.: The JH hash function. In: The 1st SHA-3 Candidate Conference (2009).