Indifferentiability Security of the Fast Wide Pipe Hash: Breaking the Birthday Barrier

Dustin Moody∗ Souradyuti Paul† Daniel Smith-Tone‡

Abstract

A hash function secure in the indifferentiability framework (TCC 2004) is able to resist all meaningful generic attacks. Such hash functions also play a crucial role in establishing the security of protocols that use them as random functions.

To eliminate multi-collision type attacks on the Merkle-Damgård mode (Crypto 1989), Lucks proposed widening the size of the internal state of hash functions. More specifically, he suggested that hash functions $h : \{0, 1\}^* \to \{0, 1\}^n$ use underlying primitives of the form $C : \{0, 1\}^a \to \{0, 1\}^{2n}$ (Asiacrypt 2005). The Fast Wide Pipe (FWP) hash mode was introduced by Nandi and Paul at Indocrypt 2010, as a faster variant of Lucks’ Wide Pipe mode. Despite the higher speed, the proven indifferentiability bound of the FWP mode has so far been only up to the birthday barrier of $n/2$ bits. The main result of this paper is the improvement of the FWP bound to $2n/3$ bits (up to an additive constant).

The $2n/3$-bit bound for FWP comes with two important implications. Many popular hash modes use primitives with $a = 2n$, that is $C : \{0, 1\}^{2n} \to \{0, 1\}^{2n}$. For this important case, the FWP becomes the only mode to achieve indifferentiability security of more than $n/2$ bits; thus we solve a longstanding open problem. Secondly, among $n$-bit hash modes with $a > 2n$, the FWP mode has the highest rate among all modes which have beyond-birthday-barrier security.

To obtain the bound of $2n/3$ bits, we follow the usual technique of constructing games with simulators, with certain BAD events to distinguish between the games. However, we introduce some novel ideas. In designing the BAD events, we used multi-collisions in addition to collisions. We also allowed the query-response graphs, maintained by the simulators, to grow for two phases every iteration, rather than just one phase. Finally, our carefully chosen set of sixteen BAD events establish an isomorphism of simulator graphs, from which the $2n/3$-bit bound follows.

We also provide evidence that extending the bound beyond $2n/3$ bits may be possible if we allow the simulator-graph to grow for three (or more) phases every iteration. Another noteworthy feature of our proof – that may be of independent interest – is that we work with only three games rather than a long sequence games.

Keywords: Indifferentiability, birthday barrier, Fast wide pipe.

∗NIST, Computer Security Div. USA, dustin.moody@nist.gov
†Univ. of Waterloo, Math Dept., Canada and K. U. Leuven, Belgium, souradyuti.paul@uwaterloo.ca
‡NIST, USA, and Univ. of Louisville, Math Dept., KY, USA, daniel.smith@nist.gov
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> Introduction</td>
<td>5</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>5</td>
</tr>
<tr>
<td>1.2 Our contribution</td>
<td>7</td>
</tr>
<tr>
<td><strong>2</strong> Preliminaries</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Notation and convention</td>
<td>9</td>
</tr>
<tr>
<td>2.2 Description of FWP mode</td>
<td>10</td>
</tr>
<tr>
<td>2.3 Indifferentiability framework</td>
<td>10</td>
</tr>
<tr>
<td><strong>3</strong> Main Theorem: Beyond-birthday-barrier Security of FWP</td>
<td>11</td>
</tr>
<tr>
<td>3.1 Proof of Theorem 3.1 outline</td>
<td>12</td>
</tr>
<tr>
<td>3.2 Organization</td>
<td>13</td>
</tr>
<tr>
<td><strong>4</strong> Data Structures</td>
<td>13</td>
</tr>
<tr>
<td>4.1 Objects used in pseudocode</td>
<td>13</td>
</tr>
<tr>
<td>4.1.1 Oracles</td>
<td>13</td>
</tr>
<tr>
<td>4.1.2 Global and local variables</td>
<td>13</td>
</tr>
<tr>
<td>4.1.3 Query and round: definitions</td>
<td>13</td>
</tr>
<tr>
<td>4.2 Graph theoretic objects used in proof of main theorem</td>
<td>14</td>
</tr>
<tr>
<td>4.2.1 Reconstructible message</td>
<td>14</td>
</tr>
<tr>
<td>4.2.2 (Full) Reconstruction graph</td>
<td>15</td>
</tr>
<tr>
<td>4.2.3 View</td>
<td>15</td>
</tr>
<tr>
<td><strong>5</strong> Main system $G_0$</td>
<td>17</td>
</tr>
<tr>
<td><strong>6</strong> Main system $G_2$</td>
<td>17</td>
</tr>
<tr>
<td>6.1 Intuition for simulator $S$</td>
<td>17</td>
</tr>
<tr>
<td>6.2 Detailed description of simulator $S$</td>
<td>18</td>
</tr>
<tr>
<td><strong>7</strong> Intermediate system $G_1$</td>
<td>19</td>
</tr>
<tr>
<td>7.1 Motivation</td>
<td>19</td>
</tr>
<tr>
<td>7.2 Detailed description of $G_1$</td>
<td>20</td>
</tr>
<tr>
<td><strong>8</strong> First Part of Main Theorem: Proof of (2)</td>
<td>22</td>
</tr>
<tr>
<td><strong>9</strong> Type0, 1, 2, and 3, of System $G_1$</td>
<td>22</td>
</tr>
<tr>
<td>9.1 Motivation</td>
<td>22</td>
</tr>
<tr>
<td>9.2 Classifying elements of $D_{ro}$, branches of $T_{ro}$, and ro-queries</td>
<td>22</td>
</tr>
<tr>
<td>9.2.1 Elements of $D_{ro}$: six types</td>
<td>25</td>
</tr>
</tbody>
</table>
9.2.2 Branches of $T_{ro}$: four types ........................................ 25
9.2.3 The $ro$-queries: seven types ........................................ 25
9.3 Definition: Type0 and Type1 on fresh queries ....................... 26
  9.3.1 Intuition ............................................................. 26
  9.3.2 Type0 event: collision in outputs of $ro$ .......................... 28
  9.3.3 Type1 event: collision in $T_{ro}$ .................................. 28
9.4 Type2 and Type3 on old queries ....................................... 29
  9.4.1 Intuition ............................................................. 29
  9.4.2 Type2 ................................................................. 29
  9.4.3 Type3 ................................................................. 30

10 Second Part of Main Theorem: Proof of (3) ............................ 30
  10.1 Definitions: GOOD$_i$ and BAD$_i$ ................................. 30
  10.2 Proof of (3) ............................................................ 31

11 A Few Combinatorial Results ........................................... 31

12 Third (or Final) Part of Main Theorem: Proof of (4) ................. 35
  12.1 Estimating probability of Type0$_i$ .................................. 35
  12.2 Estimating probability of Type1$_i$ .................................. 36
    12.2.1 Computing probability of Type1-a$_i$ ............................ 36
    12.2.2 Computing probability of Type1-b$_i$ ............................ 37
    12.2.3 Computing probability of Type1-c$_i$ ............................ 37
    12.2.4 Computing probability of Type1-d$_i$ ............................ 37
    12.2.5 Computing probability of Type1-e$_i$ ............................ 38
    12.2.6 Computing probability of Type1-f$_i$ ............................ 39
    12.2.7 Final summation .................................................. 39
  12.3 Estimating probability of Type2$_i$ .................................. 39
    12.3.1 Estimating probability of Type2-a$_i$ ............................ 40
    12.3.2 Estimating probability of Type2-b$_i$ ............................ 40
    12.3.3 Estimating probability of Type2-c$_i$ ............................ 40
    12.3.4 Estimating probability of Type2-d$_i$ ............................ 40
    12.3.5 Estimating probability of Type2-e$_i$ ............................ 41
    12.3.6 Estimating probability of Type2-f$_i$ ............................ 41
    12.3.7 Final summation .................................................. 41
  12.4 Estimating probability of Type3$_i$ .................................. 41
    12.4.1 Estimating probability of Type3-a$_i$ ............................ 41
    12.4.2 Estimating probability of Type3-b$_i$ ............................ 42
    12.4.3 Estimating probability of Type3-c$_i$ ............................ 42
    12.4.4 Final summation .................................................. 42
  12.5 Final step ............................................................. 42
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13 Experimental Results: The Bound Improves Towards $n$ bits</td>
<td>42</td>
</tr>
<tr>
<td>14 Conclusion and Open Problems</td>
<td>43</td>
</tr>
<tr>
<td>A Definitions</td>
<td>49</td>
</tr>
<tr>
<td>B WP and FWP</td>
<td>50</td>
</tr>
<tr>
<td>C Time costs of FullGraph and simulator S</td>
<td>50</td>
</tr>
<tr>
<td>D Six Types of ro-query-response Pairs</td>
<td>50</td>
</tr>
<tr>
<td>E Proof of [8]</td>
<td>51</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Motivation

Iterative hash functions are usually composed of two parts: (1) a basic primitive $C$ with finite domain and range, and (2) an iterative mode of operation $H$ to extend the domain of $C$. In studying the security of a hash function, both the security of the primitive $C$, as well as the security of the mode of operation $H$ need to be examined separately, since the hash function can be attacked by breaking either $C$ or the mode $H$. Our focus is on the security of the hash mode $H$. Throughout the paper we would assume that the primitive $C$ is an ideal object, that is, it does not exhibit any non-trivial weakness.

**Generic attacks on hash modes.** In a generic attack, an adversary attempts to break a property of a hash mode assuming the underlying primitive is an ideal object, such as a random oracle, an ideal permutation, or an ideal cipher. For example, suppose the hash function $h : \{0,1\}^* \rightarrow \{0,1\}^n$, for a given input $M \in \{0,1\}^*$, invokes a random oracle $RO : \{0,1\}^a \rightarrow \{0,1\}^b$, one or multiple times, to compute $h(M)$. Informally, a generic attack breaks a property of the hash function $h$ (e.g. 1st/2nd pre-image, collision resistance) utilizing less resources than would be required to break the same property of the random oracle $RO : \{0,1\}^* \rightarrow \{0,1\}^n$.

Generic attacks against hash modes are abundant in the literature. See, for example, Joux’s multi-collision attack [23], Kelsey-Schneier expandable message attack [25], or Kelsey-Kohno herding attack [24, 11], among others [1, 9, 22, 32].

**Indifferentiability security.** The indifferentiability security framework was introduced by Maurer et al. [27] in 2004, and was first applied to analyze hash modes of operation by Coron et al. [17] 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. Indifferentiability attacks include more attacks [3, 9, 15] than just those with known practical significance. Thus in some sense, an indifferentiable hash function can be viewed as eliminating potential future attacks. We note that the security of many cryptographic protocols (e.g. RSA-OAEP [34], RSA-PSS [10]) relies on the indifferentiability security of the underlying hash functions that the protocols use as random oracles. In such a case, the security of the hash functions against specialized attacks – such as collision, 1st/2nd pre-image attacks – is inadequate to guarantee the security of the overlying protocol. As a result, most new proposals for hash modes of operation include indifferentiability proofs of security. We note that some limitations of the indifferentiability framework have recently been discovered in [19] and [33]. These limitations, nevertheless, do not affect the proven security bounds of hash functions based on ideal objects.
Table 1: Indifferentiability security bounds (upper and lower) for several wide-pipe hash modes, where the primitive output is $2n$-bit (the hash size is $n$-bit). The primitives ro, ic and ip are shorthand for random oracle, ideal cipher, and ideal permutation. The letter $t$ denotes a positive integer. The $\epsilon$ is a small fraction due to the preimage attack on JH presented in [9].

Advantages of primitives with $2n$-bit output. Many practical iterative hash modes which use primitives with $n$-bit output have been shown to come under multi-collision attacks with $O(2^{n/2})$ queries [23]. Therefore, their indifferentiability security bounds cannot be extended beyond $n/2$ bits. A few well known examples include Merkle-Damgård [18, 29], HAIFA [10], EMD [8], and MDP [21].

As a result, to design a practical hash mode with indifferentiability security more than $n/2$ bits, it seems necessary to use primitives with $2n$ bits of output (or more) [26]. Examples of hash modes with $2n$ bits of output include: Wide Pipe-MD [26], FWP [31], JH [36], Grøstl [20], Sponge [8], Shabal [13], and Parazoa [5], to name a few.

The rate of a hash function. In any iterative hash function $h: \{0,1\}^* \rightarrow \{0,1\}^n$, the input to the underlying primitive $C: \{0,1\}^a \rightarrow \{0,1\}^b$ is composed of an $\ell$-bit message block and an $(a–\ell)$-bit chaining value from the previous iteration. It is easy to see that, for a fixed $a$ and $b$, the higher the value of $\ell$, the faster the hash computation. To formalize this property, we define the rate of a hash function as the ratio $\ell/a$, where $\ell$ is the (average) length of a message block\footnote{The average length of a message block is computed as the length of a padded message divided by the total number of invocations of the primitive(s). This addresses the issue when the same message block is used in multiple invocations of the primitive (e.g. Grøstl).}.
The challenge. Based on the previous discussion, the challenge is to design an $n$-bit hash function using primitives of the form $C : \{0,1\}^n \to \{0,1\}^{2n}$ which maximize both the rate and the indifferentiability security bound. The rates and security bounds of several hash modes with primitive output $2n$-bit have been tabulated in Table 1.

1.2 Our contribution

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{FWP_mode.png}
\caption{Diagram of the FWP mode. All unlabeled wires are $n$ bits each. The shaded region is viewed as compression function.}
\end{figure}

The FWP mode (2010). The Fast Wide Pipe (FWP) hash mode was proposed by Nandi and Paul in 2010 [31], as a faster variant of the Wide Pipe (WP) mode [26] (see Figure 1). The key idea used in FWP is that only half of the chaining value – instead of the full chaining value – is used as input into the primitive $C$, while the other half is XOR-ed into the output. Thus, FWP increases the rate by allowing message blocks larger than those used by the WP mode. See Figure 1 of Appendix 1 which compares the WP mode with the FWP mode. As is the case with all other high rate hash modes with primitive output $2n$ bits, the indifferentiability bound of FWP has so far remained only up to $n/2$ bits [31].

The difficulty of breaking the birthday-barrier. It is apparent from Table 1 that extending both the rate and the indifferentiability bound of a hash mode is a challenging task. Note that the Shabal and Wide Pipe modes have $n$-bit security bounds, but their rates are quite low. For the Sponge function – though having a high rate of 0.5 – the security bound is $n/2$-bit, which cannot be improved further, since there is a preimage attack with work approximately $2^{n/2}$ [7].

