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

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 is one of the five finalists in the ongoing NIST SHA-3 hash function competition. Despite several years of analysis, the indifferentiability security of the JH mode (with n-bit digest and $2n$-bit permutation) has remained remarkably low, *only* at $n/3$ bits (FSE 2010), while the other four finalist modes – with comparable parameter values – offer a security guarantee of $n/2$ bits. In this paper, we improve the indifferentiability security bound for the JH mode to $n/2$ bits (*e.g.* from 171 to 256 bits when $n = 512$). To put this into perspective, our result guarantees the *absence* of attacks on both 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 basic permutation is structurally strong. Our bounds are *optimal* for JH-256, and the best, so far, 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 the bad events. Our experimental data *strongly* supports the theoretically obtained results.

1 Introduction

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 $H : \{0,1\}^* \to \{0,1\}^n$. Assume that for a given input $M \in \{0,1\}^*$, H invokes an ideal object, say a random oracle $\mathsf{ro} : \{0,1\}^m \to \{0,1\}^n$, 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 $\mathsf{RO} : \{0,1\}^* \to \{0,1\}^n$.

Generic attacks against hash functions are plentiful in the literature. See, for example, Joux's multicollision attack [11], the Kelsey-Schneier expandable-message attack [13] and the Kelsey-Kohno herding attack [12], all on the popular Merkle-Damgärd hash mode. Generic attacks have also been reported on hash modes other than the plain Merkle-Damgärd mode. A few of these are the 2nd pre-image attacks on the dithered variants of the Merkle-Damgärd construction [1], a pre-image attack on the JH mode [7], a pre-image attack on the Sponge construction when used with a small-sized permutation [6], collision attacks on some concatenated hash functions [11], multicollisions in iterated concatenated and expanded hash functions [10], and multicollisions on some generalized sequential hash functions [16].

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.

Indifferentiability Security. The indifferentiability security framework was introduced by Maurer *et al.*[15] in 2004, and was first applied to analyze hash modes of operation by Coron *et al.*[9] 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 [2, 7, 8] 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, 1st/2nd preimage attacks – are inadequate to guarantee the security of the overlying protocol. As a result, it is now common to derive indifferentiability bounds for new proposed modes.

Previous analysis of the JH Mode. The JH hash function is one of the five finalist algorithms in the ongoing NIST SHA-3 hash function competition. The hash function uses an iterative mode which is novel in the sense that it is based on a permutation [18]. Several popular hash functions – such as the SHA-1/2 – are constructed instead using a blockcipher. 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 [7];[1] they have also shown a generic pre-image attack on the JH mode with (information theoretic) work which is slightly less than n bits. A year later, in [14], it was shown that the JH mode achieves the optimal collision resistance of up to $n/2$ bits. Very recently Andreeva, Mennink and Preneel have improved the 1st and 2nd pre-image resistance of the JH mode from $n/3$ to $n/2$ bits [4]. 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.

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 difficulties in our improved 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, which is able to withstand all indifferentiable adversaries up to (approximately) $2^{n/2}$ queries. 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 22 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 4.1 requires *only* three games. Compare this with the usual practice of tackling such problems using a sequence of a large number of games. The smaller number of games makes third-party verification of the proof a great deal easier.

Our indifferentiability bound guarantees the *absence of generic attacks* on the JH hash function based on $2n$-bit permutation with work less than $2^{n/2}$. When the digest-size $n = 256, 512$, the hash mode is resistance 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 using our bad events. Our experiments verify the theoretically obtained results. 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.

Table 1: The resistance of the JH mode against several attacks. Each number is in bits. The asterisk indicates the optimality of bound.

Mode of operation	Message block-length	Permutation size	First preimage	Second preimage	Collision resistance	Indiff. (old)	Indiff. (new)
JH-n	n	$2n$	$n/2$ [4]	$n/2$ [4]	$n/2^*$ [14]	$n/3$ [7]	$\boldsymbol{n/2}$
JH-512	512	1024	256	256	256^*	171	**256**
JH-256	512	1024	256^*	256^*	128^*	171	**256***

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. For concatenation of strings, we use $a||b$, or just ab if the meaning is clear. Let $x \overset{parse}{\rightarrow} x_1||x_2$ 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 $\Pr\big[A = X\big] = \Pr\big[B = X\big]$ for all $X \in S$, where $S = S_A = S_B$. Let

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

$Dom(T) = \{i \mid T[i] \neq \lambda\}$ and $Rng(T) = \{T[i] \mid T[i] \neq \lambda\}$. We write $\mathcal{A}^B$ to denote an Algorithm $\mathcal{A}$ with oracle access to B. 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. Finally, $\mathcal{U}[0, N]$ is the uniform distribution over the integers between 0 and N.

2 Indifferentiability Framework for JH

2.1 Description of the JH Mode

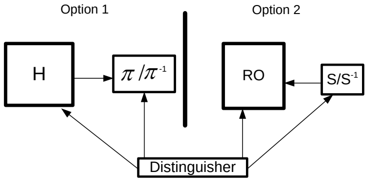


Figure 1: Indifferentiability framework for a hash function based on an ideal permutation. An arrow indicates the direction in which a query is submitted.

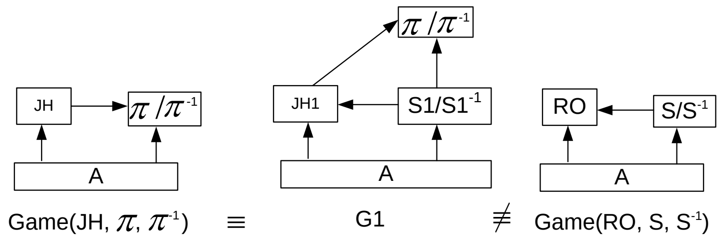


Figure 2: Schematic diagrams of the security games used in the indifferentiability framework for JH. The arrows show the directions in which the queries are submitted.

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 4(ii) and 3(a). The semantics for the notation $M \stackrel{pad}{\to} m_1 \cdots m_{k-1} m_k$ 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. For more details, the reader is referred to the original specification written by the JH designer [18].

2.2 Introduction to the Indifferentiability Framework

We will frequently refer to the use of a *random oracle*, as defined in Appendix A. 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 [9, 15].

