Permutation-Based Hashing Beyond the Birthday Bound

Charlotte Lefevre and Bart Mennink

Digital Security Group, Radboud University, Nijmegen, The Netherlands
charlotte.lefevre@ru.nl, b.mennink@cs.ru.nl

Abstract. It is known that the sponge construction is tightly indifferentiable from a random oracle up to around $2^{c/2}$ queries, where c is the capacity. In particular, it cannot provide generic security better than half of the underlying permutation size. In this paper, we aim to achieve hash function security beating this barrier. We present a hashing mode based on two b-bit permutations named the double sponge. The double sponge can be seen as the sponge embedded within the double block length hashing paradigm, making two permutation calls in parallel interleaved with an efficient mixing function. Similarly to the sponge, the permutation size is split as $b = r + c$, and the underlying compression function absorbs r bits at a time. We prove that the double sponge is indifferentiable from a random oracle up to around $2^{2c/3}$ queries. This means that the double sponge achieves security beyond the birthday bound in the capacity. In addition, if $c > 3b/4$, the double sponge beats the birthday bound in the primitive size, to our knowledge being the first hashing mode based on a permutation that accomplices this feature.

Keywords: double block length hashing, permutation-based hashing, sponge, lightweight cryptography, beyond birthday bound

1 Introduction

In recent years, there has been a growing interest in permutation-based hashing. Most of the concerned modes revolve around the sponge construction [BDPV07]. The sponge uses a global state of size b bits, also equal to the permutation size. The state size is split into two parts: r, called the rate, and c, called the capacity, so that $b = r + c$. The rate determines at which pace the messages are absorbed and the digest blocks extracted, while the capacity is a security parameter. It has been shown that the sponge is indifferentiable from a random oracle up to $\approx 2^{c/2}$ queries [BDPV08] in the framework of Maurer et al. [MRH04] and Coron et al. [CDMP05]. This indifferentiability result implies that the hash function based on an ideal primitive behaves like a random oracle. It thus provides security guarantees regarding any one-stage generic attack [RSS11], at least up to $2^{c/2}$ queries (the scheme may achieve higher security against specific attacks, see, e.g., [LM22]).

The $2^{c/2}$ indifferentiability bound of the sponge construction is tight. In particular, in order to obtain a sponge-based hash function with k bits of security, one must use a permutation of size at least $2k + 1$ bits. On the other hand, permutation-based *keyed* applications achieve a higher level of security. A long line of research on keyed applications of the sponge [BDPV07, BDPV11b, CDH+12, ADMV15, GPT15, MRV15, NY16, DMP22] and duplex [BDPV11a, JLM14, SY15, MRV15, DMV17, DM19, JLM+19, CJN20, Men23] has demonstrated that typical sponge-/duplex-based encryption, authentication, and authenticated encryption can achieve around $\min\{2^c/M, 2^k\}$ security, where k is the key size and M the online complexity (the number of keyed evaluations). Clever use of domain separators can even push security to $\min\{2^c, 2^k\}$ (see, e.g., [Men23, Corollary 1]).

The gap between $2^{c/2}$ security in keyless applications and $2^c/M$ security in keyed applications (assuming the key is long enough) is insignificant in case a general purpose permutation such as Keccak-f[1600] [BDPV11b] is available. However, when only small permutations are available, the situation changes drastically. Suppose, for the sake of example, we have the ISO/IEC standardized permutation P_{256} of the PHOTON family [GPP11] at our disposal (a comparable example can be given for ISO/IEC standardized Spongent [BKL+11]). If the permutation is used in duplex-based authenticated encryption, we obtain generic 2^{192} security provided that the online complexity is bounded by 2^{64} blocks [Men23, Section 9]. In a keyless setting, however, in the plain sponge construction this permutation yields 2^{127} security at best! The contrast becomes even more pronounced for smaller permutations. For example, NIST Lightweight Cryptography [NIS19] finalist Elephant [BCDM20] is a permutation-based authenticated scheme instantiated with three different permutations of sizes respectively 160, 176, and 200 bits. Assuming that the online complexity is bounded by 2^{50} bytes (the limit as stated in the call for proposals of the NIST Lightweight Cryptography competition), Elephant achieves 112, 127, and 127 bits of security in the random permutation model. On the other hand, the Elephant finalist did *not* offer a hashing functionality, the reason simply being that the three permutations were too small for hashing [BCDM20, Section 1].

This issue is not unique for the sponge: alternative hash functions such as the modes of Grindahl [KRT07], the SHA-3 finalists Grøstl [GKM+11] and JH [Wu11], or the generalized parazoa framework [AMP12] achieve security up to $b/2$ bits at best (see also Table 2 later on).