Several other designs (JH (2007), Grøstl (2007), FWP (2010) and the Parazoa family (2011)) have shown promise. Each achieves the high rate of 0.5 (if $a = 3n$ then the FWP can achieve a rate of 0.67), and their indifferentiability bounds can potentially be improved beyond the birthday barrier. Despite several attempts, so far none of them has been shown to have the beyond-birthday-barrier security. See [9, 30, 31] and [4].
In all of the previous attempts, the basic approach for proving indifferentiability security has been more or less the same. First, a suitable compression function is constructed around the primitive $C$. See, for example, the compression function contained in the shaded part of Figure 1. Then a set of events are identified that primarily consider collisions on $n$ (out of $2n$) bits of output of the compression function. These events are typically called BAD events, and are used to differentiate between pairs of games. Lastly, using some sophisticated combinatorial tricks and counting techniques, the total probability of the BAD events are computed by summing them across all rounds and messages.

The main deficiency in the above approach is two-pronged. The probability of the BAD events is dominated by the probability of $n$-bit collisions of the compression function output, which is too high. Considering $n$-bit collisions can only give a bound of $n/2$ bits. On the other hand, if we consider collisions on the $2n$ bits of the compression function output, then after $i$ queries we end up having $O(i^2)$ reconstructible messages (defined later), and this will again lead to a bound only of $n/2$ bits.

Evidently to go beyond the $n/2$-bit security bound, we need to identify BAD events whose probability of occurrence will be as low as the probability of random $2n$-bit collisions (rather than of $n$-bit collisions). We also need that the number of all reconstructible messages after $i$ queries will be linear in $i$, rather than quadratic.

**The main result.** Our main results is the improvement of the indifferentiability security bound for the FWP mode from $n/2$ to $2n/3$ bits (up to a constant factor). The FWP mode is based on a primitive of the form $C : \{0,1\}^a \rightarrow \{0,1\}^{2n}$, and our $2n/3$-bit security bound is valid for all $a \geq 2n$. We make two important observations.

Let $\overline{H}(a,b,r)$ denote the class of all rate $r$, $n$-bit hash functions with a primitive of the form $C : \{0,1\}^a \rightarrow \{0,1\}^b$. The FWP mode is the only known hash mode with the beyond-birthday-barrier security in the important class $\overline{H}(2n,2n,0.5)$. Compare FWP with JH, Grostl, Sponge, and the Parazoa family. This essentially settles a longstanding open problem.

Let $\overline{H}(a,b)$ denote the class of all $n$-bit hash functions with the beyond-birthday-barrier security based upon primitives of the form $C : \{0,1\}^a \rightarrow \{0,1\}^b$. In the important class $\overline{H}(a,2n)$, the FWP mode achieves the highest rate for all $a \geq 2n$. Compare FWP with chop-MD, Shabal and BLAKE.

**The tools.** The first new idea used to break the $n/2$-bit bound is in using certain 3-multi-collisions on $n$ bits – in addition to collisions on $2n$ bits – as potential BAD events. We show that the 3-multi-collisions on $n$ bits as well as the $2n$-bit collisions both occur with low probabilities. In particular, we carefully design a set of sixteen bad events defined on the query, primitive and compression function outputs.

The second trick is to split the above $2n$-bit collisions into two distinct $n$-bit collisions occurring in two different phases at the time of updating the simulator’s graph (technically we shall call it a reconstruction graph). Updating the reconstruction graph – whose
branches represent messages built from the queries and their responses – in a sequence of two phases, rather than just one, is crucial to the results in this work.

Using these two techniques, we are able to overcome the aforementioned obstacles in moving beyond the $n/2$-bit bound. In particular, the absence of the BAD events allows the reconstruction graph to grow for a maximum of two phases every round, while restricting the number of newly added nodes in the graphs to a constant number. As a result, we have $O(i)$ reconstructible messages after $i$ rounds, and at the same time, the probability of occurrence of the BAD events remains low. Once these important requirements are fulfilled, the final step is to prove an isomorphism of graphs, which then directly implies the claimed bound.

Another feature of our work, which may be of independent interest, is that the proof of our main theorem 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 – in our opinion – makes third-party verification of the proof a great deal easier. In addition, our proof technique can likely be used to improve the security analysis of other modes.

**Beyond the 2n/3-bit barrier** It seems likely that the 2$n$/3-bit bound of FWP could be further improved, if we switched from two phases to a three phase framework. We experimented with a slightly different set of BAD events in the three (or more) phase framework. The results provide ample evidence that the indifferentiability bound for the FWP mode can be stretched closer to $n$ bits. We leave it as an open problem to complete the theoretical analysis required for such an improved bound.

**Warning.** As is necessary for any analysis of cryptosystems based on ideal objects, we caution the reader that the security guarantee of 2$n$/3 bits for any practical hash function, based on the FWP mode, can be achieved as long as the underlying concrete primitive is free from all structural weaknesses, about which the paper makes no claims.

## 2 Preliminaries

### 2.1 Notation and convention

Throughout the paper we let $n$ be a fixed integer. While representing a bit-string, we follow the convention of low-bit first (or little-endian bit ordering). For concatenation of strings, we use $a \| b$, or just $ab$ if the meaning is clear. The symbol $\langle n \rangle_m$ denotes the $m$-bit encoding of $n$. The symbol $|x|$ denotes the bit-length of the bit-string $x$, or sometimes the size of the set $x$. Let $x_{\text{parse}} = x_1 x_2 \cdots x_k$ means parsing $x$ into $x_1, x_2, \ldots, x_k$ such that $|x_1| = |x_2| = \cdots = |x_{k-1}| = n$ and $|x_k| = |x| - |x_1 x_2 \cdots x_{k-1}|$. Let $\text{Dom}(T) = \{ i \mid T[i] \neq \bot \}$ and $\text{Rng}(T) = \{ T[i] \mid T[i] \neq \bot \}$. 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, $U[0,N]$ is the uniform distribution over the integers between 0 and $N$. The symbol $r \overset{S}{\leftarrow} A$ denotes the operation of assigning $r$ with a value sampled uniformly from the set $A$. The symbol $\lambda$ denotes the empty string.

### 2.2 Description of FWP mode

We now revisit the description of the FWP mode of [31]. Suppose that $\ell$ and $n$ are integers such that $\ell \geq n \geq 1$. Let $C : \{0,1\}^{\ell+n} \rightarrow \{0,1\}^{2n}$ be a cryptographic primitive which we use to build the hash function FWP: $\{0,1\}^* \rightarrow \{0,1\}^n$. The diagram and the description of the FWP transform are given in Figures 1 and 3(a) where $C$ is the random oracle $ro$.

**Padding rule.** The notation $\text{pad}(M) = m_1 \cdots m_k$ is interpreted as follows: Using an injective function $\text{pad}: \{0,1\}^* \rightarrow \bigcup_{i \geq 1} \{0,1\}^{(i+1)\ell-n}$, the message $M$ is mapped into a string $\text{pad}(M) = m_1 \cdots m_k$ such that $k = \left\lceil \frac{|M|}{\ell} \right\rceil + 1$, where $|m_i| = \ell$ for $1 \leq i \leq k-1$, and $|m_k| = \ell - n$. In addition to the injectivity of $\text{pad}(\cdot)$, we will also require another property that there exists a function $\text{dePad}(\cdot)$ that can efficiently compute $M$, given $\text{pad}(M)$. Formally, the function $\text{dePad}: \bigcup_{i \geq 1} \{0,1\}^{(i+1)\ell-n} \rightarrow \{\perp\} \cup \{0,1\}^*$ computes $\text{dePad}(\text{pad}(M)) = M$, for all $M \in \{0,1\}^*$, and otherwise $\text{dePad}(\cdot)$ returns $\perp$. We note that the padding rules of all practical hash functions have the above properties. An example of $\text{pad}(\cdot)$, satisfying the above properties, is $\text{pad}(M) = M || 1 || 0^t$, where $t$ is the least non-negative integer such that $t = (|M| - 1 \mod \ell) + \ell - n$. Another example can be found in [31].

### 2.3 Indifferentiability framework

**Definition 2.1 (Indifferentiability framework)** [17] An interactive Turing machine (ITM) $T$ with oracle access to an ideal primitive $F$ is said to be $(t_A, t_S, \sigma, \varepsilon)$-indifferentiable from an ideal primitive $G$ if there exists a simulator $S$ such that, for any distinguisher $A$, the following equation is satisfied:

$$
\text{Adv}^{T,F}_{G,S}(A) \overset{\text{def}}{=} |\Pr[A^{T,F} = 1] - \Pr[A^{G,S} = 1]| \leq \varepsilon.
$$

The simulator $S$ is an ITM which has oracle access to $G$ and runs in time at most $t_S$. The distinguisher $A$ runs in time at most $t_A$. The number of queries used by $A$ is at most $\sigma$. Here $\varepsilon$ is a negligible function in the security parameter of $T$. See Figure 2(a) for a pictorial representation. $\text{Adv}^{T,F}_{G,S}(A)$ denotes the advantage of adversary $A$ in distinguishing $(T, F)$ from $(G, S)$.

The significance of the framework is as follows. Suppose, an ideal primitive $G$ is indifferentiable from an algorithm $T$ based on another ideal primitive $F$. In such a case, any cryptographic system $P$ based on $G$ is as secure as $P$ based on $T^F$ (i.e., $G$ replaces $T^F$ in
Figure 2: (a) Indifferentiability framework formalized in Definition 2.1. (b) Schematic diagrams of the security games – described in Section 3 – used in the indifferentiability framework for FWP. The arrows show the directions in which the queries are submitted.

\[ P \). For a more detailed explanation, we refer the reader to [28]. Some limitations of the indifferentiability framework have recently been discovered in [19] and [33]. They offer a deep insight into the framework; nevertheless, the observations are not known to affect the security of the indifferentiable hash functions in any meaningful way.

An oracle, a system, and a game. An oracle is an algorithm (accessed by another oracle or algorithm) which, given an input as an appropriately defined query, responds with an output. For example, in Figure 2(a) T, F, G and S are oracles. A system is a set of oracles (e.g. System 1 = \( (T, F) \), System 2 = \( (G, S) \) in Figure 2(a)). A game is the interaction of a system with an adversary. We refrain from providing a formal definition of a game, since such formalization will not be necessary in our analysis.

3 Main Theorem: Beyond-birthday-barrier Security of FWP

Let RO: \{0,1\}^* \rightarrow \{0,1\}^n and ro: \{0,1\}^{\ell+n} \rightarrow \{0,1\}^{2n} be two random oracles (see Appendix A for a definition).

Our indifferentiability framework uses three systems G0 = (FWP, ro), G1 = (FWP1, S1), and G2 = (RO, S) (see Figure 2(b)). The correspondence between the entities of Figures 2(a) and 2(b) are as follows: G = RO, T = FWP and F = ro. The description of FWP1, S, and S1 will be provided in Sections 5, 6, and 7.

Now we state our main theorem using Definition 2.1

**Theorem 3.1 (Main Theorem)** The hash function FWP^{ro} is \( (t_A, t_S, \sigma, \varepsilon) \)-indifferentiable from RO, where \( t_A = \infty \), \( t_S = \mathcal{O} \sigma^5 \), and \( \sigma = K2^{2n/3} \), where \( K \) is a fixed constant derived...
from $\varepsilon$.

In the next few sections, we will prove Theorem 3.1 by breaking it into several components. First, we briefly describe what the theorem means: it says that no adversary with unbounded running time can mount a non-trivial generic attack on the hash function $\text{FWP}^{\text{ro}}$ using at most $K2^{2n/3}$ queries. The parameter $K$ is an increasing function in $\varepsilon$, and is constant for all $n > 0$, for a fixed $\varepsilon$. To reduce the notation complexity, we compute the indifferentiability bound assuming $\varepsilon = 1/2$, for which, we shall derive $K = 1/\sqrt[206]{2}$.

### 3.1 Proof of Theorem 3.1: outline

Proof of Theorem 3.1 consists of the following two components (see Definition 2.1): Firstly, we need to construct a simulator $S$ with the worst-case running time $t_S = O(\sigma^3)$. This is done in Section 7. Secondly, we need to show that, for any adversary $A$, with unbounded running time,

$$\Pr[A^{G0} \Rightarrow 1] - \Pr[A^{G2} \Rightarrow 1] \leq \frac{103\sigma^3}{2^{2n}},$$

assuming $\varepsilon = 1/2$. We will prove (1) by, again, splitting it into several parts.

- In Section 8, we show that

$$\Pr[A^{G0} \Rightarrow 1] = \Pr[A^{G1} \Rightarrow 1].$$

- In Section 9, we will appropriately define a set of events $\text{BAD}_i$ and $\text{GOOD}_i$ for $G1$, and establish in Section 10 that

$$\Pr[A^{G1} \Rightarrow 1] - \Pr[A^{G2} \Rightarrow 1] \leq \sum_{i=1}^{\sigma} \Pr[\text{BAD}_i \mid \text{GOOD}_{i-1}].$$

- Finally, we show in Section 12 that

$$\sum_{i=1}^{\sigma} \Pr[\text{BAD}_i \mid \text{GOOD}_{i-1}] \leq \frac{103\sigma^3}{2^{2n}},$$

assuming

$$\sum_{i=1}^{\sigma} \Pr[\text{BAD}_i \mid \text{GOOD}_{i-1}] \leq \varepsilon = 0.5.$$ 

Note that (1) is easily established by combining (2), (3) and (4).

\footnote{Setting $\frac{103\sigma^3}{2^{2n}} \leq \varepsilon = 1/2$, we get $\sigma \leq \frac{1}{\sqrt[204]{2}} 2^{2n/3}$. Therefore, $K = 1/\sqrt[206]{2}$.}
3.2 Organization

In Sections 4, 5, 6, and 7 we describe the systems G0, G1 and G2. Using them, in Section 8 we prove (2). In Section 9, we define certain ‘bad’ events in system G1, using which in Section 10 we prove (3). In Section 11, some combinatorial results are established to finally prove (4) in Section 12.

In Section 13, we provide experimental evidence as to why the FWP should have a better security bound than the proven $2n/3$-bit. In the final section Section 14, we conclude, and pose some open questions.

4 Data Structures

The systems G0, G1, and G2 have been mentioned in Section 3 (see schematic diagram in Figure 2(b)). The pseudocode of them is given in Figures 3(a), 5, and 3(b). In this section we describe several data structures used by these systems.