Definition 2.1 (Indifferentiability framework) *[9] 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 [15].

Pictorial Description of Definition 2.1 (Figure 1). In the figure, the five entities involved in Definition 2.1 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 based on the ideal permutation π. In this setting, Definition 2.1 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 $\pi/\pi^{-1} : \{0,1\}^{2n} \to \{0,1\}^{2n}$ as the basic primitive of JH. To obtain the indifferentiability security bound, we follow the usual game-playing techniques [3, 5]. The schematic diagrams of the two cryptographic systems Option 1 and Option 2 (of Figure 1) are Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}) illustrated in Figure 2. 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 3(a).

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, when it interacts with the adversary $\mathcal{A}$. The view of the game G after j queries with respect to the adversary $\mathcal{A}$, is the sequence $\{(x_1y_1), \ldots, (x_jy_j)\}$. 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. For simplicity, we will *only* deal with games that expose identical interfaces to their adversaries. The definition of *equivalence of games* is provided in Appendix B.

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 2. The pseudocode for all the games is given in Figures 3 and 5.

The functionalities JH, JH1, and RO are mappings from $\{0,1\}^*$ to $\{0,1\}^n$. The function S is a mapping from $\{0,1\}^{2n}$ to $\{0,1\}^{2n}$. Also, π, π^{-1}, S1, and $\mathsf{S1}^{-1}$ are all permutations on $\{0,1\}^{2n}$, while S^{-1} is function from $\{0,1\}^{2n}$ to $\{0,1\}^{2n} \cup \{\text{"INVALID"}\}$. We call any query submitted to JH, JH1, or RO an l-query, short for long query. Likewise, we refer to queries to S or S1 as s-queries, (for short query). An s^{-1}-query is a query submitted to S^{-1} or $\mathsf{S1}^{-1}$.

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 three types according to their location in the tables: (1) a *current* query denotes the query in question, which should be evident from context; (2) an *old* query is one which is already present in the database D_l, D_s or D_π; (3) a *fresh* query is when the current query is not present in any of the databases D_l, D_s or D_π. We assume that identical queries are not submitted by the adversary more than once.

Description of Game(JH, π, π^{-1}) (Figure 3(a)). Following the definition provided in Section 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} have been implemented through lazy sampling. The query-response pairs for π/π^{-1} are stored in the table D_π.

Description of Game(RO, S, S^{-1}) (Figure 3(b)). The functions S and S^{-1} of this game 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 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.

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 FullGraph. See Figure 4 for the pictorial description of how several components of the graph T_s are built. For example, suppose $M \overset{pad}{\to} m_1m_2M'$. Then the path $IVIV' \overset{m_1}{\to} y_1y_1' \overset{m_2}{\to} y_2y_2'$ 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 $y_2y_2' = \mathsf{S}(y_1||y_1' \oplus 0||m_2) \oplus m_2||0$.

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 FindNode$(y = x[0, n-1])$ to check whether there exists nodes in T_s with left-coordinate y. If present, then the subroutine FindBranch(y) collects all paths between the root (IV, IV') and the nodes yz'. A set $\mathcal{M}$ is returned, containing all the sequences of arrows on those

$\mathrm{JH}(M)$

01. $M \stackrel{pad}{\rightarrow} m_1 m_2 \ldots m_{k-1} m_k$;
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;

$\pi(x)$

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

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

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

(a) Game(JH, π, π^{-1}): Global variable is the table D_π.

$\mathsf{RO}(M)$

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

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

201. $x \stackrel{parse}{\rightarrow} yy'$;
202. if $\mathsf{FindNode}(y) = 0$ then 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 $\mathcal{M}$;

$\mathsf{S}(x)$

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

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

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 $x \stackrel{\$}{\leftarrow} \{0,1\}^{2n}$;
304. If $x \notin Dom(D_s)$ then $D_s[x] = r$;
305. return x;

(b) Game(RO, S, S^{-1}): Global variables are the tables D_l and D_s, and the graph T_s.

Figure 3: The main games Game(JH, π, π^{-1}) and Game(RO, S, S^{-1})

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

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 $\mathcal{M}$. If $|\mathcal{M}| = 1$ then $r[0, n-1]$ is assigned the n-bit string $\mathsf{RO}(M) \oplus z$, where $M \in \mathcal{M}$ and $M \stackrel{pad}{\rightarrow} m_1 m_2 \cdots m_k = X||z$. Finally, D_s and T_s are updated, and the value of r is returned. In Appendix C, we show that the worst-case running time of the S on the i-th query is $\mathcal{O}(i^4)$.

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 $\mathcal{U}[0, 2^{2n} - 1]$. If $x \notin Dom(D_s)$ then $D_s[x]$ is assigned r. Finally x is returned.

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 G_1 (Fig. 5). The description of the game G_1 apparently looks a bit artificial in the sense 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 transit point from Game(JH, π, π^{-1}) to Game(RO, S, S^{-1}) so that their difference in execution can be understood.

First, in the description of this game, we omit the 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 $\big|\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1\big] - \Pr\big[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\big]\big|$ in Section 5. Now we describe the subroutines used by this game.

$\mathsf{PartialGraph}(x, r)$: 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 (depicted in Figure 4). 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 in at most one phase; hence the name $\mathsf{PartialGraph}$. The details are as follows.

First, the subroutine $\mathsf{CreateCoset}(y_c = x[0, n-1])$ is invoked, that returns a set Coset containing all nodes in T_π