This leads to a fundamental question as to whether it is possible to obtain more than $b/2$ bits of security for a hash function based on a permutation of size b bits, i.e., to beat the birthday bound in the underlying permutation size. Note that this is not just a theoretical question: a construction of this type would allow to hash with smaller permutations, and it would allow to reduce the gap between the required permutation sizes for keyed and keyless permutation-based constructions for a given security level.

1.1 Compression Functions Based on Permutations

To answer this question, one direction is to search for a secure permutation-based compression function. Indeed, to facilitate the design of hash functions, one can start from a robust compression function upon which one applies a domain extender to turn it into a hashing mode. Yet the question of finding "good" compression functions based on permutations is much more challenging than for block ciphers, since as opposed to the latter, permutations are non-compressing primitives. Stam's conjecture [Sta08], later proved by Steinberger [Ste10] and Steinberger et al. [SSY12], implies that for a compression function compressing m bits, collision resistance of 2^s requires at least $\frac{s+m}{b-s}$ different b-bit permutation calls per compression function call, which is hard to reach in practice. In addition, most of the existing literature regarding permutation-based compression functions [RS08, SS08, MP12] only focus on collision resistance and preimage resistance. In particular, no compression function construction is known to guarantee security in the indifferentiability framework. On top of that, looking at the sponge, it has an insecure compression function, suggesting that achieving security at the compression function level is overkill.

1.2 Double Block Length Hashing Based on a Block Cipher

Instead of focusing on designing a secure compression function, we rather take inspiration from the double block length hashing design rationale, an idea that dates back to Meyer and Schilling with the design of MDC-2 and MDC-4 [MS88]. It consists of using a state twice as large as the primitive size with at least two primitive calls per compression function call. In block cipher-based hashing, the literature studying compression functions for double block length hashing is vast [MS88, Sta08, Sta09, LM92, Hir06, Hir04, ÖS09, LS15, JÖS12, Men12, LSS11a, LK11, FGL09, KP97, Men14, AFK+11, LSS11b]. All of these constructions have been proven collision/preimage resistant, but none of them is indifferentiable (or only with a weak bound), as shown in [Men13]. Thus a dedicated proof is required to prove indifferentiability at the hash function level. Examples of proven indifferentiable hash function constructions include Naito's construction using Hirose's compression function without feed-forward [Nai17], MDPH [Nai19], used by the NIST lightweight cryptography finalist Romulus [IKMP20], and EXEX-NI from Naito et al. [NSS21]. The latter scheme uses an invertible (thus insecure) and efficient compression function, and the construction is indifferentiable up to $n - \log(n)$ bits, where n denotes the block size. We will look into the same direction, i.e., aim at finding an efficient double block length hashing scheme based on permutations with an insecure compression function.

1.3 Our Contribution

In detail, we propose a double block length hashing mode based on two permutations of size $b = r + c$ bits, called the "double sponge". The double sponge is described in detail in Section 3 and depicted in Fig. 1. The scheme can be seen as two sponges whose states are blended together after each absorption of a message block, using an efficient mixing function. Similarly to most double block length hashing schemes, the double sponge processes the same message block through two different permutation calls. As before, its security and efficiency depend on the parameters r and c, which are called rate and capacity, respectively, as before.

An indifferentiability proof of the double sponge up to a bound of $2c/3$ bits is given in Section 4. The simulator introduced for this proof is designed to be indistinguishable from a random permutation up to $2^{2c/3}$ queries, as we demonstrate in Lemma 4. The major challenge in the design of the simulator is to guarantee that it provides answers matching the ones of the random oracle. Indeed, as we target a security bound beyond $2^{c/2}$, inner collisions may occur and the simulator has to deal with them. This problem is resolved mostly by allowing the simulator to selectively define certain (but a limited number of) queries in advance, and by defining sophisticated bad events taming the inner collisions. The simulator is described in detail in Section 4.2, including an extensive rationale.

In Section 6, we describe a differentiability attack with respect to our simulator which succeeds after approximately $2^{\frac{2c+r}{3}}$ queries. The attack gets close to the security bound of the double sponge, but admittedly leaves a (small) gap of $2^{\frac{r}{3}}$ queries. This gap is likely caused by restrictions we had to impose to avoid inner multi-collisions. Releasing this conditions, i.e., allowing inner multi-collisions, would incur a more complex simulator tree structure and new, even more sophisticated, bad events. In Section 7 we conclude the work, and in particular discuss in more detail the (im-)possibilities to improve the differentiability attack or the indifferentiability proof.