4.1 Objects used in pseudocode

4.1.1 Oracles

The main component of a system is the set of oracles that receive queries from the adversary. In Figure 2(b) any algorithm that receives a query is an oracle. Note that, except the adversary $\mathcal{A}$, each rectangle denotes an oracle.

The systems use a total of 6 oracles. The oracles FWP, FWP1, and RO are mappings from $\{0,1\}^*$ to $\{0,1\}^n$. The oracles $S$, ro, and $S_1$ are mappings from $\{0,1\}^{t+n}$ to $\{0,1\}^{2n}$. Instruction-by-instruction description of these oracles and the used subroutines are provided in the subsequent sections.

4.1.2 Global and local variables

The oracles described above will use several global and local variables. The local variables are re-initialized every new invocation of the system, while the global data structures maintain their states across queries. The tables $D_t$, $D_s$ and $D_{ro}$ are global variables initialized with ⊥. The graphs $T_{ro}$ and $T_s$ are also global variables which initially contain only the root node $(IV, IV')$. Other than them, all other variables are local, and they are initialized with ⊥.

4.1.3 Query and round: definitions

In Figure 2(b) an arrow denotes a query. The submitter and receiver algorithms of a query are denoted by the rectangles attached to the head and the tail of the arrow.
Long query. Any query submitted to FWP, FWP1, or RO is a long query. A long query and its response are stored in the table $D_l$.

Short query. Queries submitted to $S$, $S_1$ are $s$-queries. The $s$-queries and their responses are stored in table $D_s$. Similarly, queries submitted to ro are ro-queries; these queries and their responses are stored in table $D_{ro}$. Each of the above queries is classified as short query. Note that, for $G0$, $D_s = \emptyset$; for $G1$, $D_{ro} \supseteq D_s$; and, for $G2$, $D_{ro} = \emptyset$.

Fresh and old queries. The current short query can also be of two disjoint types: (1) an old query, which is already present in the relevant database (e.g. for G1, when an adversary submits an $s$-query which is an intermediate ro-query of a previously submitted long query); or (2) a fresh query, which is so far not present in the relevant database.

Message block. In order to compare the time complexities of the oracles FWP, FWP1 and RO on a uniform scale, we recall the notion of a message block. A long query $M$ – irrespective of the oracle – is assumed to be a sequence of $k$ message blocks $m_1, m_2, \ldots, m_k$, where $\text{pad}(M) = m_1m_2 \cdots m_k$. Note that, for FWP and FWP1, every message block $m_i$ corresponds to a ro-query $x||m_i$ for some bit-string $x$. However, it is not known how the RO processes the message blocks of a long query $M$. We assume that the RO processes the message blocks sequentially, and that the time taken to process a message block is equal for all FWP, FWP1 and RO.

Round (and query). The time interval to process a short query or a message block is defined as a round. We assume that each round takes an equal amount of time. To simplify the analysis, henceforth, unless otherwise specified, a query would mean either a short query or a message block.

Rules of the game. An adversary never re-submits an identical long or $s$-query.

4.2 Graph theoretic objects used in proof of main theorem

In addition to objects defined in the section above, we will use the following notions for a rigorous mathematical analysis of our results.

Suppose, ro: $\{0, 1\}^{\ell+n} \rightarrow \{0, 1\}^{2n}$ is a random oracle, and $D$ is a finite set of pairs of the form $(x, \text{ro}(x))$.

4.2.1 Reconstructible message

From the high level, $M$ is a reconstructible message for the set $D$, if $D$ contains all the ro-queries and responses $(x, \text{ro}(x))$, required to compute $\text{FWP}^{\text{ro}}(M)$.
More formally, \( M \) is a reconstructible message for \( D \) if, for all \( 0 \leq i \leq k - 2 \), \((y_i, m_{i+1}, \text{ro}(y_i, m_{i+1})) \in D \), and \((y_{k-i-1}y_{k-i}m_k, \text{ro}(y_{k-i+1}y_{k-i}m_k)) \in D \), where \( \text{pad}(M) = m_1m_2\cdots m_k \) and \( y_iy_{i+1} = \text{ro}(y_i, m_{i+1}) + (y_i'\|0) \) for all \( 0 \leq i \leq k - 2 \). All \( y_i \)'s and \( y_i' \)'s are \( n \) bits each; \( m_i \)'s are \( \ell \) bits each, except \( m_k \), which is an \((\ell - n)\)-bit string.

4.2.2 (Full) Reconstruction graph

To put it loosely, a reconstruction graph stores reconstructible messages on its branches. A full reconstruction graph stores all reconstructible messages. We now define it formally, using the notion of a weighted digraph.

A weighted digraph \( T = (V, E) \) is defined by the set of nodes \( V \), and the set of weighted edges \( E \). A weighted edge \((v, w, v') \in E \) is an ordered triple, such that \( v, v' \in V \), and \( w \) is the weight of the ordered pair \((v, v')\).

**Definition 4.1 (Reconstruction graph)** Suppose a weighted digraph \( T = (V, E) \) is such that \( V \) is a set of \( 2n \)-bit strings, and, for all \((a, b, c) \in E \), the weight \( b \) is an \( \ell \)-bit string.

The graph \( T \) is called a reconstruction graph for \( D \) if, for every \((y_1y'_1, m_2, y_2y'_2) \in E \), the following equation holds: \( y_2y'_2 = \text{ro}(y_1m_2) + (y_1'\|0) \) \((y_1, y'_1, y_2, \text{ and } y'_2 \) are \( n \) bits each, and \( m_2 \) is \( \ell \)-bit), where \((y_1m_2, \text{ro}(y_1m_2)) \in D \). (An example of reconstruction graph is given in Figure 4, which will be discussed in detail in the subsequent sections.)

A branch \( B \) of a reconstruction graph \( T \), rooted at \( y_0y'_0 = IVIV' \), is fertile, if \( \text{dePad}_n(m_1m_2\cdots m_k) = \bot \), where \( \{m_1, m_2, \cdots, y_{k-1}m_k\} \) is the sequence of weights on the branch \( B \). The \( y_{k-1}' \) is computed following the recursion: \( y_{i+1}y_{i+1}' = \text{ro}(y_i, m_{i+1}) + (y_i'\|0) \), for all \( 0 \leq i \leq k - 2 \). All \( y_i \)'s and \( y_i' \)'s are \( n \) bits each; \( m_i \)'s are \( \ell \) bits each, except \( m_k \), which is an \((\ell - n)\)-bit string.

**Remark:** Each fertile branch of a reconstruction graph corresponds to exactly one reconstructible message.

**Definition 4.2 (Full reconstruction graph)** A reconstruction graph \( T \) (for the set \( D \)) is full, if, for each reconstructible message \( M \) (for \( D \)), \( T \) contains a fertile branch \( B \) that corresponds to \( M \).

4.2.3 View

Very loosely, the data structure view records the history of the interaction between a system and an adversary. Let \( x_i \) and \( y_i \) be the \( i \)-th query from the adversary and the corresponding response from the system. The view of the system after \( j \) queries is the sequence of queries and responses \( \{(x_1y_1), \ldots, (x_jy_j)\} \).
\[\text{FWP}(M)\]

01. \(\text{pad}(M) := m_1 m_2 \ldots m_{k-1} m_k;\)
02. \(y_0 := IV, y'_0 := IV';\)
03. for \((i := 1, 2, \ldots, k - 1)\)
   \(y_i y'_i := \text{ro}(y_{i-1} || m_i) \oplus (y'_{i-1} || 0);\)
04. \(r := \text{ro}(y_{k-1} || y'_{k-1} || m_k);\)
05. return \(r[n, 2n - 1];\)

\[\text{ro}(x)\]

11. if \(x \notin \text{Dom}(D_{ro})\) then \(D_{ro}[x] \leftarrow \{0, 1\}^{2n};\)
12. return \(D_{ro}[x];\)

(a) System G0: Global variable is the table \(D_{ro}\). All \(y_i's\) are \(n\) bits each; \(m_i\) is an \(\ell\)-bit string for all \(1 \leq i \leq k - 1\); \(m_k\) is \((\ell - n)\)-bit long. See Section 5 for description.

\[\text{RO}(M)\]

001. if \(M \in \text{Dom}(D_t)\)
    then return \(D_t[M];\)
002. \(r \leftarrow \{0, 1\}^n;\)
003. return \(r;\)

\[\text{MessageRecon}(x, T_s)\]

201. \(x \xrightarrow{\text{pad}} yy'm;\)
202. \(M' := \text{FindBranch}(yy', T_s);\)
203. \(M := \{\text{dePad}(M'm) \mid M' \in M'\};\)
204. return \(M;\)

(b) System G2: Global variables are the tables \(D_t\) and \(D_s\), and the graph \(T_s\). \(|y| = |y'| = n\) and \(|m| = \ell - n\). See Section 6 for description.

Figure 3: The systems G0 and G2
5 Main system G0

Following the definition in Section 2.2 the system G0 implements the FWP mode using the random oracle \( \mathsf{ro} : \{0,1\}^{\ell+n} \rightarrow \{0,1\}^{2n} \) (see Figure 3(a)).

6 Main system G2

See Figure 3(b) for the pseudocode. The random oracle \( \mathsf{RO} \) mentioned in Section 3 is implemented through lazy sampling. The only remaining part is to construct the simulator \( S \). Our design strategy for the simulator is fairly straightforward and simple. Before going into the details, we first provide a high level intuition.

6.1 Intuition for simulator \( S \)

The purpose of the simulator pair \( S \) is two-pronged: (1) to output values that are indistinguishable from the output from the random oracle \( \mathsf{ro} \), and (2) to respond in such a way that \( \mathsf{FWP}^\mathsf{ro}(M) \) and \( \mathsf{RO}(M) \) are identically distributed. It will easily follow that as long as the simulator \( S \) is able to output values satisfying the above conditions, no adversary can distinguish between G0 and G2.

To achieve (1), the simulator \( S \), for a distinct input \( x \), should output a random value, such that the distributions of \( S(x) \) and \( \mathsf{ro}(x) \) are close.

To achieve (2), the simulator needs to do the following:

- Building the full reconstruction graph. To asses the adversarial power, the simulator \( S \) maintains the full reconstruction graph \( T_s \) for the set \( D_s \) containing all \( s \)-queries and responses; this helps the simulator keep track of all ‘FWP-mode-compatible’ messages (more formally, all reconstructible messages) that can be formed using the ‘known’ information. This is accomplished by a special subroutine \( \text{FullGraph} \). The pictorial representation of the reconstruction graph \( T_s \) is given in Figure 4.

- Adjusting the elements of the tables \( D_t \) and \( D_s \). Whenever a new reconstructible message \( M \) is found using the aforementioned reconstruction graph \( T_s \), the simulator makes this crucial adjustment: it assigns \( \mathsf{FWP}^S(M) := \mathsf{RO}(M) \). It is fairly intuitive that, if \( S \) and \( \mathsf{ro} \) produce outputs according to statistically close distributions, then the distributions of \( \mathsf{FWP}^S(M) \) and \( \mathsf{FWP}^\mathsf{ro}(M) \) are also close. Since \( \mathsf{FWP}^S(M) = \mathsf{RO}(M) \), the distributions of \( \mathsf{RO}(M) \) and \( \mathsf{FWP}^\mathsf{ro}(M) \) are also close. This is accomplished by the subroutine \( \text{MessageRecon} \).
6.2 Detailed description of simulator $S$

We first describe the two most important parts of the simulator $S$: the subroutines FullGraph and MessageRecon. See Figure 3(b) for the pseudocode.

**FullGraph** ($D_s$): This routine builds the full reconstruction graph $T_s$ using all the $s$-queries and responses stored in $D_s$. Hence the name FullGraph. We do not provide the pseudocode for this subroutine, since its operation is straightforward and brute force: every invocation constructs the graph $T_s$ by searching through the elements in $D_s$, then creating all possible nodes, and finally connecting them to create the graph $T_s$. The reconstruction graph $T_s$ is pictorially presented in Figure 4. In Appendix C, we compute the running time of FullGraph on the $i$-th query to be $O(i^4)$.

![Figure 4: The reconstruction graph $T_s$ for $D_s$ (or $T_{ro}$ for $D_{ro}$) updated by FullGraph of G2 (or PartialGraph of G1).]

**MessageRecon** ($x, T_s$). The graph $T_s$ is already the full reconstruction graph for the set $D_s$. Given the current $s$-query $x$, this subroutine derives all new reconstructible messages $M$, such that, in the computation of $\text{FWP}^S(M)$, the final input to $S$ is $x$, and all other intermediate inputs to $S$ are old $s$-queries.

To determine such messages $M$, first, FindBranch($x, T_s$) collects all branches between the nodes ($IV, IV'$) and $x[0, 2n-1]$; then, it selects the sequence of weights $X = m_1m_2 \cdots m_{k-1}$ for all such branches. Finally it returns the set \{M = dePad($X||x[2n, \ell - 1]$)\} for all $X$. If no such $M = \perp$ is found, then the subroutine returns the empty set.

With the definition of the above subroutines, we now describe how $S$ responds to queries.
An s-query and response: For an s-query, the simulator $S$ assigns a uniformly sampled 2n-bit value to $r$. Then the subroutine $\text{MessageRecon}(x, T_s)$ is invoked that returns a set of reconstructible messages $\mathcal{M}$. If $|\mathcal{M}| = 1$ then the RO is invoked on $M \in \mathcal{M}$, and the value is assigned to $r[n, 2n - 1]$. Finally, the graph $T_s$ is updated by $\text{FullGraph}$, before $r$ is returned. In Appendix C we show that the worst-case running time of $S$ after $\sigma$ queries is $\mathcal{O}(\sigma^5)$.

7 Intermediate system G1

For description see Figure 5. For the sake of clear understanding, we first discuss the motivation for designing this system.

7.1 Motivation

The main motivation for constructing a new system G1 is that it is difficult to compare between the executions of the systems G0 and G2, instruction by instruction. The difficulty arises from the fact that G2 has a graph $T_s$, and two extra subroutines $\text{FullGraph}$, and $\text{MessageRecon}$, while G0 has no such graphs or subroutines. To get around this difficulty, we reduce G0 to an equivalent system G1 by endowing it with additional memory for constructing the graph $T_{ro}$, and by supplying it with additional subroutines $\text{MessageRecon}$ and $\text{PartialGraph}$. These additional components do not result in any difference in the input and output distributions of the systems G0 and G1 for any adversary (this result is formalized in Proposition 8.1); therefore, in the indistinguishability framework, G0 can be replaced by G1.