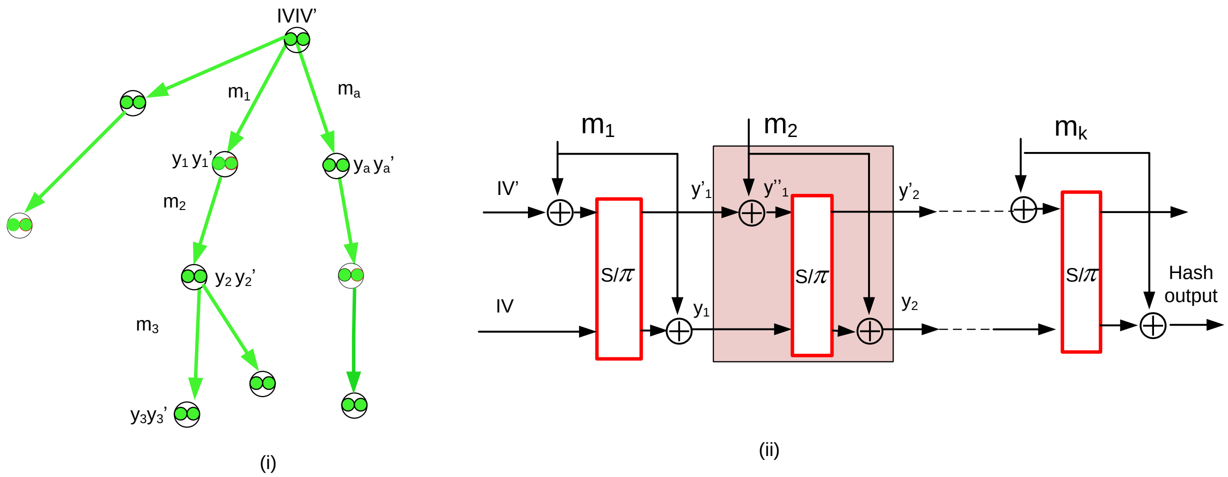


Figure 4: All arrows and dots are n bits each. (i) The directed graph T_s (or T_π) which is updated by the subroutine FullGraph of Game(RO, S, S^{-1})(or PartialGraph of G_1) (see Figures 3(b) and 5). Example: The edge (y_1y_1', m_2, y_2y_2') is composed of the head node y_1y_1', the arrow m_2, and the 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 $M \stackrel{pad}{\to} m_1m_2\cdots m_k$. 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);

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 EdgeNew, and added to T_π using the subroutine AddEdge.

MessageRecon(x, T_s)**:** 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; x is the current s-query. This subroutine has been described already in game Game(RO, S, S^{-1}).

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

If an l-query M has already been queried by 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, the $D_l[M]$ 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 3.1 *For any distinguishing adversary $\mathcal{A}$, Game(RO, S, S^{-1}) $\equiv G_1$.*

Proof. From the description of 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 $M \in \{0,1\}^*$, JH1(M) = JH(M). □

The events Type0, Type1, Type2, Type3, and Type4 of G_1 are still not defined. These events finally tell apart the game G_1 from the game Game(RO, S, S^{-1}). We describe them in the following sections.

4 Definition of the events: BAD$_i$, GOOD$_i$ and a few more

Round of a Game. A round of a game is defined based on the type of the submitted query.

For an s-query: For the game G_1, a round spans the lines 100 through 106 (Fig. 5). For the game Game(RO, S, S^{-1}), a round spans the lines 101 through 106 (Fig. 3(b)).

For an s^{-1}-query: For the game G_1, a round spans the lines 601 through 606. For Game(RO, S, S^{-1}) a round spans the lines 300 through 305.

For an l-query: Let $M \stackrel{pad}{\to} m_1m_2\cdots m_k$. For the game G_1, the lines 004 through 007 form a round for the message-blocks m_1, m_2, $\cdots$ and m_{k-1}. For the last block, m_k, the round is between the lines 008 and 014. For the Game(RO, S, S^{-1}), it is not specified how the random oracle RO$(\cdot)$ processes the individual message-blocks m_j $(1 \leq j \leq k)$

Figure 5: Game G_1: Global variables are the tables D_l, D_s and D_π, and the graphs T_π and T_s.

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. $\quad y''_{i-1} = y'_{i-1} \oplus m_i$;
005. $\quad r = \pi(y_{i-1} y''_{i-1})$;
006. $\quad y_i y'_i = r \oplus m_i || 0$;
007. $\quad$ if $y_{i-1} y''_{i-1}$ is fresh then
$\qquad$ PartialGraph$(y_{i-1} y''_{i-1}, r)$;$\}$
008. $y''_{k-1} = y'_{k-1} \oplus m_k$;
009. If $M \in Dom(D_l)$ then
$\qquad$ **if Type3 then BAD := True;**
010. $r = \pi(y_{k-1} y''_{k-1})$;
011. **if Type0-b then BAD =True;**
012. if $y_{k-1} y''_{k-1}$ is fresh then
$\qquad$ PartialGraph$(y_{k-1} y''_{k-1}, r)$;
013. $D_l[M] = r[0, n-1] \oplus m_k$;
014. return $D_l[M]$;

MessageRecon(x, T_s)

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

$\pi(x)$

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

S1(x)

100. **if Type2 then BAD =True;**
101. $r = \pi(x)$;
102. **if Type0-a then BAD =True;**
103. $\mathcal{M}$=MessageRecon(x, T_s);
104. if $|\mathcal{M}| = 1 \wedge M \notin Dom(D_l)$ then
$\qquad D_l[M] = r[0, n-1] \oplus z$;
105. $D_s[x] = r$;
106. if x is fresh then PartialGraph(x, r);
107. 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 $= \{(y_c y'_c, m, yy') \mid$
$\qquad y_c y'_c \in$ 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. $\qquad$ **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 Type0-c then BAD =True;**
604. **if Type1-c then BAD =True;**
605. $D_s[x] = r$;
606. return x;

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

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

internally. We assume that it processes the message-blocks sequentially and the time taken to process each block is equal.
The sum of the numbers of message-blocks, s-queries and s^{-1}-queries before the $i+1$st round is i.
Events GOOD_i and BAD_i. BAD_i denotes the event when the variable BAD is set during round i of G_1. The event BAD_{i^*} occurs when Type0, Type2, Type3 or Type4 events occur in the i-th round. Let the symbol GOOD_i denote the event $\neg \bigvee_{j=1}^{i} \mathsf{BAD}_i$. The event $\mathsf{GOOD}_{i-\frac{1}{2}}$ is defined as $\mathsf{GOOD}_{i-1} \wedge \neg \mathsf{BAD}_{i^*}$. For brevity, $\mathsf{GOOD}_{(i+1)-\frac{1}{2}}$ will be denoted by $\mathsf{GOOD}_{i+\frac{1}{2}}$. 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 straight-forward: we will show that if BAD_i does not occur, and if GOOD_{i-1} did occur, then the views of G_1 and Game$(\mathsf{RO},\mathsf{S},\mathsf{S}^{-1})$ (after i rounds) are identically distributed for *any* attacker $\mathcal{A}$. Using the above fact the following theorem can be established.