1.4 Comparison with Existing Hashing Modes

The double sponge allows to have a hash function using a b-bit permutation while it was not possible with existing permutation-based constructions when the target level of security κ is between $\frac{1}{2}(b-1)$ and $\frac{2}{3}(b-1)$. In particular, for $\kappa = 112$ (corresponding to NIST Lightweight Cryptography requirements), one needs a double sponge with $c \geq 168$ (as opposed to $c \geq 224$ for the plain sponge). This concretely means that if one has at their disposal only the ISO/IEC standardized 176-bit Spongent, or the 196-bit PHOTON permutation, then one can use them with the DS and attain 112 bits of security. One can even use smaller variants of these standardized primitives and still attain a decent level of security, although slightly below $\kappa = 112$. Moreover, for certain parameter sets, using a DS can improve the security, without sacrificing the number of bits compressed per permutation call. More precisely, for a plain sponge with a rate of r bits and primitive size b bits, when $r \leq \frac{b}{5}$, then using DS with a rate of $2r$ improves the security bound. Of course, this security improvement does

Table 1: Comparison of the double sponge (DS) with several double block length hashing modes. r and c denote respectively the rate and capacity used in the DS. n and k denote respectively the block size and the key size of the block cipher-based hashing modes. The security bound is in bits, and holds in the indifferentiability framework. In the security bound of block cipher-based modes, logarithmic factors in n are omitted.

Mode	# Primitives per compression	Compression rate (ρ)	Security bound (κ)	State size	Primitive size		Note	Reference
					input	output		
Mennink's in, e.g., ChopMD	3	n	$\frac{n}{2}$	$4n$	$2n$	n	$k = n$	[Men13]
MDPH	2	$k - n$	n	$3n + k$	$n + k$	n	$k > n$	[Nai19, GIM22]
EXEX-NI	2	$k - n$	n	$n + k$	$n + k$	n	$k \geq 2n$	[NSS21]
DS	2	r	$\frac{2c}{3}$	$2(r + c)$	$r + c$	$r + c$		This work

not come for free: our construction requires two distinct permutations,[1] a state twice as large, and additionally two multiplications by 2 and additions in $\mathrm{GF}(2^b)$ per compression function evaluation.

1.4.1 Comparison with Double Block Length Hashing Modes

In Table 1, we compare DS with the most notable double block length hashing modes that were proven to be indifferentiable from a random oracle: Mennink's indifferentiable double block length compression function in any suitable indifferentiable hashing mode [Men13], MDPH [Nai19], and EXEX-NI [NSS21]. In this discussion we exclude Naito's mode on top of Hirose's compression function [Nai17] and the MDC-4 [MS88] compression function in any suitable hashing mode [Men13] as they give worse numbers. We also do not include hashing modes that were only proven collision resistant; for this, we refer to the discussion of Naito et al. [NSS21, Table 1]. It is important to note that block ciphers are compressing primitives, and for MDPH and EXEX-NI we require the key size to be larger than the block size ($k > n$ and $k \geq 2n$, respectively). Henceforth, in our comparison we consider the primitive *input* size, which equals $k + n$ for block cipher-based modes and $r + c$ for the DS.

If we restrict our focus to a fixed security bound κ and a fixed rate ρ, MDPH and EXEX-NI require a primitive of size $\rho + 2\kappa$ bits whereas DS requires a primitive of size $\rho + \frac{3}{2}\kappa$ bits. Mennink's construction is restricted to $2\kappa = \rho = n$, in which case the primitive input size is 4κ as opposed to $\frac{7}{2}\kappa$ for DS.

The gain comes at a cost in state size. Again fixing κ and ρ, EXEX-NI requires a state size of $\rho + 2\kappa$ bits (that mode is specifically designed for having a low memory size) as opposed to $2\rho + 3\kappa$ for DS. MDPH has a state size of $\rho + 4\kappa$, which is better than DS only if $\rho \geq \kappa$.

1.4.2 Comparison with Permutation-Based Hashing Modes

We replicate our analysis in Table 2, focusing this time on permutation-based hashing modes that have a proven indifferentiability bound. We draw the comparison between DS on the one hand and the sponge [BDPV07, BDPV08], Grøstl [GKM+11, AMP10], and JH [Wu11, BMN10, MPS16] on the other hand. We do not include the Grindahl [KRT07, AMP12] and PHOTON [GPP11, NO14] modes, as they have identical metrics to the sponge, nor the general parazoa framework [AMP12].

It is important to note that for DS, as well as for Grøstl and JH, the proven security bound κ is not tightly matching the best attack λ. Nevertheless, regardless of whether we

[1] However, we discuss the possibilities of using a single permutation in Section 4.5.