Even though G1 and G2 now appear 'similar', there are still important differences. The most crucial of them is that, in the former case, the long queries are processed as a sequence of ro-queries; therefore, current s-queries of G1 may match old ro-queries and responses, while such events are not possible for G2. This difference comes with two implications:

1. The reconstruction graph $T_{ro}$ in G1 is built using s-queries, ro-queries and their responses stored in the table $D_{ro}$; in case of G2, the reconstruction graph $T_s$ is built using only s-queries and responses stored in $D_s$. We extract from $T_{ro}$ the maximally connected subgraph built from all the s-queries and responses stored in $D_s$; we call the subgraph $T_s$. Now the reconstruction graph $T_s$ in both systems are comparable, since they are both built from the set $D_s$.

2. In G1, the reconstruction graph $T_{ro}$ may not be full for the set $D_{ro}$, since the subroutine $\text{PartialGraph}$ adds only a few nodes – rather than all nodes – to $T_{ro}$ every round; by contrast, the reconstruction graph $T_s$ – built by the subroutine $\text{FullGraph}$ – for G2 is necessarily full for the set $D_s$. In Section 9 we identify a set of events in the system G1, and then, in Section 10 show that, if those events do not occur, then the reconstruction graphs in both the systems are full.
7.2 Detailed description of G1

Figure 5: System G1: Global variables are the tables $D_l$ and $D_{ro}$ ($D_s$ is contained in $D_{ro}$), and the graph $T_{ro}$ ($T_s$ is a connected subgraph of $T_{ro}$). See Section 5 for description.

FWP1($M$)
001. pad($M$) := $m_1m_2 \cdots m_{k-1}m_k$;
002. $y_0 := IV, y'_0 := IV'$;
003. for $(i := 1, \ldots, k-1)$ do
004. $r := ro(y_{i-1}m)$;
005. $y_iy'_i := r \oplus (y'_{i-1}|0)$;
006. if $y_{i-1}m$ is fresh then PartialGraph($y_{i-1}m, r, T_{ro}$);
007. if Type3 then [BAD := True];
008. $r := ro(y_{k-1}y_{k-1}m_k)$;
009. if $y_{k-1}y_{k-1}m_k$ is fresh then PartialGraph($y_{k-1}y_{k-1}m_k, r, T_{ro}$);
010. $D_i[M] := r[n, 2n - 1]$;
011. return $r[n, 2n - 1]$;

MessageRecon($x, T_s$)
201. $x \mapsto yy'm$;
202. $M' := \text{FindBranch}(yy', T_s)$;
203. $M := \{ \text{dePad}(M'm) | M' \in M' \}$;
204. return $M$;

ro($x$)
301. if $x \notin \text{Dom}(D_{ro})$ then $D_{ro}[x] := \{0, 1\}^{2n}$;
302. return $D_{ro}[x]$;

S1($x$)
100. if Type2 then [BAD := True];
101. $r := ro(x)$;
102. $M := \text{MessageRecon}(x, T_s)$;
103. if $|M| = 1 \wedge M \not\in \text{Dom}(D_l)$ then $D_r[M] := r[n, 2n - 1]$;
104. $D_s[x] := r$;
105. if $x$ is fresh then PartialGraph($x, r, T_{ro}$);
106. return $r$;

PartialGraph($x, r, T_{ro}$)
400. $x \mapsto y_c m; r \mapsto y'y'$;
401. if Type0 then [BAD := True];
402. $C := \text{ContactPoints}(y_c)$;
403. $E := \{ (y_c y'_c, m, y'y') | y := y' \oplus y'_c, y_c y'_c \in C \}$;
404. for $e \in E \{ \text{AddEdge}(e) \}$;
405. if Type1-a $\lor$ Type1-b $\lor$ Type1-c then [BAD := True];
/*2nd Phase: (406 - 413)*/
406. for $(x, r) \in D_{ro}$
407. for $(y_c y'_c, m, y'y') \in E$
408. if $y = x[0, n - 1]$ then
409. $\{ z := r[0, n - 1] \oplus y' \}$
410. $z' := r[n, 2n - 1]$;
411. $m' := x[n, n + \ell - 1]$;
412. $\text{AddEdge}(yy', m', zz')$;
413. if Type1-d $\lor$ Type1-e $\lor$ Type1-f then [BAD := True];

In our first description of this system, we will ignore the statements where the variable BAD is set, since they impact neither the output nor the global data structures. The variable BAD is set when certain events occur in the global data structures. Those events will be discussed in Section 9. We now describe the subroutines used by this system.
PartialGraph\((x, r, T_\text{ro})\): This subroutine is called only when a fresh ro-query is produced. This subroutine updates the reconstruction graph \(T_\text{ro}\) (for the set \(D_\text{ro}\)) in the following way: Rather than building all possible paths using elements of \(D_\text{ro}\), this routine augments the \(T_\text{ro}\) in at most two phases; hence the name PartialGraph. The details are as follows. First, the subroutine ContactPoints\((y_r = x[0, n - 1])\) is invoked, which returns a set \(C\) containing all nodes in \(T_\text{ro}\), whose least-significant \(n\) bits are \(x[0, n - 1]\). We note that the nodes of \(T_\text{ro}\) contained in \(C\) are the places where the fresh ro-query will be attached. The pictorial representation of \(T_\text{ro}\) is in Figure 4.

1ST PHASE: Using the members of the set \(C\) and the fresh query-response pair \((x, r)\), fresh edges are constructed, stored in the set \(E\), and then added to \(T_\text{ro}\) using the subroutine AddEdge.

2ND PHASE: In this phase, a 2nd set of fresh nodes are added to the fresh nodes of the 1st phase, using all query-response pairs stored in table \(D_\text{ro}\). The details are as follows: if the least significant \(n\) bits of an old query \(x \in \text{Dom}(D_\text{ro})\) equals the least significant \(n\) bits of the tail node of an edge \(e \in \text{EdgeNew}\), then a fresh edge is constructed as before, and then it is attached to the tail node of \(e\) using AddEdge.

MessageRecon\((x, T_s)\). This subroutine has already been described in the context of G2. However, in this context of G1, there is an important point to note: The graph \(T_s\) used by this subroutine is the maximally connected subgraph of \(T_\text{ro}\) generated by the \(s\)-queries and responses with root \((IV, IV')\).

Now we describe how the oracles FWP1 and S1 respond to queries.

**FWP1**: FWP1 mimics FWP, while updating the graph \(T_\text{ro}\) using the subroutine PartialGraph, whenever a fresh ro-query is generated. \(D_l[M]\) is assigned \(r[n, 2n - 1]\), where \(r\) is the output from the final ro call. Finally, \(r[n, 2n - 1]\) is returned.

**Simulator S1**: Given the \(s\)-query \(x\), the \(s\)-oracle S1 computes \(ro(x) = r\). Then the subroutine MessageRecon\((x, T_s)\) is called which returns a set of messages \(\mathcal{M}\). If \(|\mathcal{M}| = 1\), and if \(M \notin D_l[M]\), then \(D_l[M]\) is assigned the value of \(r[n, 2n - 1]\). Before finally returning \(r[n, 2n - 1]\), the subroutine PartialGraph is called with input \((x, r, T_\text{ro})\), if it is fresh, to update the existing graph \(T_\text{ro}\).
8 First Part of Main Theorem: Proof of (2)

With the description of the systems (in Sections 5, 6, and 7) at our disposal, we are well equipped to prove (2).

Proposition 8.1 For any distinguishing adversary $A$,

$$\Pr A^{G_0} \Rightarrow 1 = \Pr A^{G_1} \Rightarrow 1.$$  

Proof. From the description of $S_1$, we observe that, for all $x \in \{0,1\}^n$, $S_1(x) = ro(x)$. Likewise, from the description of FWP1 and FWP, for all $M \in \{0,1\}^*$, $\text{FWP1}(M) = \text{FWP}(M)$.

9 Type0, 1, 2, and 3, of System G1

In this section, we concretely define the Type0, Type1, Type2, and Type3 events of the system G1 (see Figure 3). Informally they will be called ‘bad’ events, since these events set the variable BAD in G1. We first provide the motivation for these events.

9.1 Motivation

We recall that the adversary submits $s$- and long queries to the system G1 and receives responses, and based on the history of query-response pairs — known as view — she then tries to distinguish G1 from G2. Intuitively, those events are called ‘bad’, for which the outputs from the ro oracles of G1 can be predicted by the adversary with probability better than when interacting with G2. These events primarily involve various forms of collision occurring in the outputs of queries, allowing the adversary to generate non-trivial reconstructible messages. Secondly, we need to catch the events where current queries match old queries too. One can intuit that these events may help the adversary in distinguishing G1 from G2. It is also important to note that, if $T_{ro}$ is not a full reconstruction graph then the adversary can also use this fact to compel G1 to produce outputs different from those from G2 (since G2 always maintains the full reconstruction graph $T_s$). Lastly, the absence of ‘bad’ events will be able to restrict the growth of the reconstruction graph $T_{ro}$ every round; this limits the number of reconstructible messages.

Next sections deal with concrete definitions of these events, keeping the above motivation in mind.

9.2 Classifying elements of $D_{ro}$, branches of $T_{ro}$, and ro-queries

Definition of Type0 to Type3 events depend on the elements in $D_{ro}$, the branches of $T_{ro}$, and the types of ro-queries. In the following sections we first classify them.
(a) (i) Six types of a re-query and response; (ii), (iii), and (iv) Type2 events of system G1. The type Q5 in (iv) is divided into two subcases in Figure 7. The \( l + n \) bits of query and \( 2n \) bits of response of the current query have been denoted by \( x \) and \( r \) in all cases.

(b) Different types of a branch in the graph \( T_o \), (i), (ii) and (iii) show red branches. (iv) A green branch is a branch in the subgraph \( T_s \). The final input to \( ro \) is denoted by \( y_{k-1}y_{k-1}m_k \) in all cases.

Figure 6: Pictorial description of Type2 and Type3 events of the system G1 described in Figure 5
Figure 7: Two possible cases of type Q5 (see description in Section 10).
9.2.1 Elements of $D_{ro}$: six types

We deal with an event when the current $s$-query $x$ is already in the table $D_{ro}$. We classify the elements stored in $D_{ro}$, according to its known and unknown parts. The known part of a ro-query and response is the part that is present in the view of the system $G1$, or it can be derived from the view with probability 1; the unknown part is not present in the view, and it cannot be derived from the view with probability 1. There are six types of an old query-response denoted by $Q1$, $Q2$, $Q3$, $Q4$, $Q5$ and $Q6$, as shown in Figure 6(a).i. See Appendix [D] for the proof. The first five types were generated as the intermediate ro-queries and responses during the execution of old long queries; the sixth type is an old $s$-query and the response. The red and green circles denote the unknown and the known parts. The higher order bits are placed on the right. We divide a $Q5$ query into two cases according to its position in $T_{ro}$ (depicted in Figure 7): (Q5-1) In a branch, all ro-queries preceding the $Q5$ query are of type $Q6$; (Q5-2) In a branch, there is at least one ro-query (preceding the $Q5$ query) which is not of type $Q6$.

9.2.2 Branches of $T_{ro}$: four types

The branches of $T_{ro}$ can be classified into four types, as shown in Figure 6(b)(i) to (iv). A branch $B$ is: type (i), if the final query is $Q1$, $Q2$ or $Q5$; type (ii), if the final query is $Q3$ or $Q4$; type (iii), if the final query is $Q6$, and if one of the intermediate queries is $Q1$, $Q2$, $Q3$, $Q4$ or $Q5$; type (iv), if all queries are $Q6$. The first three types are called red branch. The fourth type is called green branch.

9.2.3 The ro-queries: seven types

We observe that – based on the types described in the sections above – the current ro-query can be categorized into the following classes.

1. Current ro-query is an $s$-query. This can be of two types.

   (a) The ro-query is fresh.

   (b) The ro-query is one of six types of elements in $D_{ro}$ described in Section 9.2.1

2. Current ro-query is an intermediate ro-query for the current long query. This is of three types.

   (a) Current long query is present on a red branch – as defined in Section 9.2.2 – of the graph $T_{ro}$. The ro-query in this case is necessarily one of six types stored in $D_{ro}$; we divide it into two cases.

      i. The ro-query is the final one.

      ii. The ro-query is a non-final one.
(b) Current long query is present on a green branch of the graph \( T_\text{ro} \). The ro-query in this case is also one of six types stored in \( D_\text{ro} \).

(c) Current long query is not present on a branch of the graph \( T_\text{ro} \). We divide the ro-query into two types.
   i. The ro-query is fresh.
   ii. The ro-query is one of six types of elements in \( D_\text{ro} \).

9.3 Definition: Type0 and Type1 on fresh queries

9.3.1 Intuition

We address the classes 1a, and 2c(i) of Section 9.2.3 together, since they are connected by the fact that the ro-query is fresh. As described in Section 9.1 for a fresh query, the absence of ‘bad’ events (1) prevents generation of nontrivial reconstructible messages, (2) (linearly) restricts the growth of the graph \( T_\text{ro} \), and (3) makes \( T_\text{ro} \) a full reconstruction graph.

1. A non-trivial reconstructible message is generated, (i) if the fresh ro-query causes a node collision in the graph \( T_\text{ro} \), or (ii) if it causes an old query to be attached to a fresh node. Type1-a, Type1-c, Type1-d and Type1-f events cover all the above conditions. See Figure 8(b).

2. Absence of Type0, Type1-b, and Type1-d restricts the growth of the graph \( T_\text{ro} \) to a constant number of nodes every fresh query (i.e., linearly after \( \sigma \) fresh queries). See Figures 8(a) and 8(b).

3. The goal of Type1-f – in addition to the one described in (1) – is that its absence makes \( T_\text{ro} \) a full reconstruction graph after two phases.