Theorem 4.1 (Computational Paradigm) *Let $\mathcal{A}$ be an indifferentiability adversary interacting with the games G_1 and Game$(\mathsf{RO},\mathsf{S},\mathsf{S}^{-1})$. If $\mathcal{A}$ is limited by σ queries, then*

$$\left|\Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]\right| \leq \Pr\left[\neg \mathsf{GOOD}_{\sigma-\frac{1}{2}}\right] \leq \sum_{i=1}^{\sigma} \Pr\left[\mathsf{BAD}_i \mid \mathsf{GOOD}_{i-1}\right].$$

Proof. We postpone the proof until Section 4.3. □

In the next few subsections, we concretely define the Type0, Type1, Type2, Type3 and Type4 events of the game G_1 (see Figure 5).

4.1 Events Type0 and Type1: current π/π^{-1}-query is fresh (total 6 cases)

4.1.1 Event Type0: Distance of random permutation from the uniform (3 cases)

Type0 event occurs when the output of a fresh π/π^{-1}-query is *distinguishable* from the uniform distribution $\mathcal{U}[0, 2^{2n}-1]$. A Type0 event can be of three types: event Type0-a occurs when a fresh π-query is an s-query; event Type0-b occurs when a fresh π-query is the final π-query of an l-query; event Type0-c occurs when an s^{-1}-query is a fresh π^{-1}-query.

4.1.2 Event Type1: Collision on T_π (3 cases)

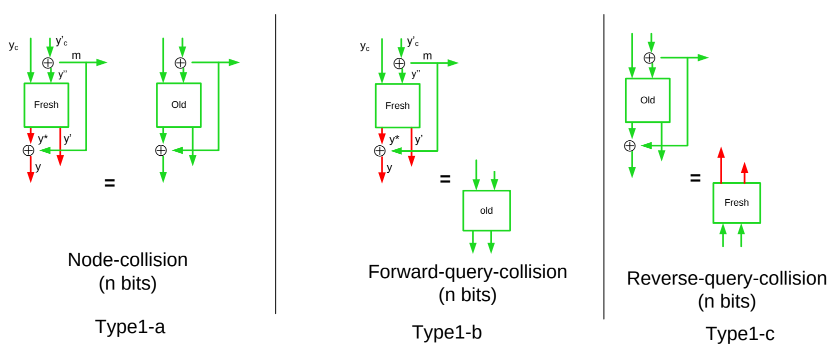


Figure 6: Type1 events of game G_1 defined inFigure 5. All arrows are n bits each. Red arrow denotes fresh n bits of output from the ideal permutation π/π^{-1}. The symbol "=" denotes n-bit equality.

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 – that are due to π-query and its response – are shown in Figure 6. 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** (Figure 6(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** (Figure 6(Type1-b)): This event occurs if y collides with the least-significant n bits of a query already in D_π.

Type1 event, which is due to a fresh π^{-1}-query and its response is denoted by Type1-c.
• **Event Type1-c** (Figure 6(Type1-c)): This event occurs, if the least-significant n bits of output of a π^{-1}-query matches with the left-coordinate of a node already in T_s.

4.2 Events Type2, 3 and 4: current π/π^{-1}-query is old (total 16 cases)

Before we define this event, we first classify all the old query-response pairs for the oracles π/π^{-1} stored in D_π, according to its known and unknown parts. The known part of a query-response pair is the part that is present in the view of the game G_1, 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 Figure 7(a)(i) and (ii); the head and tail nodes in each type denote the input and output, each of size $2n$ bits. Two-sided arrowhead indicates that the corresponding input-output pair is generated from either a π-query or a π^{-1}-query. The *red* and *green* circles denote the unknown and the known parts of size n bits each. The queries of type Q0 are the old s/s^{-1}-queries already present in the table D_π (*i.e.,* the π/π^{-1}-queries submitted to the simulators $\mathsf{S1}/\mathsf{S1}^{-1}$); since this query-response is present in the view, it has no *red* circles. 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 (see Figure 9 of Appendix E.1): 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.

4.2.1 Event Type2: current s-query is old (total 7 cases)

This event is presented in Fig. 7(a). A Type2 event occurs when one of the following conditions occurs. There are three subcases Type2-1,-2 and-3 (see Fig. 7(a)(ii)).
TYPE2-1: If the current s-query is equal to an old query which is one of the types Q1, Q2, Q3 and Q4.
TYPE2-2: This event occurs in relation to an old query of type Q5. This case is divided into two subcases as described in Figure 9 of Appendix E.1. (i) If the current s-query is equal to an old query of type Q5-1 and the most significant n bits of output are *distinguishable* from the uniform distribution $\mathcal{U}[0, 2^n - 1]$. (ii) If the current s-query is equal to an old query of type Q5-2.
TYPE2-3: If the current s-query is equal to an old query of type Q6 and the $2n$ bits output are *distinguishable* from the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$.

4.2.2 Event Type3: current l-query forms a *red* branch (total 3 cases)

A *red* branch. Let M be the current l-query such that $M \stackrel{pad}{\rightarrow} m_1 m_2 \cdots m_k$ was already present as a branch in T_π, but not in T_s (see Fig. 7(b)(i) to (iii)); such a branch is called a *red* branch since it has at least n bits of unknown part. We divide a *red* branch into three types, according to the final π-query – denoted by $y_{k-1}y''_{k-1}$ –in the computation of $JH^\pi(M)$. The three types of a *red* branch are below: (i) $y_{k-1}y''_{k-1}$ is one of types Q1, Q2 and Q5; (ii) $y_{k-1}y''_{k-1}$ is one of types Q3, Q4 and Q6; (iii) $y_{k-1}y''_{k-1}$ is of type Q0, and one of the intermediate query-response pairs on the *red* branch is not of type Q0.
Event Type3-1/-2/-3. There are three types of a Type3 event: (Type3-1) If the current π-query is the final π-query of a *red* branch of type (i).[2] (Type3-2) If the current π-query is the final π-query of a *red* branch of type (ii), as well as the most significant n bits of output being *distinguishable* from the uniform distribution $\mathcal{U}[0, 2^n - 1]$. (Type3-3) If the current π-query is the final π-query of a *red* branch of type(iii).

4.2.3 Event Type4: current s^{-1}-query is old (total 6 cases)

This event is shown in Figure 7(c). 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.

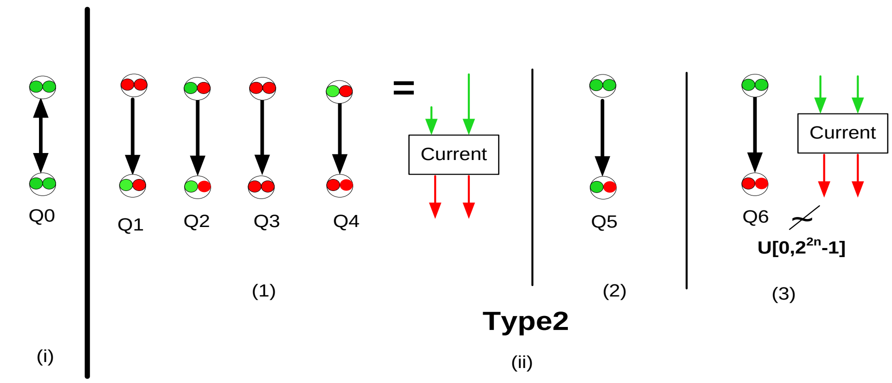
4.3 Proof of Theorem 4.1

With the help of the events described in Sections 4.1.2, 4.2.1, 4.2.2 and 4.2.3 we are equipped to prove Theorem 4.1. Recall we need to show two things:

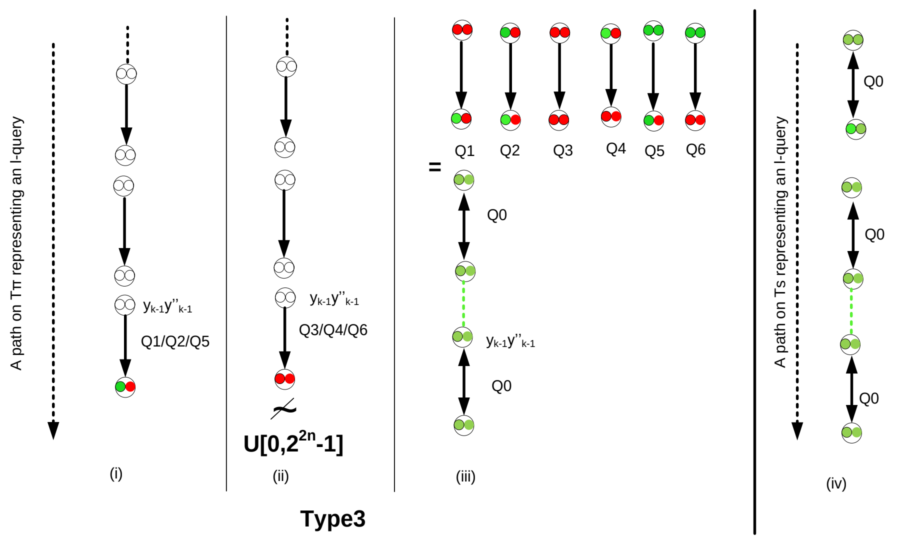
$$\left|\Pr\left[\mathcal{A}^{G_1} \Rightarrow 1\right] - \Pr\left[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\right]\right| \leq \Pr\left[\neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\right], \tag{1}$$

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

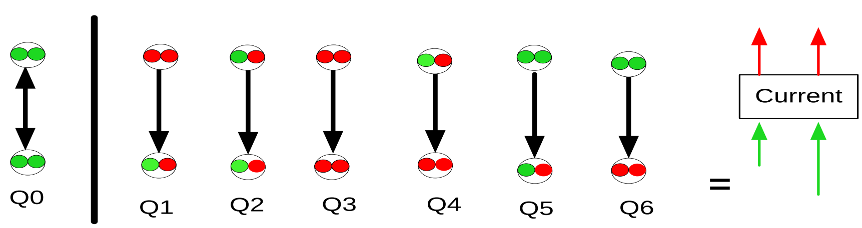
[2]Observe that this case implies a node-collision in T_π, since the $y_{k-1}y''_{k-1}$ is the final π-query for two distinct l-queries, the current M and also an old one. Therefore, if Type1 event did not occur in the previous rounds, this event is impossible in the current round.



(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 in Figure 9 of Appendix E.1. The corresponding Type2 events are also shown.



(b) Different types of a branch in the graph T_π. (i), (ii) and (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. 4.2.2. (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 G_1.

Figure 7: Pictorial description of Type2, Type3 and Type4 events of the game G_1 (Figure 5). *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.

The proof of (2) is straight-forward. To prove (1), we proceed in the following way. Observe

$$
\begin{aligned}
&\Big|\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1\big] - \Pr\big[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1\big]\Big| \\
&= \Big|\Big(\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-\frac{1}{2}}\big] - \Pr\big[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-\frac{1}{2}}\big]\Big) \cdot \Pr\big[\mathsf{GOOD}_{\sigma-\frac{1}{2}}\big] \\
&+ \Big(\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1 \mid \neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\big] - \Pr\big[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1 \mid \neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\big]\Big) \cdot \Pr\big[\neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\big]\Big|. \qquad (3)
\end{aligned}
$$

If we can show that

$$\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-\frac{1}{2}}\big] = \Pr\big[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1 \mid \mathsf{GOOD}_{\sigma-\frac{1}{2}}\big], \qquad (4)$$

then (3) reduces to (1), since $\Big|\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1 \mid \neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\big] - \Pr\big[\mathcal{A}^{\mathsf{RO},\mathsf{S},\mathsf{S}^{-1}} \Rightarrow 1 \mid \neg\mathsf{GOOD}_{\sigma-\frac{1}{2}}\big]\Big| \leq 1$. As a result, we focus on establishing (4).

Let V_1^i and V_2^i denote the views of the games G_1 and Game$(\mathsf{RO},\mathsf{S},\mathsf{S}^{-1})$ respectively, after i queries have been processed. To prove (4), 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 G_1. As the details are quite technical, we move the 17 cases to the Appendix D. The main idea is that if no bad events have occurred, then the graphs T_s are isomorphic, as indicated in the following lemma. From the isomorphism, the identical distribution of the views is an easy consequence. The proof of the lemma is given in Appendix D.1.