Importance of the two-phase framework: The first novelty of our work lies in our carefully designed ‘bad’ events – especially the Type0 and Type1 events – that are spread across two phases. More precisely, the absence of these events allows the graph \( T_\text{ro} \) to be augmented in two phases, rather than in one phase (see Figure 8); at the same time, it allows the graph to have the aforementioned properties. The two-phase framework – as we will see subsequently – is essential in breaking the birthday barrier of \( n/2 \) bits. In a similar way, the two-phase framework could be extended to a three-phase framework to go even beyond \( 2n/3 \) bits (see Section 13). But a rigorous theoretical analysis of that is a challenging task.
### Type0 event of system G1

(a) $\text{Type0}$

<table>
<thead>
<tr>
<th>Type</th>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old1</td>
<td>$\text{Old1} \equiv \text{Old2}$</td>
<td>3-multi-collision on right coordinates</td>
</tr>
</tbody>
</table>

#### Type1-a

- **1st phase**: Fresh $y$, Old $y'$, $m$
- **2nd phase**: $\text{Old1} \equiv \text{Old2}$
- **Event**: Node-collision

#### Type1-b

- **1st phase**: Fresh $y$, Old $y'$, $m$
- **2nd phase**: $\text{Old1} \equiv \text{Old2}$
- **Event**: 3-multi-collision on left coordinates

#### Type1-c

- **1st phase**: Fresh $y$, Old $y'$, $m$
- **2nd phase**: $\text{Old} \equiv \text{Old}$
- **Event**: Query collision (2n bits)

#### Type1-d

- **1st phase**: Fresh $y$, Old $y'$, $m$
- **2nd phase**: $\text{Old1} \equiv \text{Old2}$
- **Event**: 3-multi-collision on left coordinates

#### Type1-e

- **1st phase**: Fresh $y$, Old $y'$, $m$
- **2nd phase**: $\text{Old1} \equiv \text{Old2}$
- **Event**: Query collision (n bits)

#### Type1-f

- **1st phase**: Fresh $y$, Old $y'$, $m$
- **2nd phase**: $\text{Old3} \equiv \text{Old2}$
- **Event**: Query collision (2n bits)

### Notation

- $\equiv$ = event of $n$ bit equality
- $\equiv\equiv$ = event of 2n bit equality
- Random $n$ bits
- $n$ bits
- $l-n$ bits
- $l$ bits

(b) Type1-a,b,c,d,e,f events of system G1.

Figure 8: The variable **Bad** is set in the subroutine **PartialGraph** of system G1 described in Figure 5.
9.3.2 Type0 event: collision in outputs of $\text{ro}$

See Figure 8(a). This event occurs if the right coordinate (or the most significant $n$ bits) of the output of the $\text{ro}$-query is equal to the right coordinates of the outputs of two distinct old queries in $D_{\text{ro}}$.

The absence of this event ensures the following: Suppose $a_1, a_2, a_3, \ldots, a_k$ are the final $\text{ro}$-queries for $k$ distinct long queries, where $a_1[n, 2n−1] = a_2[n, 2n−1] = \ldots = a_k[n, 2n−1]$. Then $k$ is at most 2. This event will be used in Section 11 to bound the number of certain nodes in $T_{\text{ro}}$.

9.3.3 Type1 event: collision in $T_{\text{ro}}$

See Figure 8(b). Let $(x, r)$ be the current fresh $\text{ro}$-query and response, such that $x = y_c||m$, and $r = y^*||y'$. Let the edge $(y_c y'_c, m, yy')$ be generated from $(x, r)$. We define this event by partitioning it into six cases.

- **Type1-a**: This event occurs if $yy'$ collides with a node already in $T_{\text{ro}}$. This collision can be used to generate at least two reconstructible messages in the next rounds – one of them can be used to distinguish G1 from G2.

- **Type1-b**: This event occurs if $y$ collides with the already colliding left-coordinates (or the least-significant $n$ bits) of two distinct nodes in $T_{\text{ro}}$; that is, these nodes together form a 3-multi-collision. The absence of this event – as we will see in Section 11 – bounds the new nodes added to $T_{\text{ro}}$ to a constant number every round.

- **Type1-c**: This event occurs if $yy'$ collides with the least significant $2n$ bits of an old query stored in $D_{\text{ro}}$; like before, this event can also be used to form a non-trivial reconstructible message in the next rounds.

- **Type1-d**: This event occurs if $y$ collides with the least significant $n$ bits of an old query, and if the resulting node $zz'$ (added in Phase 2) collides with a node already in $T_{\text{ro}}$. Note $z||z' = D_{\text{ro}}[y y'[0, 2n−1] \oplus (y'||0)]$. This collision can be used to generate at least two reconstructible messages in the next rounds.

- **Type1-e**: This event occurs if $y$ collides with the least significant $n$ bits of an old query, and if the left-coordinate $z$ of the resulting node $z||z'$ (added in Phase 2) collides with the left-coordinates of two distinct nodes already in $T_{\text{ro}}$. Like Type1-b, the absence of this event bounds the new nodes added to $T_{\text{ro}}$ to a constant number in the next round.

- **Type1-f**: This event occurs if $y$ collides with the least significant $n$ bits of an old query, and if the left-coordinate $z$ of the resulting node $z||z'$ (added in Phase 2) collides with the
least significant \( n \) bits of an old query. The absence of this event serves two goals at the same time: (1) it rules out the generation of a non-trivial reconstructible message and (2) it restricts the growth of \( T_{ro} \) only up to two phases every round.

9.4 Type2 and Type3 on old queries

9.4.1 Intuition

Now we deal with the classes 1b, 2a, 2b and 2c(ii) of Section 9.2.3. All of them address the issue when the current queries match old ones.

The class 1b happens, when an \( s \)-query matches one of five types of old elements stored in \( D_{ro} \); these events can potentially help the adversary in distinguishing between G1 and G2, and we identify class 1b as Type2, and class 2a as Type3 events; the case by case analysis of the events will follow in a while.

The remaining classes are now 2a, 2b and 2c(ii), when the adversary submits a long query – say \( M \) – to the oracle FWP1, and it is found that \( M \) is already present on some (fertile) branch of the graph \( T_{ro} \) (2a and 2b), or it is not present at all on any branch of \( T_{ro} \) (2c(ii)). The class 2c(ii) necessarily includes a fresh \( ro \)-query followed possibly by old \( ro \)-queries, and this scenario has already been considered in various forms of Type1 events.

It is clear that the class 2a(ii) and 2b will not help the adversary in distinguishing G1 from G2.

So now we focus on the class 2a(i), which deals with the final \( ro \)-query of a red branch. Depending on the type of branch, the adversary tries to predict the most significant \( n \) bits of the final \( ro \)-query (i.e., the hash output) with non-trivial probability; she succeeds only for Type3 events that will be discussed shortly.

9.4.2 Type2

Recall that a query-response pair in \( D_{ro} \) can be of six types: Q1 to Q6. Type2 event is divided into several cases depending on the type of the current \( s \)-query. See Figure 6(a)(ii-iv) for the pictorial presentation. Suppose \( (x, r) \) is the input-output of a fresh query.

**Type2-a:** If the query is of type Q1.

**Type2-b:** If the query is of type Q2.

**Type2-c:** If the query is of type Q5-2.

**Type2-d:** If the query is of type Q3, and if \( r \) is distinguishable from the uniform distribution \( U([0, 2^{2n} - 1]) \).
**Type2-e:** If the query is of type Q4, and if \( r \) is distinguishable from the uniform distribution \( \mathcal{U}[0, 2^{2n} - 1] \).

**Type2-f:** If the query is of type Q5-1, and if \( r[0, n - 1] \) is distinguishable from the uniform distribution \( \mathcal{U}[0, 2^n - 1] \).

9.4.3 **Type3**

In this case, we consider the final ro-query of a red branch as the current query. Several types of red branch – (i), (ii), and (iii) – are shown in Figure 6(b).

There are three types of Type3 event:

**Type3-a** If the current long query \( M \) is present as a red branch of type (i)\(^3\)

**Type3-b** If the current long query \( M \) is present as a red branch of type (ii), and if the most significant \( n \) bits of output being distinguishable from the uniform distribution \( \mathcal{U}[0, 2^n - 1] \).

**Type3-c** If the current long query \( M \) is present as a red branch of type (iii).

10 **Second Part of Main Theorem: Proof of (3)**

First, we first fix a few definitions.

10.1 **Definitions: GOOD\(_i\) and BAD\(_i\)**

*Events GOOD\(_i\) and BAD\(_i\).*  BAD\(_i\) denotes the event when the variable BAD is set during round \( i \) of G1, that is, when Type0, Type 1, Type2, or Type3 events occur. Let the symbol GOOD\(_i\) denote the event \(-\bigvee_{j=1}^{i-1} \text{BAD}_j\). 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 \( \text{BAD}_i \) does not occur, and if \( \text{GOOD}_{i-1} \) did occur, then the views of G1 and G2 (after \( i \) rounds) are identically distributed for any attacker \( \mathcal{A} \).

*Events GOOD\(_1\) and BAD\(_1\).* In order to get around a small technical difficulty in establishing the uniform probability distribution of certain random variables, we need to modify the above events GOOD\(_i\) and BAD\(_i\) slightly. The event BAD\(_1\) occurs when Type0, Type2, or Type3 events occur in the \( i \)-th round. The event GOOD\(_1\) is defined as \( \text{GOOD}_{i-1} \land \neg \text{BAD}_1 \).

\(^3\)Observe that this case implies a node-collision in \( T_0 \), since the \( y_{k-1} y_{k-1}^r m_k \) is the final ro-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.
10.2 Proof of (3)

With the help of the Type0 to Type3 events described in Section 9, we are equipped to prove (3). Recall that we need to show two things:

\[ \Pr \mathcal{A}^{G1} \Rightarrow 1 \leq \Pr \neg \text{GOOD}_1 \sigma \]

as well as

\[ \Pr \neg \text{GOOD}_1 \sigma \leq \Pr \neg \text{GOOD}_\sigma \leq \Pr \text{BAD}_i | \text{GOOD}_{i-1} . \]

Proof of (6) is straight-forward. To prove (5), we proceed in the following way. Observe

\begin{align*}
\Pr \mathcal{A}^{G1} \Rightarrow 1 \leq \Pr \mathcal{A}^{G2} \Rightarrow 1 \\
= \left( \Pr \mathcal{A}^{G1} \Rightarrow 1 | \text{GOOD}_1 \sigma \right) \cdot \Pr \text{GOOD}_1 \sigma \\
+ \left( \Pr \mathcal{A}^{G1} \Rightarrow 1 | \neg \text{GOOD}_1 \sigma \right) \cdot \Pr \neg \text{GOOD}_1 \sigma .
\end{align*}

If we can show that

\[ \Pr \mathcal{A}^{G1} \Rightarrow 1 | \text{GOOD}_1 \sigma = \Pr \mathcal{A}^{G2} \Rightarrow 1 | \text{GOOD}_1 \sigma , \]

then (7) reduces to (5), since

\[ \Pr \mathcal{A}^{G1} \Rightarrow 1 | \neg \text{GOOD}_1 \sigma \leq \Pr \mathcal{A}^{G2} \Rightarrow 1 | \neg \text{GOOD}_1 \sigma \leq 1. \]

As a result, we focus on establishing (8), which is done in Appendix E.

11 A Few Combinatorial Results

In order to prove (4), we will need a few combinatorial results. We first fix some notation.

Node \([i] \): The multiset of nodes in \( T_{ro} \) after \( i \) rounds in system G1.

\( N_1^{(i)} \) (and \( N_2^{(i)} \)): The number of nodes added to \( T_{ro} \), during the 1st phase (and 2nd phase) of the \( i \)-th iteration of system G1.

\( D_{ro}^{(i)} \): The table \( D_{ro} \) after \( i \) rounds.

\( N_{\text{right}}^{(i)}(a) \): The number of nodes in \( T_{ro} \) after \( i \) rounds, where the most significant \( n \) bits equal \( a \).
Left-Coset\(_A(x)\): Suppose \(A\) is a multiset on \(\{0,1\}^{2n}\). The multiset \(\text{Left-Coset}_A(x) = \{a \in A | a[0,n-1] = x\}\) contains all elements of \(A\) whose least significant \(n\) bits are equal to \(x\). Such a sub-multiset will be called a left-coset of \(A\), or simply a left-coset if \(A\) is clear from the context.

Right-Coset\(_A(x)\): Suppose \(A\) is a multiset on \(\{0,1\}^{2n}\). The multiset \(\text{Right-Coset}_A(x) = \{a \in A | a[n,2n-1] = x\}\) contains all elements of \(A\) whose most-significant \(n\) bits equal \(x\). As above, we will call such a sub-multiset a right-coset of \(A\), or simply a right-coset.

twin-left/twin-right: A 2\(n\)-bit string \(a\) is a twin-left/twin-right of a 2\(n\)-bit string \(b\), if \(a[0,n-1] = b[0,n-1]/\)if \(a[n,2n-1] = b[n,2n-1]\).

We now prove three important lemmas. The first one upper-bounds the size of the graph \(T_{ro}\), while the other two provide upper-bounds for the collision probability on the left and right coordinates of query-outputs and nodes on the graph.

**Lemma 11.1 (Node Counting)** Given \(\text{GOOD}_{i-1}\) occurs \((i \geq 1)\), then \((i) N^{(i)}_1 \leq 2\), \((ii) N^{(i)}_2 \leq i\), \((iii) N^{(i-1)}_{\text{right}}(a) \leq 4\) for all \(a \in \{0,1\}^n\), and \((iv) |\text{Node}^{(i-1)}| \leq 2i - 1\).

**Proof.** Since \(\text{GOOD}_{i-1}\) occurred, the events Type0, Type1-b and Type1-e did not occur during the first \(i - 1\) rounds of system G1. Therefore, the maximum size of the set Coset is 2 after \(i - 1\) rounds.

(i) \(N^{(i)}_1\) is upper-bounded by the maximum size of the set Coset after \(i - 1\) rounds, from which the result follows.

(ii) In the second phase of the \(i\)-th round, a query cannot be added to more than 1 node of \(T_{ro}\), since the nodes generated during the first phase have distinct left-coordinates. As there are \(i\) queries, we get the result.

(iii) We note that, given \(\text{GOOD}_{i-1}\) occurred – which essentially implies that Type0 (or 3-multi-collision on the most significant \(n\) bits of the output of a query), Type1-b and Type1-e (3-multi-collision on the least significant \(n\) bits of a node) did not occur – in the first \(i - 1\) rounds. Therefore, one query can be placed in a maximum of 2 places on \(T_{ro}\), and at most two queries can have identical most significant \(n\) bits, implying the result.