Lemma 4.2 (Graph Isomorphism Lemma) *Given GOOD_i and $V_1^i = V_2^i$, the graphs T_s for the games G_1 and Game$(\mathsf{RO},\mathsf{S},\mathsf{S}^{-1})$ are isomorphic after i rounds.*

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

We individually compute the probabilities of each of the events described in Sections 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 5.1 (Correction Factor) *Let ε be a negligible function in the security parameter $n > 0$. If the advantage of an indifferentiable adversary $\mathcal{A}$ for the games G_1 and Game(RO,S), limited by σ queries, is bounded by ε, then*

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

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

Proof. Since $\varepsilon < 1$ for all $n > 0$, $\Pr\big[\mathcal{A} \text{ sets } \mathsf{BAD} \text{ in } G_1\big] \leq \varepsilon \leq 1 - \frac{1}{C}$ 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 $\mathcal{O}(i)$ nodes. Assuming $i \leq 2^{n/2}$, from Figure 6 we obtain,

$$
\begin{aligned}
\Pr\Big[\text{Type1}_i \mid \mathsf{GOOD}_{i-1}\Big] &\leq 3i/(2^n - i), \\
&= \mathcal{O}\Big(\frac{i}{2^n}\Big), \qquad (5)
\end{aligned}
$$

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

Using the definition of Type2, Type3, Type4, and Type0 events in Section 4, it is straightforward to deduce:

$$\Pr\big[\text{Type2}_i \mid \mathsf{GOOD}_{i-1}\big] = \mathcal{O}\Big(\frac{i}{2^n}\Big),$$

$$\Pr\big[\text{Type3}_i \mid \mathsf{GOOD}_{i-1}\big] = \mathcal{O}\Big(\frac{1}{2^n}\Big),$$

$$\Pr\big[\text{Type4}_i \mid \mathsf{GOOD}_{i-1}\big] = \mathcal{O}\Big(\frac{i}{2^n}\Big),$$

and for $i \le 2^{n/2}$

$$\Pr\big[\text{Type0}_i \mid \mathsf{GOOD}_{i-1}\big] \le 1/(2^{2n}-i) = \mathcal{O}\Big(\frac{1}{2^{2n}}\Big).$$

Note that the constant C from Lemma 5.1 is absorbed by the O notation.

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

$$\begin{aligned}\Pr\big[\mathsf{BAD}_i \mid \mathsf{GOOD}_{i-1}\big] &\le \Pr\big[\text{Type0}_i \mid \mathsf{GOOD}_{i-1}\big] + \Pr\big[\text{Type1}_i \mid \mathsf{GOOD}_{i-1}\big] \\ &\quad + \Pr\big[\text{Type2}_i \mid \mathsf{GOOD}_{i-1}\big] + \Pr\big[\text{Type3}_i \mid \mathsf{GOOD}_{i-1}\big] + \Pr\big[\text{Type4}_i \mid \mathsf{GOOD}_{i-1}\big] \\ &= \mathcal{O}\Big(\frac{i}{2^n}\Big). \end{aligned} \tag{6}$$

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

$$\begin{aligned}\Big|\Pr\big[\mathcal{A}^{G_1} \Rightarrow 1\big] - \Pr\big[\mathcal{A}^{\text{Game}(\mathsf{RO},\,\mathsf{S},\,\mathsf{S}^{-1})} \Rightarrow 1\big]\Big| &\le \sum_{i=1}^{\sigma} \Pr\big[\mathsf{BAD}_i \mid \mathsf{GOOD}_{i-1}\big] \\ &\le \sum_{i=1}^{\sigma} \mathcal{O}\Big(\frac{i}{2^n}\Big) \\ &= \mathcal{O}\Big(\frac{\sigma^2}{2^n}\Big). \end{aligned} \tag{7}$$

Using (7) 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 G_1 and Game(RO, S, S^{-1}) (or between the games Game(JH, π, π^{-1}) and Game(RO, S, S^{-1}), since $G_1 \equiv$ Game(JH, π, π^{-1}) by Proposition 3.1). This yields the indifferentiability bound of $n/2$ bits for the JH mode.

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

We did not consider the Type0, Type2, Type3, and Type4 events, since, their probabilities are dominated by that of the Type1 events, for any efficient adversary. We found that choosing the values at which to place the 1st 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 Figure 10 of Appendix E.2. The data support the theoretically obtained bound of $\sigma = \Omega(2^{n/2})$ (see (7)). 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 for game G_1 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 out result that, for our definition of Type1 bad events and any fixed value of $R < 1$, $\sigma = \Theta(2^{n/2})$.

7 Conclusion and Open Problems

JH hash function is 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. Our experimental results strongly indicate that the bound could be further improved, and it is likely to be close 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 1st/2nd preimage 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. Also, the complexity for the simulator could be improved.

References

[1] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second preimage attacks on dithered hash functions. In Smart [17], pages 270–288.

[2] Elena Andreeva, Atul Luykx, and Bart Mennink. Provable Security of BLAKE with Non-Ideal Compression Function. *3rd SHA-3 Candidate Conference*, 2012.

[3] Elena Andreeva, Bart Mennink, and Bart Preneel. On the Indifferentiability of the Gröstl 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.

[4] 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.

[5] 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.

[6] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Smart [17], pages 181–197.

[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.

[8] Donghoon Chang, Mridul Nandi, and Moti Yung. Indifferentiability of the Hash Algorithm BLAKE. Cryptology ePrint Archive, Report 2011/623, 2011. `http://eprint.iacr.org/`.

[9] 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.

[10] Jonathan J. Hoch and Adi Shamir. Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2006.

[11] Antoine Joux. Multicollisions in Iterated Hash Functions: Application to Cascaded Constructions. In *CRYPTO 2004*, pages 306–316, 2004.

[12] 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.

[13] 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.

[14] Jooyoung Lee and Deukjo Hong. Collision resistance of the jh hash function. Cryptology ePrint Archive, Report 2011/019, 2011. `http://eprint.iacr.org/`.

[15] 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.

[16] Mridul Nandi and Douglas R. Stinson. Multicollision Attacks on Some Generalized Sequential Hash Functions. *IEEE Transactions on Information Theory*, 53(2):759–767, 2007.

[17] Nigel P. Smart, editor. *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*. Springer, 2008.

[18] Hongjun Wu. The JH Hash Function. The 1st SHA-3 Candidate Conference.

A Definitions

Definition A.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.*

B Equivalence of Games

Definition B.1 (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.*

C 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 T_s at the i-th 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 time to construct T_s dominates over others, the simulator time complexity is also $\mathcal{O}(\sigma^5)$.

D Proof of the Induction Step

We need 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. 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 G_1 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 4.2 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 G_1 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 Figure 8). The reader is invited to verify that all cases are considered.

Case 1: s-query, $|\mathcal{M}| = 0$, and Fresh:
Implication. The condition implies that O_1^{i+1} follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$ (Fig. 5), since a Type0 event did not occur in the $i+1$st round. Since the graphs T_s are isomorphic in both games G_1 and Game$(\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})$ by Lemma 4.2, $|\mathcal{M}| = 0$ for Game$(\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})$ (Fig. 3(b)). This implies that O_2^{i+1} follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$ (Fig. 3(b)).

Case 2: s-query, $|\mathcal{M}| = 0$, not Fresh, and type Q6:
Implication. The event $\mathsf{GOOD}_{i+\frac{1}{2}}$ implies that Type2 event did not occur for G_1 in the current $i+1$th round; therefore, since $|\mathcal{M}| = 0$, O_1^{i+1} follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$. As the graphs T_s of the games G_1 and Game$(\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})$ are isomorphic by Lemma 4.2, $|\mathcal{M}| = 0$ for Game$(\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1})$. This implies that $O_2^{i+1} = r$ follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$.

Case 3: s-query, $|\mathcal{M}| = 0$, not Fresh, and type Q5-1:
Implication. This case is impossible since $|\mathcal{M}| = 0$ and I^{i+1} being of type Q5-1 contradict each other.

Case 4: s-query, $|\mathcal{M}| = 0$, 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 G_1 in the current $i+1$st round. The given conditions create a Type2 event.

Case 5: s-query, $|\mathcal{M}| > 1$:
Implication. If $|\mathcal{M}| > 1$ 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 G_1 in the previous i rounds, and a node-collision in T_s is a Type1 event.

Case 6: s-query, $|\mathcal{M}| = 1$, and Fresh:
Implication. Since I^{i+1} is fresh, O_1^{i+1} follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$, since a Type0 event did not occur in the $i+1$st round. Now, for G_1, $M \in \mathcal{M}$ 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 $\mathcal{U}[0, 2^{2n} - 1]$.

Case 7: s-query, $|\mathcal{M}| = 1$, 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 G_1; therefore, O_1^{i+1} follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$. In G_1, $M \in \mathcal{M}$ 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 $\mathcal{U}[0, 2^{2n} - 1]$.

Case 8: s-query, $|\mathcal{M}| = 1$, 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 G_1; therefore, $O_1^{i+1}[n, 2n-1]$ follows the uniform distribution $\mathcal{U}[0, 2^n-1]$, and $O_1^{i+1}[0, n-1]$ is a fixed value. Now, for G_1, $M \in \mathcal{M}$ 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 D_l are identical in both games. Therefore, $O_2^{i+1}[0, n-1] = D_l[M] \oplus z$ (line 103 of Fig. 3(b)); also note that $O_2^{i+1}[n, 2n-1]$ follows the uniform distribution $\mathcal{U}[0, 2^n - 1]$. In conclusion, O_1^{i+1} and O_2^{i+1} are identically distributed.

Case 9: s-query, $|\mathcal{M}| = 0$, 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 or Q5-2.

Case 10: s^{-1}-query and Fresh:
Implication. The condition implies that O_1^{i+1} follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$, since a Type0 event did not occur in the current $i+1$st round. 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 $\mathcal{U}[0, 2^{2n}-1]$.

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 G_1, 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 $\mathcal{U}[0, 2^n - 1]$, since a Type0 event did not occur in the $i+1$st round. 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 $\mathcal{U}[0, 2^n - 1]$.

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 4.2. 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 (three types of a *red* branch are defined in Section 4.2.2). 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 $\mathcal{U}[0, 2^n - 1]$. Also observe, for G_1, 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 $\mathcal{U}[0, 2^n - 1]$.

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 4.1. ∎

D.1 Proof of Graph Isomorphism Lemma

PROOF. For each fresh π/π^{-1}-query, the graph T_π for game G_1 is augmented in one phase (see the subroutine PartialGraph of Figure 5). 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