(iv) We claim that the number of edges in \(T_{ro}\) after \(i - 1\) rounds is at most \(2i - 2\). Suppose there were more than \(2i - 2\) edges in \(T_{ro}\). This would require that we have at least one query which has been added to the graph at more than 2 nodes. However, this leads to a contradiction due to the fact that \(\text{GOOD}_{i-1}\) occurred. Namely, we have that events
Type1-b and Type1-e did not occur. Now, since each edge has one tail node, including the root-node \((IV, IV')\) we get \(\lvert \text{Node}^{(i-1)} \rvert \leq 2i - 1. \)

Since we assume \(\sum_{i=1}^{\sigma} \Pr \text{BAD}_i \mid \text{GOOD}_{i-1} \leq \varepsilon = 1/2\) (see Section 3), (6) implies that \(\text{GOOD}_i \geq 1/2\) for all \(0 \leq i \leq \sigma\). In the following two lemmas we will use this fact.

**Lemma 11.2 (Left Coordinate Collision)** \(\text{The following inequality holds:}\)

\[
P_{\text{lec}}^i(y, A) \overset{\text{def}}{=} \Pr \left[ \text{Left-Coset}_A(y) \geq 2 \mid \text{GOOD}_{i-1} \land \exists yx \in A \right] \leq \frac{2(|A| - 1)}{2^n},
\]

where \(A \subseteq \text{Node}^{(i)}\), and \(x \in \{0, 1\}^n\).

**Proof.** We now label the elements of \(A = \{y_jx_j \mid j = 1, 2, \ldots, k\}\). We choose any pair \(y_jx_j\) and \(y_j'x_j'\) from \(A\), with \(j = j'\) and note that

\[
\begin{align*}
y_j &= \text{ro}(a_{j_1})[n, 2n-1] \oplus \text{ro}(a_{j_2})[0, n-1], \\
y_j' &= \text{ro}(a_{j'_1})[n, 2n-1] \oplus \text{ro}(a_{j'_2})[0, n-1].
\end{align*}
\]

We note that, if \(\text{GOOD}_{i-1}\) occurs then \(a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}\), implying

\[
\begin{align*}
\Pr [y_j = y_{j'} \mid \text{GOOD}_{i-1}] \\
= \Pr [y_j = y_{j'} \mid \text{GOOD}_{i-1} \land a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}] \\
= \Pr [y_j = y_{j'} \mid \text{GOOD}_{i-1} \land a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}] \cdot \Pr [\text{GOOD}_{i-1} \mid a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}] \\
\leq \frac{\Pr[y_j = y_{j'} \mid a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}]}{\Pr [\text{GOOD}_{i-1} \mid a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}]} \\
\leq \frac{1}{\Pr [\text{GOOD}_{i-1} \mid a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}]} \cdot \Pr [\text{GOOD}_{i-1} \mid a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}]
\end{align*}
\]

(9)

Now, \(\Pr [\text{GOOD}_{i-1} \mid a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}] = \frac{\Pr [\text{GOOD}_{i-1}]}{\Pr [a_{j_1}||a_{j_2} \neq a_{j'_1}||a_{j'_2}]}\) since

\[
\Pr [a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2} \mid \text{GOOD}_{i-1}] = 1.
\]

33
Putting this result in (9), we get,

\[
\Pr[y_j = y_j'] \mid \text{GOOD}_{i-1} \leq \frac{\Pr[a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2}]}{\Pr[\text{GOOD}_{i-1}]} \cdot \Pr[y_j = y_j'] \mid a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2} \\
\leq \frac{1}{\Pr[\text{GOOD}_{i-1}]} \cdot \Pr[y_j = y_j'] \mid a_{j_1}||a_{j_2} = a_{j'_1}||a_{j'_2} \\
= \frac{1}{\Pr[\text{GOOD}_{i-1}]} \cdot \frac{1}{2^n} \\
\leq \frac{2}{2^n}, \text{ since } \Pr[\text{GOOD}_{i-1}] \geq 1/2.
\]  

(10)

Notice that \(P_{\text{rec}}(y, A)\) is essentially the probability that the multiset \(A\) contains at least one twin-left of the node \(yx \in A\), given \(\text{GOOD}_{i-1}\) occurred. Setting \(y_1x_1 = yx \in A\), we get

\[
P_{\text{rec}}^{i}(y, A) \leq \Pr \bigg[ \bigvee_{j=2}^{\frac{|A|}{2}} (y_1 = y_j) \mid \text{GOOD}_{i-1} \bigg] \\
\leq \Pr y_1 = y_j \mid \text{GOOD}_{i-1} \\
= \frac{2(|A| - 1)}{2^n} \text{ (using (10)).}
\]  

(11)

Thus the proof is complete. \(\square\)

Lemma 11.3 (Right Coordinate Collision) The following inequality holds:

\[
P_{\text{rec}}^{i}(y, A) \overset{\text{def}}{=} \Pr[\text{Right-Coset}_A(y) \geq 2 \mid \text{GOOD}_{i-1} \land \exists xy \in A \leq \frac{2(i - 2)}{2^n},
\]

where the multiset \(A = \{x_jy_j \mid j = 1, 2, \ldots, i - 1\}\) contains the outputs of previous \(i - 1\) ro-queries, and \(x \in \{0, 1\}^n\).

Proof. We choose any pair \(x_ay_a\) and \(x_by_b\) from \(A\), with \(a = b\) and note that

\[
y_a = \text{ro}(m)[n, 2n - 1], \\
y_b = \text{ro}(n)[n, 2n - 1],
\]

where \(m\) and \(n\) are two previous ro-queries. Since \(m = n\),

\[
\Pr y_a = y_b \mid \text{GOOD}_{i-1} \leq \frac{1}{\Pr[\text{GOOD}_{i-1}]} \cdot \Pr y_a = y_b \\
\leq \frac{2}{2^n}, \text{ since } \Pr[\text{GOOD}_{i-1}] \geq 1/2.
\]  

(12)
Notice that $P_{rcc}^i(y, A)$ is essentially the probability that the multiset $A$ contains at least one twin-right of a node $xy \in A$, given $\text{GOOD}_{i-1}$ occurred. W.l.g, setting $x_1y_1 = xy \in A$, we get

\[
P_{rcc}^i(y, A) \leq \Pr \left( i-1 \prod_{j=2}^{i-1} (y_1 = y_j) \mid \text{GOOD}_{i-1} \right)
\]

\[
\leq \Pr \left( i-1 \prod_{j=2}^{i-1} (y_1 = y_j) \mid \text{GOOD}_{i-1} \right)
\]

\[
\leq \frac{2(i - 2)}{2^n} \quad \text{(by (12))}.
\]

\[\Box\]

12 Third (or Final) Part of Main Theorem: Proof of (4)

To prove (4), we need individually compute the probabilities $\text{Type0}_i$, $\text{Type1}_i$, $\text{Type2}_i$, and $\text{Type3}_i$ events described in Section 9. The suffix $i$ denotes the corresponding event in the round $i$.

12.1 Estimating probability of $\text{Type0}_i$

The Type0 event is displayed in Figure 8(a). The $2n$-bit output of the $i$th ro-query – which is fresh – is denoted by $y_i^* y_i'$. Let the multiset $A = \{y_j^* y_j' \mid j = 1, 2, \cdots, i - 1\}$ contain outputs of all previous $i - 1$ ro-queries. Now, from the definition of Type0 event we establish the following:
\[
\Pr \, \text{Type0}_i \mid \text{GOOD}_{i-1} \leq \Pr \, \exists y^* y'_i \in A \land \text{Right-Coset}_A(y'_i) \geq 2 \mid \text{GOOD}_{i-1}
\]

\[
= \Pr \, \exists y^* y'_i \in A \mid \text{GOOD}_{i-1} \\
\cdot \Pr \, \text{Right-Coset}_A(y'_i) \geq 2 \mid \text{GOOD}_{i-1} \land \exists y^* y'_i \in A
\]

\[
\leq \Pr \bigg( \bigcup_{j=1}^{i-1} (y'_i = y'_j) \bigg) \mid \text{GOOD}_{i-1} \cdot P^{\text{RCC}}_i(y'_i, A)
\]

\[
\leq \Pr \bigg( \bigcup_{j=1}^{i-1} (y'_i = y'_j) \bigg) \mid \text{GOOD}_{i-1} \cdot P^{\text{RCC}}_i(y'_i, A)
\]

\[
= \Pr \bigg( \bigcup_{j=1}^{i-1} (y'_i = y'_j) \bigg) \mid \text{GOOD}_{i-1} \cdot P^{\text{RCC}}_i(y'_i, A)
\]

\[
\leq \frac{i-1}{2^n} \cdot \frac{2(i-2)}{2^n}
\]

\[
\leq \frac{2i^2}{2^{2n}}.
\]

(13)

**12.2 Estimating probability of Type1**

We recall that if GOOD$_{i-1}$ occurs, then $|\text{Node}^{(i-1)}| = 2i - 1, N^{(i)}_1 \leq 2$ and $N^{(i)}_2 \leq i$ by Lemma [11.1]. Now, we can bound the probability of various Type1 events. The factor 2 on the left side of each inequality arises due to the two fresh nodes that can be added in the 1st phase. Several Type1 events are pictorially represented in Figure 8(b).

**12.2.1 Computing probability of Type1-a**

Let $N$ denote the number of nodes in the graph $T_{ro}$ after $i-1$ full rounds and the 1st phase of round $i$, given GOOD$_{i-1}$ occurred. Therefore, $N = |\text{Node}^{(i-1)}| + N^{(i)}_1 \leq 2i - 1 + 2 = 2i + 1$. It is straightforward to see from the figure,

\[
\Pr \, \text{Type1-a}_i \mid \text{GOOD}_{i-1} \leq 2 \cdot \frac{N}{2^{2n}} = \frac{2(2i + 1)}{2^{2n}} \leq \frac{6i}{2^{2n}}.
\]

(14)
12.2.2 Computing probability of Type1-b$_i$

Let a new node generated in the 1st phase of the $i$-th round be denoted by $yy'$. Let $A$ denote the multiset of all nodes added to the graph $T_{ro}$ up to the end of the 1st phase of the $i$-th round minus the node $yy'$. Therefore, $|A| \leq |\text{Node}_{i}^{(i)}| - 1 = 2i$. We label the elements of $A = \{y_jx_j \mid j = 1, 2, \cdots, k\}$. For any $x \in \{0, 1\}^n$,

\[
\Pr \ Type1-b_i \mid \text{GOOD}_{i-1} \leq 2 \cdot \Pr \ \exists yx \in A \land |\text{Left-Coset}_A(y)| \geq 2 \mid \text{GOOD}_{i-1} \\
= 2 \cdot \Pr \ \exists yx \in A \mid \text{GOOD}_{i-1} \\
\cdot \Pr \ |\text{Left-Coset}_A(y)| \geq 2 \mid \text{GOOD}_{i-1} \land \exists yx \in A
\]

Estimated in Lemma 11.2

\[
\leq 2 \cdot \Pr \ \bigvee_{j=1}^{\frac{|A|}{2n}} (y = y_j) \mid \text{GOOD}_{i-1} \cdot P_{lcc}^i(y, A) \leq 2 \cdot \Pr \ \bigvee_{j=1}^{\frac{|A|}{2n}} (y = y_j) \mid \text{GOOD}_{i-1} \cdot P_{lcc}^i(y, A)
\]

\[
\leq 2 \cdot \Pr \ y = y_j \mid \text{GOOD}_{i-1} \cdot P_{lcc}^i(y, A) \leq 2 \cdot \Pr \ y = y_j \mid \text{GOOD}_{i-1} \cdot P_{lcc}^i(y, A)
\]

\[
\leq 2 \cdot \frac{|A|}{2n} \cdot \frac{2(|A| - 1)}{2n} \leq 2 \cdot \frac{2i}{2n} \cdot \frac{2(2i - 1)}{2n} \leq \frac{16i^2}{2^{2n}}.
\]

12.2.3 Computing probability of Type1-c$_i$

Since the maximum number of queries after $i$ rounds is also $i$, from Figure 8(b) we see:

\[
\Pr \ Type1-c_i \mid \text{GOOD}_{i-1} \leq 2 \cdot \frac{i}{2^{2n}} \leq \frac{2i}{2^{2n}}. \quad (15)
\]

12.2.4 Computing probability of Type1-d$_i$

We now define two events $E_1$ and $E_2$ as shown in Figure 8(b). $E_1$ denotes the event that the least-significant $n$ bits of an old query are equal to the least-significant $n$ bits of a fresh node – denoted by $y$ – in the 1st phase. $E_2$ denotes the event that the node $zz'$ – which is generated in the 2nd phase – are equal to two distinct nodes in the graph $T_{ro}$. Since there are at most two fresh nodes in the 1st phase and there are at most $i$ queries, we get
Pr Type1-d_i | GOOD_{i-1} \leq 2 \cdot i \cdot \Pr E_1 \land E_2 | GOOD_{i-1}  \\
= 2i \cdot \Pr E_1 | GOOD_{i-1} \cdot \Pr E_2 | E_1 \land GOOD_{i-1}  \\
= 2i \cdot \Pr E_1 | GOOD_{i-1} \cdot \Pr E_2 | GOOD_{i-1} \cdot Pr GOOD_{i-1}. \quad (16)

The last equality of (16) holds since $E_1$ is independent of $GOOD_{i-1}$, $E_2$ and $E_2 \land GOOD_{i-1}$.

It is easy to see that $\Pr E_1 | GOOD_{i-1} = 1/2^n$. Now we estimate $\Pr E_2 | GOOD_{i-1}$.