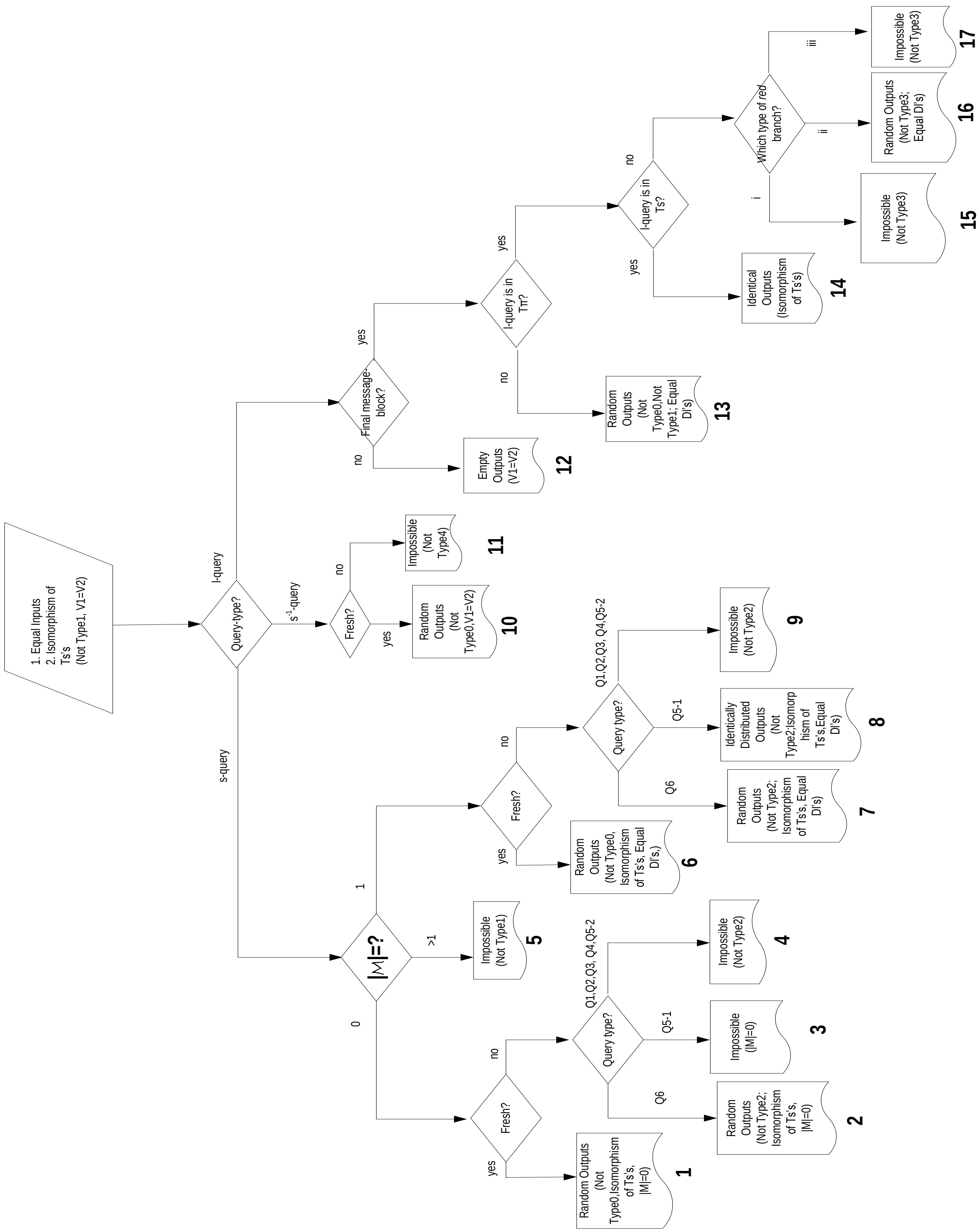


Figure 8: The decision tree for the proof of the induction step of Theorem 4.1. The conditions for the game G_1 are shown inside the diamonds of the decision tree. The text in each leaf-node shows the implications of the conditions to the outputs of games G_1 and Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$), while the reasons for such implications are described in brief inside the bracket.

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 G_1 contains all paths generated from all s/s^{-1}-queries and responses with root (IV, IV').
We note 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 G_1 and Game($\mathsf{RO}, \mathsf{S}, \mathsf{S}^{-1}$) are isomorphic after i rounds. □

E Graphics

E.1 Two subcases of Type2-2

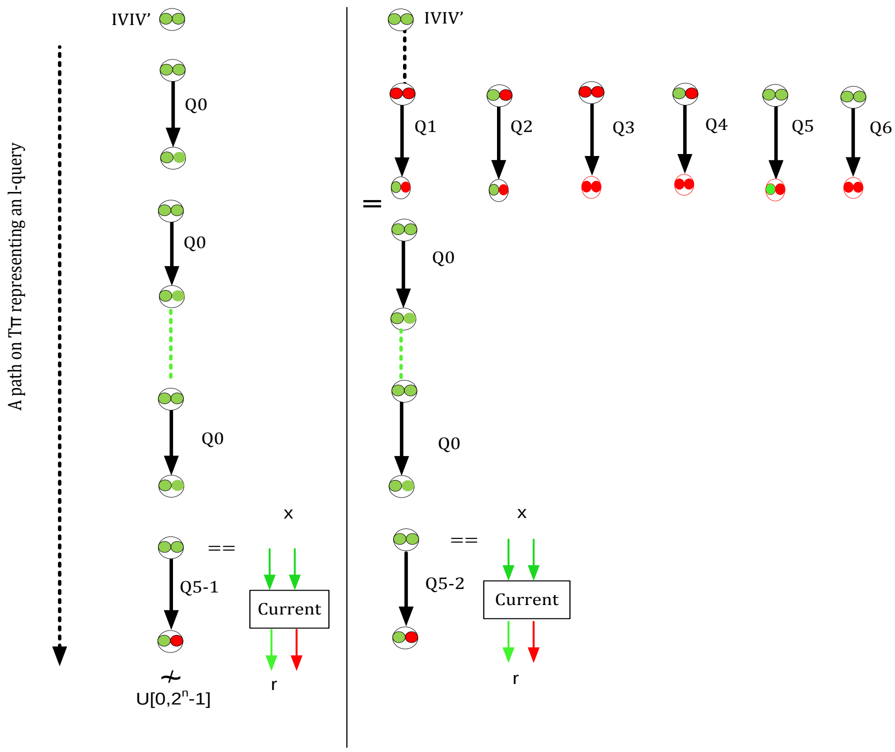


Figure 9: A query of type Q5-1 and Q5-2; the corresponding Type2-2 events are also shown.

E.2 Experimental Data

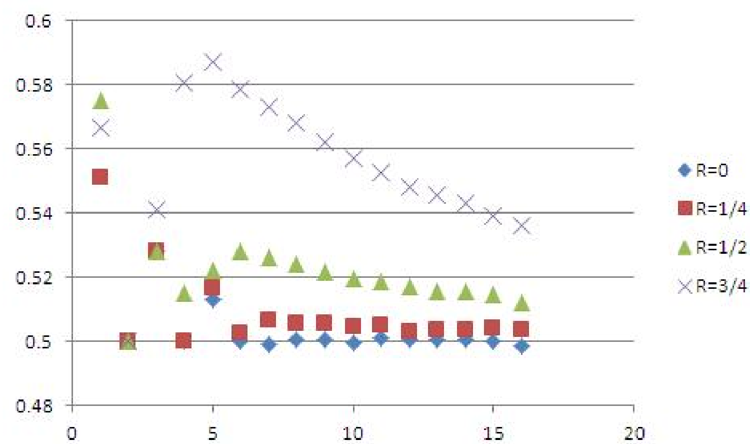


Figure 10: Plot of experimental data of value of n versus the normalized logarithm of σ, $\log_2(\sigma)/n$, for the game G_1 with various values of r, the proportion of reverse queries allowed.