As denoted in Figure 8(b) (Type1-d), the node $zz'$ has been generated by the query Old1. We observe that, given $GOOD_{i-1}$, the number of nodes, other than the node $zz'$, generated from query Old1 in the graph $T_{ro}$ is at most 2 (otherwise, there is a 3-collision on the left-coordinates in $T_{ro}$, which is prohibited by $GOOD_{i-1}$). Similarly, given $GOOD_{i-1}$, the number of nodes generated from query Old2 – which is different from Old1 – is at most $|\text{Node}^{(i-1)}| + N^{(i)}_1 + N^{(i)}_2 \leq 3i - 1$ by Lemma 11.1. Therefore,

$$
\Pr E_2 | GOOD_{i-1} \leq \frac{2}{1} \cdot \frac{1}{2^n} + \frac{3i-1}{1} \cdot \frac{1}{2^n} \cdot \frac{1}{\Pr GOOD_{i-1}} \leq \frac{3}{2^n}.
$$

Putting the above values in (16), we get

$$
\Pr \text{Type1-d}_i | GOOD_{i-1} \leq 2i \cdot \frac{1}{2^n} \cdot \frac{3}{2^n} \leq \frac{6i}{2^{2n}}. \quad (17)
$$

12.2.5 Computing probability of Type1-e_i

As before, we define two events $E_1$ and $E_2$ as shown in Figure 8(b). $E_1$ is defined identically as before; therefore, $\Pr E_1 | GOOD_{i-1} = 1/2^n$. The event $E_2$ occurs when $z$ equals the least-significant $n$ bits of two distinct nodes in the graph $T_{ro}$. In other words, event $E_2$ occurs if $\text{Left-Coset}_{\text{Node}^{(i)}}(z) \geq 3$ which is equivalent to the event $\exists zy \in A \land \text{Left-Coset}_{A}(z) \geq 2$ where, $A = \text{Node}^{(i)} \setminus \{zz'\}$. As before, $E_1$ is independent of $GOOD_{i-1}$, $E_2$ and $E_2 \land GOOD_{i-1}$. Therefore,

$$
\Pr \text{Type1-e}_i | GOOD_{i-1} \leq 2i \cdot \Pr E_1 | GOOD_{i-1} \cdot \Pr E_2 | GOOD_{i-1}  \\
= 2i \cdot \frac{1}{2^n} \cdot \Pr E_2 | GOOD_{i-1}. \quad (18)
$$
Now, we estimate the following probability

\[
\Pr E_2 \mid \text{GOOD}_{i-1} = \Pr \exists y \in A \land \text{Left-Coset}_A(z) \geq 2 \mid \text{GOOD}_{i-1}
\]

\[
= \Pr \exists y \in A \mid \text{GOOD}_{i-1} \cdot \Pr \text{Left-Coset}_A(z) \geq 2 \mid \text{GOOD}_{i-1} \land \exists y \in A.
\]

\[
\leq \frac{3i}{2^n} \cdot \frac{2(3i - 1)}{2^n} \leq \frac{18i^2}{2^{2n}}
\]

since, given \( \text{GOOD}_{i-1} \), \(|A| = |\text{Node}^{(i)}| - 1 = |\text{Node}^{(i-1)}| + N_1^{(i)} + N_2^{(i)} - 1 \leq (2i-1) + 2 + i - 1 = 3i \) by Lemma 11.1.

Using the above equation and (18), we finally estimate

\[
\Pr \text{Type1-e}_i \mid \text{GOOD}_{i-1} \leq \frac{36i^3}{2^{3n}}.
\]

### 12.2.6 Computing probability of Type1-f_i

The event has been described in Figure 8(b). The event \( E_1 \) is the same as above. The event \( E_2 \) occurs when \( z \) equals the least-significant \( n \) bits of a query. As in the previous case, \( E_1 \) is independent of \( \text{GOOD}_{i-1} \), \( E_2 \) and \( E_2 \land \text{GOOD}_{i-1} \). Therefore,

\[
\Pr \text{Type1-f}_{i+1} \mid \text{GOOD}_i = 2i \cdot \Pr E_1 \mid \text{GOOD}_{i-1} \cdot \Pr E_2 \mid \text{GOOD}_{i-1}
\]

\[
\leq 2i \cdot \frac{1}{2^n} \cdot \frac{i}{2^n} = \frac{2i^2}{2^{2n}}.
\]

### 12.2.7 Final summation

Adding all the previous constituent probabilities we obtain,

\[
\Pr \text{Type1}_i \mid \text{GOOD}_{i-1} \leq 6i/2^{2n} + 16i^2/2^{2n} + 2i/2^{2n} + 6i/2^{2n} + 36i^3/2^{3n} + 2i^2/2^{2n}
\]

\[
\leq 14i/2^{2n} + 18i^2/2^{2n} + 36i^3/2^{3n}
\]

\[
\leq 68i^2/2^{2n}.
\]

### 12.3 Estimating probability of Type2_i

The following probabilities are easy to compute using the definition of Type2 events in Section 9.4.2.
12.3.1 Estimating probability of Type2-a_i

Note that a query of type Q1 is always the final ro-query for a long query; this implies that the middle \( n \) bits of such a query are the most-significant \( n \) bits of the output of another query. Since Type0 did not occur in the \( i - 1 \) rounds, there can be at most 2 queries with outputs having identical most significant \( n \) bits; each of these queries can be attached to the graph in at most 2 places, since Type1-b or Type1-e did not occur in the first \( i - 1 \) rounds. Therefore, the number of Q1 queries with identical middle \( n \) bits can be at most 4. Therefore,

\[
\Pr \text{Type2-a}_i \mid \text{GOOD}_{i-1} \leq \frac{\Pr \text{Type2-a}_i}{\Pr \text{GOOD}_{i-1}} \leq \frac{8}{2^n}.
\]

12.3.2 Estimating probability of Type2-b_i

Since there can be at most \( i \) queries of type Q2,

\[
\Pr \text{Type2-b}_i \mid \text{GOOD}_{i-1} \leq \frac{\Pr \text{Type2-b}_i}{\Pr \text{GOOD}_{i-1}} \leq \frac{2i}{2^{2n}}.
\]

12.3.3 Estimating probability of Type2-c_i

Note that there can be at most 1 query of type Q5-2 with identical \( \ell + n \) bits of input. Also note that the Q5-2 query is the final query of one branch – call it \( B \) – as shown in Figure 7. Now, we see that the branch \( B \) has at least one of Q1 to Q5 queries, followed by a Q5 or a Q6 query. Therefore,

\[
\Pr \text{Type2-c}_i \mid \text{GOOD}_{i-1} \leq \frac{\Pr \text{Type2-c}_i}{\Pr \text{GOOD}_{i-1}} \leq \frac{2}{2^n}.
\]

12.3.4 Estimating probability of Type2-d_i

Let \( E_1 \) denote the event that the current query is type Q3, and let \( E_2 \) denote the event that the output is distinguishable from the uniform distribution. Now Type2-d = \( E_1 \land E_2 \). Therefore,

\[
\Pr \text{Type2-d}_i \mid \text{GOOD}_{i-1} = \Pr E_1 \land E_2 \mid \text{GOOD}_{i-1} \\
\leq \Pr E_2 \mid \text{GOOD}_{i-1} \\
\leq \frac{2 - 1}{2^{2n}} \\
= \frac{1}{2^{2n}}.
\]
12.3.5 Estimating probability of Type2-\(e_i\)

Let \(E_1\) denote the event that the current query is type Q4, and let \(E_2\) denote the event that the output is distinguishable from the uniform distribution. Now \(\text{Type2-}e = E_1 \land E_2\). Therefore,

\[
\Pr \text{Type2-}e_i \mid \text{GOOD}_{i-1} = \Pr E_1 \land E_2 \mid \text{GOOD}_{i-1} \\
\leq \Pr E_2 \mid \text{GOOD}_{i-1} \\
\leq \frac{2 - 1}{2^n} \\
= \frac{1}{2^{2^n}}.
\]

12.3.6 Estimating probability of Type2-\(f_i\)

Let \(E_1\) denote the event that the current query is type Q5-1, and let \(E_2\) denote the event that the least significant \(n\) bits of output are distinguishable from the uniform distribution. Now \(\text{Type2-}f = E_1 \land E_2\). Therefore,

\[
\Pr \text{Type2-}f_i \mid \text{GOOD}_{i-1} = \Pr E_1 \land E_2 \mid \text{GOOD}_{i-1} \\
\leq \Pr E_2 \mid \text{GOOD}_{i-1} \\
\leq \frac{2 - 1}{2^n} \\
= \frac{1}{2^n}.
\]

12.3.7 Final summation

Adding all the previous constituent probabilities we obtain,

\[
\Pr \text{Type2}_i \mid \text{GOOD}_{i-1} \leq \frac{11}{2^n} + \frac{2}{2^{2n}} + \frac{2i}{2^{2n}} \leq \frac{15}{2^n} \\
\leq \frac{30}{2^n}.
\]

12.4 Estimating probability of Type3\(i\)

Examining the definition of Type3 events in the following results can be established.

12.4.1 Estimating probability of Type3-\(a_i\)

This event cannot occur – as observed in Section 9.4.3 – if Type1 event did not occur in the first \(i - 1\) rounds. Therefore,

\[
\Pr \text{Type3-}a_i \mid \text{GOOD}_{i-1} = 0.
\]
12.4.2 Estimating probability of Type3-b\(_i\)

From the definition in Section 9.4.3,

\[
\Pr \text{Type3-b}_i \mid \text{GOOD}_{i-1} \leq \frac{2 - 1}{2^n} = \frac{1}{2^n}.
\]

12.4.3 Estimating probability of Type3-c\(_i\)

From the definition in Section 9.4.3,

\[
\Pr \text{Type3-c}_i \mid \text{GOOD}_{i-1} \leq \frac{\Pr \text{Type3-c}_i}{\Pr \text{GOOD}_{i-1}} \leq \frac{2}{2^n}.
\]

12.4.4 Final summation

Adding all the previous constituent probabilities we obtain,

\[
\Pr \text{Type3}_i \mid \text{GOOD}_{i-1} \leq \frac{3}{2^n}.
\]

12.5 Final step

We prove (1) by combining the above bounds into the following inequality which holds for \(1 \leq i \leq \sigma\):

\[
\Pr \text{BAD}_i \mid \text{GOOD}_{i-1} \leq \Pr \text{Type0}_i \mid \text{GOOD}_{i-1} + \Pr \text{Type1}_i \mid \text{GOOD}_{i-1}
+ \Pr \text{Type2}_i \mid \text{GOOD}_{i-1} + \Pr \text{Type3}_i \mid \text{GOOD}_{i-1}
\leq 2i^2/2^{2n} + 6i^2/2^{2n} + 30/2^n + 3/2^n
\leq 103i^2/2^{2n}.
\]

Therefore,

\[
\sigma \sum_{i=1}^{\sigma} \Pr \text{BAD}_i \mid \text{GOOD}_{i-1} \leq \frac{103\sigma^3}{2^{2n}}.
\]

13 Experimental Results: The Bound Improves Towards \(n\) bits

We performed a series of experiments studying the effects of the bad events in our theoretical framework. Our simple C implementation of the game G1 simulated the random oracle, ro, with randomness supplied by rand(), by maintaining a database of input/output pairs,
assuring that \( r_0 \) commits to an output while allowing fresh queries to remain independent of all past events.

We collected data providing accurate estimates for the values \( \Pr \) Type1\(_i\) \( \mid \) GOOD\(_{i-1}\) described in Section 9. Compiling these data we computed the relative percentage of several Type1 events. The results are provided in Table 13. The experimental results strongly agree with the bounds computed in Section 12.

In addition to these event probabilities, we calculated security bounds for several values of \( n \). The computation was achieved by randomly generating a large number of graphs, \( T_{r_0} \), and determining the number of queries, \( \sigma \), required to cause \( \sum \Pr \text{ Type1}_i \mid \text{GOOD}_{i-1} \geq 0.5 \). The results of the experiments following this method are summarized in the blue line of Figure 9. Naturally, the data follow the theoretically obtained bound of \( \sigma = \Omega(2^{2n/3}) \) (see Theorem 3.1). Some of the values in the graph are slightly lower than \( 2/3 \), due to the effect of constants. The data asymptotically approach \( 2/3 \).

We did not consider the Type2 and Type3 events, since, for any efficient adversary the probabilities of these events are dominated by that of the Type1 events. We found that choosing the values at which to place the 1st query uniformly at random from among all possible maximal left-cosets was the most advantageous strategy for an adversary.

Hoping that the real indistinguishability security bound is more than \( 2n/3 \)-bit, we experimented with a smaller set of ‘bad’ events than the ones described in Section 9. We first removed the ‘artificial’ Type1-b, Type1-e, and Type0 events; it is easy to see that (2) and (3) can be proved without these events. These events were inserted so that it is mathematically easier to derive the bounds on the size of the graph \( T_{r_0} \) (see Section 11). It is our intuition that a cleverer mathematician will be able to obtain similar bounds even without these ‘artificial’ events. Then we have switched from the two-phase framework to a natural extension of three-phase (or multi-phase) framework. In short, with the construction of a multi-phase version of the game \( G_{1} \), removing the artificial Type1-b, Type1-e, and Type0 events, we see that the normalized logarithm of \( \sigma \) increases towards one as \( n \) increases. The Type1 events of the three-phase version of \( G_{1} \) are illustrated in Figure 10. These data indicate that the bound may be significantly improved, possibly even to nearly \( n \) bits, as reflected in the red line of Figure 9.

14 Conclusion and Open Problems

Indistinguishability security guarantees absence of all generic attacks. In this paper we improved the indistinguishability security bound of the FWP hash mode from \( n/2 \) to \( 2n/3 \) bits. Many popular hash modes use primitives of the form \( C : \{0, 1\}^{2n} \rightarrow \{0, 1\}^{2n} \). For such an important family, the FWP becomes the only mode to achieve indistinguishable security of more than \( n/2 \) bits. Secondly, among \( n \)-bit hash modes with \( a \geq 2n \), the FWP mode has the highest rate among all modes which have beyond-birthday-barrier security. Our experimental results strongly indicate that the bound could be further improved, possibly
<table>
<thead>
<tr>
<th>Event</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type1-a</td>
<td>6.2%</td>
<td>4.0%</td>
<td>2.7%</td>
<td>1.6%</td>
<td>1.0%</td>
<td>1.1%</td>
<td>0.6%</td>
</tr>
<tr>
<td>Type1-b</td>
<td>51.7%</td>
<td>58.2%</td>
<td>62.7%</td>
<td>67.2%</td>
<td>70.9%</td>
<td>74.2%</td>
<td>77.0%</td>
</tr>
<tr>
<td>Type1-c</td>
<td>5.5%</td>
<td>3.4%</td>
<td>2.3%</td>
<td>1.4%</td>
<td>0.7%</td>
<td>0.5%</td>
<td>0.5%</td>
</tr>
<tr>
<td>Type1-d</td>
<td>0.7%</td>
<td>0.3%</td>
<td>0.1%</td>
<td>0.1%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>Type1-e</td>
<td>1.4%</td>
<td>1.1%</td>
<td>0.9%</td>
<td>0.5%</td>
<td>0.4%</td>
<td>0.1%</td>
<td>0.3%</td>
</tr>
<tr>
<td>Type1-f</td>
<td>17.5%</td>
<td>15.7%</td>
<td>14.2%</td>
<td>12.5%</td>
<td>10.6%</td>
<td>10.0%</td>
<td>7.0%</td>
</tr>
<tr>
<td>Type0</td>
<td>17.1%</td>
<td>17.2%</td>
<td>17.1%</td>
<td>16.6%</td>
<td>16.5%</td>
<td>14.2%</td>
<td>14.6%</td>
</tr>
</tbody>
</table>

Table 2: The relative percentage of Type1 bad events for various values of $n$. Columns 6-9, 10-11, and 12 were computed by generating 100000, 10000, and 5000 trees, $T_0$, respectively.

Figure 9: Plot of experimental data of value of $n$ versus the normalized logarithm of $\sigma$, $\log_2(\sigma)/n$, for the systems “2-phase G1” (or simply G1) and “multi-phase G1” (labeled by $G'_1$).
Figure 10: The Type1 events of the 3-phase version of the system G1.
even close to \( n \) bits. In addition, our proof technique is novel, and it uses only three games with a set of specially designed bad events.

Our work leaves room for more research. The security upper-bound for the FWP is \( n \)-bit, while the proven lower-bound is \( \frac{2}{3}n \)-bit. One research direction would be to close the gap between these upper and lower bounds. Also, one may try to optimize the complexity of the simulator running time.

Acknowledgment

We would like to thank Donghoon Chang, Mridul Nandi, Tom Shrimpton, Cristina Tone, Meltem Sönmez Turan, and the Keccak team for their suggestions and remarks which helped a lot to improve the quality of the paper.

References


[29] Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [12], pages 428–446. (Cited on page 6.)


A Definitions

Definition A.1 (Random oracle) A random oracle is a function $RO: X \rightarrow Y$ chosen uniformly at random from the set of all $|Y|^{|X|}$ functions that map $X \rightarrow Y$. In other words, a function $RO: X \rightarrow Y$ is a random oracle if and only if, for each $x \in X$, the value of $RO(x)$ is chosen uniformly at random from $Y$. 

49
Figure 11: (i) Wide Pipe and (ii) Fast Wide Pipe modes; both are using the identical primitive $C$. $M_i$’s are message-blocks.

B  WP and FWP

C  Time costs of FullGraph and simulator S

Since there are $i$ queries after $i$ rounds, the maximum number of nodes in $T_s$ after $i$ round is $i^2$. Therefore, to construct $T_s$ in the $i$-th round, the amount of time required by FullGraph is $O(i^4)$. Now, if the adversary submits $\sigma$ queries, then the time complexity of FullGraph is $O(\sigma^5)$. Since the time of FullGraph dominates over the other costs such as MessageRecon, the worst-case simulator time complexity of $S$ is also $O(\sigma^5)$.

D  Six Types of ro-query-response Pairs

Lemma D.1 (Sextuple Queries) Based on the known and unknown parts, there are at most 6 types of a query-response stored in the table $D_{ro}$ – as shown in Figure 6(a)(i).

Proof. We observe from the (known and unknown parts of) $l$- and $s$-queries, and their responses that there are three types of output for an ro-query: (1) the least significant $n$ bits are unknown (they are the outputs of the final ro-queries in $l$-queries); (2) all $2n$ bits are unknown (they are the outputs of the intermediate ro-queries in $l$-queries); (3) all $2n$
bits are known (they are the outputs of the $s$-queries. Note that in case (1), the output can be of three types of input (Q1, Q2 and Q5); any extra type of input for this output will require at least four types of output in total, which is impossible. Case (2) output can have two types of input (Q3 and Q4). Finally, for case (3), an output can have only one type of input (Q6). Altogether, there are six types of query-response pairs.

E  Proof of \((8)\)

Let $V_i^1$ and $V_i^2$ denote the views of the systems G1, and G2 respectively, after $i$ queries have been processed. To prove \((8)\), it suffices to show that given $\text{GOOD}_1$, the views $V_i^1$ and $V_i^2$ are identically distributed. We do this by induction on the number of queries $\sigma$.

Induction Hypothesis: Given $\text{GOOD}_i$, $V_i^1$ and $V_i^2$ are identically distributed.

Base: When $i = 0$, then no query has been made; therefore the hypothesis is true.

Induction Step: Now assume the induction hypothesis holds. We have to show that if $\text{GOOD}_{i+1}$ occurred, then $V_{i+1}^1$ and $V_{i+1}^2$ are identically distributed.

Let $(I_{i+1}, O_{i+1}^1)$ and $(I_{i+1}, O_{i+1}^2)$ denote the input-output pairs for the systems G1 and G2 respectively in the $i+1$st round.

A little reflection shows that proving the induction step is equivalent to proving the following proposition.

**Proposition E.1 (Proof of Induction Step)**  Given $\text{GOOD}_{i+1}$ and $V_i^1 = V_i^2$

1. the input-views $I_{i+1}^1$ and $I_{i+1}^2$ are identically distributed;

2. if $I_{i+1}^1 = I_{i+1}^2$ then the output-views $O_{i+1}^1$ and $O_{i+1}^2$ are identically distributed.

**Proof.**

1. This result is easy since $V_i^1 = V_i^2$.

2. First, we establish the following lemma which is the main ingredient in our proof.

**Lemma E.2** The reconstruction graphs $T_s$ of the systems G1 and G2 are isomorphic after $i$ rounds, given $\text{GOOD}_i$ and $V_i^1 = V_i^2$.

**Proof.** For each fresh ro-query, the graph $T_\text{ro}$ of system G1 is augmented in two phases (see Figure 5). In these two phases, all possible nodes are added to the graph $T_\text{ro}$. An analysis of the Type1-a,c,d and f events show that if these events do not occur then no
nodes can be added beyond these two phases. In other words, if Type1-a,c,d and f events do not occur in i rounds then the graph $T_{\text{ro}}$ contains all possible paths generated from all elements stored in the table $D_{\text{ro}}$ in i rounds with root $(IV, IV')$. Note that the graph $T_s$ is the maximally connected subgraph of $T_{\text{ro}}$ rooted at $(IV, IV')$, generated only by the s-queries and responses stored in $D_s$. This implies that the graph $T_s$ of the system G1 contains all paths generated from all s-queries and responses with root $(IV, IV')$.

We note that the graph $T_s$ of G2 also contains all paths generated from all s-queries and responses with root $(IV, IV')$. As $V_i^1 = V_i^2$, the graph $T_s$ of G1 and G2 are isomorphic after i rounds.

Let $I_i^{i+1}$ denote the shared query input $I_i^{i+1} = I_2^{i+1}$. We continue by considering all possible cases based on a set of conditions for the system G1 in the i + 1st round; cases 1 through 9 consider when $I_{i+1}$ is an s-query, while cases 10 through 15 consider when $I_{i+1}$ is part of an long query. Our decision tree produced the above 15 cases, which have been derived from a sequence of questions (see Figure 12). The reader is invited to verify that all cases are considered.

**Case 1: s-query, Fresh, $|M| = 0$.**

*Implication.* The condition directly implies that $O_i^{1} = r$ follows the uniform distribution $U[0, 2^{2^{n}} - 1]$. Since the graphs $T_s$ are isomorphic in both systems G1 and G2 by Lemma [E.2] $|M| = 0$ for G2. This implies that $O_i^{2} = r$ follows the uniform distribution $U[0, 2^{2^{n}} - 1].$

**Case 2: s-query, Old, Type Q3 or Q4, $|M| = 0$.**

*Implication.* The event GOOD1$_{i+1}$ implies that Type2 event did not occur for G1 in the current i + 1th round; therefore, since $|M| = 0$, $O_i^{1} = r$ follows the uniform distribution $U[0, 2^{2^{n}} - 1]$. As the graphs $T_s$ are isomorphic in both systems G1 and G2 by Lemma [E.2] $|M| = 0$ for G2. This implies that $O_i^{2} = r$ follows the uniform distribution $U[0, 2^{2^{n}} - 1].$

**Case 3: s-query, Old, Type Q5-1, $|M| = 0$.**

*Implication.* This case is impossible since $|M| = 0$.

**Case 4: s-query, Old, Type Q1, Q2, or Q5-2, $|M| = 0$.**

*Implication.* This case is impossible since GOOD1$_{i+1}$ implies that Type2 event did not occur for G1 in the current i + 1st round.

**Case 5: s-query, $|M| > 1$**

*Implication.* $|M| > 1$ implies node-collision in $T_s$ which is impossible since GOOD1$_{i+1}$ ensures that Type1 event did not occur for G1 in the previous i rounds forbidding the occurrence of node-collision in $T_s$.

**Case 6: s-query, Fresh, $|M| = 1$**
Implication. Since $I_{i+1}$ is fresh, $O^1_{i+1} = r$ follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$. Now, for $G1$, $M \in \mathcal{M}$ implies that $M \notin \text{Dom}(D_i)$ in the first $i$ rounds, since the current $s$-query $I_{i+1}$ is fresh. Also note that $V_i^1 = V_i^2$ and the isomorphism of $T_s$’s together imply that $D_i$ in both systems are identical. Therefore, for $G2$ too, $M \notin \text{Dom}(D_i)$ in the first $i$ rounds. This implies that $O^2_{i+1} = r$ follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$.

Case 7: $s$-query, Old, Type Q3 or Q4, $|\mathcal{M}| = 1$.

Implication. The event GOOD1$_{i+1}$ implies that Type2 event did not occur in the $i + 1$st round of $G1$; therefore, $O^1_{i+1} = r$ follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$. Now, for $G1$, $M \in \mathcal{M}$ implies that $M \notin \text{Dom}(D_i)$ in the first $i$ rounds, since the current $s$-query $I_{i+1}$ is one of type Q3 and type Q4; the final ro-query of any long query cannot be of type Q3 or of type Q4. As in the previous case, $V_i^1 = V_i^2$ and the isomorphism of $T_s$’s together imply that $D_i$ in both systems are identical. Therefore, for $G2$ too, $M \notin \text{Dom}(D_i)$ in the first $i$ rounds. This implies that $O^2_{i+1} = r$ follows the uniform distribution $\mathcal{U}[0, 2^{2n} - 1]$.

Case 8: $s$-query, Old, Type Q5-1, $|\mathcal{M}| = 1$.

Implication. The event GOOD1$_{i+1}$ implies that Type2 event did not occur in the $i + 1$st round of $G1$; therefore, $O^1_{i+1}|[n, n - 1]$ follows the uniform distribution $\mathcal{U}[0, 2^n - 1]$, and $O^1_{i+1}[n, 2n - 1]$ is a fixed value. Now, for $G1$, $M \in \mathcal{M}$ implies that $M \in \text{Dom}(D_i)$ after the first $i$ rounds, since the current $s$-query $I_{i+1}$ is of type Q5-1; also note that $O^1_{i+1}[n, 2n - 1] = D_i[M]$. As in the previous case, $V_i^1 = V_i^2$ and the isomorphism of $T_s$’s together imply that $D_i$ in both systems are identical. Therefore, $O^2_{i+1}|[n, 2n - 1] = D_i[M]$; also note that $O^2_{i+1}[0, n - 1]$ 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, Old, Type Q1, Q2, or Q5-2, $|\mathcal{M}| = 1$.

Implication. This case is impossible since $|\mathcal{M}| = 1$, and $I_{i+1}$ cannot be any of the types Q1, Q2, and Q5-2.

Case 10: long query, Non-final Block.

Implication. Since $V^1_{i+1} = V^2_{i+1}$, it is easy to verify that $O^1_{i+1} = O^2_{i+1} = \lambda$, where, $\lambda$ is the empty string.

Case 11: long query, Final Block, long query not in $T_\text{ro}$.

Implication. Let $M$ be the long query in question. Since the event GOOD1$_{i+1}$ implies that Type1 did not occur in the previous $i$ rounds of $G1$, there are no node-collisions in the graph $T_\text{ro}$. Therefore, the final ro-query is fresh, implying $O^1_{i+1}$ follows the uniform distribution $\mathcal{U}[0, 2^n - 1]$. As before, the table $D_i$ in both systems were identical when the long query $M$ was submitted; therefore, at that time of submission, $M \notin \text{Dom}(D_i)$ for both the systems. This ensures that $O^2_{i+1} = \text{RO}(M)$ follows the uniform distribution $\mathcal{U}[0, 2^n - 1]$.
Case 12: long query, Final Block, long query in $T_\text{ro}$, long query in $T_s$.  
*Implication.* Since the graph $T_s$ in both systems are isomorphic by Lemma E.2, $O_{i+1}^1 = O_{i+1}^2$.

Case 13, 14 and 15: long query, Final Block, long query in $T_\text{ro}$, long query not in $T_s$. $I_{i+1}$ is the final message-block of a long query (denoted by $M$) which forms a red branch (see Section 9.4 and Figure 6(b)).

Case 13: Final ro-query is Type Q1, Q2 or Q5.  
*Implication.* Note that this case implies a node collision in $T_\text{ro}$. By definition, the ro-query $y_k y_{k-1} m_k$ is already the final ro-query of a previous long query since it is of type Q1, Q2 and Q5; now, $y_k y_{k-1} m_k$ is also the final ro-query of the current long query $M$. Hence the collision in $T_\text{ro}$. This case is impossible since GOOD1$_{i+1}$ implies that Type1 event did not occur in the first $i$ rounds; therefore $T_\text{ro}$ cannot have a node-collision.

Case 14: Final ro-query is Type Q3 or Q5.  
*Implication.* Since the event Type3 did not occur in the $i+1$st round, $O_{i+1}^1$ follows the uniform distribution $U[0, 2^n - 1]$. Now, for G1, the long query $M \notin \text{Dom}(D_t)$ when $M$ was submitted since the final ro-query of any long query cannot be of type Q3 or Q4. As the table $D_t$ of both systems are identical, for G2, $M \notin \text{Dom}(D_t)$ when $M$ was submitted. Therefore, $O_{i+1}^2 = \text{RO}(M)$ follows the uniform distribution $U[0, 2^n - 1]$.

Case 15: Final ro-query is Type Q6, and an intermediate query is Type Q1, Q2, Q3, Q4 or Q5.  
*Implication.* This case is impossible since Type3 in the $i+1$st round did not occur.

To conclude, in all 15 cases above we have shown that the outputs $O_{i+1}^1$ and $O_{i+1}^2$ are identically distributed if the variable BAD is not set. This completes the proof. □
Figure 12: The decision tree for the proof of Proposition E.1(2). The square-box at the root of the tree contains the initial conditions that are satisfied by Proposition E.1(1) and Lemma E.2. The conditions for the system G1 are shown inside the diamonds. The text in each leaf-node shows the implications of the conditions to the outputs of systems G1 and G2, while the reasons for such implications were describe in brief inside the bracket.