Table 2: Comparison of the double sponge (DS) with several permutation-based hashing modes. r and c denote respectively the rate and capacity used in the plain sponge. The best known attack and security bound are in bits, and the latter holds in the indifferentiability framework.

Mode	# Primitives per compression	Compression rate (ρ)	Security bound (κ)	Security attack (λ)	State size	Primitive size	Reference
Sponge	1	r	$\frac{c}{2}$	$\frac{c}{2}$	$r+c$	$r+c$	[BDPV07, BDPV08]
Grøstl	2	b	$\frac{b}{4}$	$\frac{b}{2}$	$3b$	b	[GKM+11, AMP10]
JH	2	$\frac{b}{2}$	$\frac{b}{4}$	$\frac{b}{2}^{(*)}$	b	b	[Wu11, BMN10, MPS16]
DS	2	r	$\frac{2c}{3}$	$\frac{2c+r}{3}$	$2(r+c)$	$r+c$	This work

$^{(*)}$: [MPS16] mentions that the indifferentiability bound on JH is not tight and suggests that it could possibly be improved further to $\frac{b}{2}$ bits.

consider the the best known attack or the security bound, the DS outperforms the sponge in terms of permutation size. In detail:

– for fixed security bound κ and a fixed rate ρ, the sponge requires a primitive of size $\rho + 2\kappa$ bits whereas DS requires a primitive of size $\rho + \frac{3}{2}\kappa$ bits;
– for fixed security attack λ and a fixed rate ρ, the sponge requires a primitive of size $\rho + 2\lambda$ bits whereas DS requires a primitive of size $\frac{\rho}{2} + \frac{3}{2}\lambda$ bits.

The improvement of DS over the sponge comes at a cost in extra state size: DS operates on a state of $2\rho + 3\kappa$ bits as opposed to $\rho + 2\kappa$ for the sponge.

Grøstl is restricted to $\rho = 4\kappa$, in which case the permutation size is 4κ too, as opposed to $\frac{11}{2}\kappa$. Thus, for the restricted regime of Grøstl, Grøstl outperforms the other schemes. JH is restricted to $2\kappa = \rho = \frac{b}{2}$, in which case the permutation size is 4κ as opposed to $\frac{7}{2}\kappa$ for DS. If we consider instead the best known attack λ, JH is restricted to $\lambda = \rho$, in which case both JH and the DS require a primitive of size 2λ bits. Therefore, there is little to no difference between JH and DS. These results may not come as a surprise as both Grøstl and JH have a large absorption rate, and cannot be proven secure beyond the birthday bound in the permutation size.

2 Preliminaries

2.1 Notation

In the following, we use $x := y$ to define x as being equal to y. Let $\{0,1\}^*$ be the set of binary strings of arbitrary length. Given $X \in \{0,1\}^*$, $X[a_1 : a_2]$ denotes the bits of X between positions a_1 and a_2 (latter excluded). For $b, r, c \in \mathbb{N}$ with $b := r + c$, and $|X| = b$, the outer part of X is $\text{outer}_r(X) := X[0 : r]$, and the inner part is $\text{inner}_c(X) := X[r : b]$. For $X, Y \in \{0,1\}^b$, $X \overset{c}{=} Y$ means that $\text{inner}_c(X) = \text{inner}_c(Y)$.

If S is a finite set, $x \overset{\$}{\leftarrow} S$ means that x is sampled uniformly at random from S. Vectors are denoted with bold letters. If $\boldsymbol{x}$ is a vector with k coordinates, then for any $i \in \{1, \dots, k\}$, $\boldsymbol{x}_i$ is the i^{th} element of $\boldsymbol{x}$. We consider random oracles taking as input arbitrary length messages and outputting random streams [BR93]. For a random oracle $\mathcal{RO}$ and a message M, $\mathcal{RO}(M) \in \{0,1\}^*$ denotes the stream. If **evt** is an event, for any $i \in \{1, \dots, q\}$, $\mathbf{evt}_i$ denotes that **evt** is triggered after i queries. Moreover, if P_1 and P_2 are two probability distributions, for $k \in \{1, 2\}$, $\mathbf{Pr}_k(\mathbf{evt})$ denotes the probability that **evt** is set with the distribution P_k. If S is a set of size 2, then for $s \in S$, $\bar{s}$ is the unique element in S different from s. In the algorithms, the procedures are highlighted in red, the comments are in blue and formatted using the C language syntax. Finally, given a dictionary `dict` with keys in a set S, $\mathbf{Img}(\texttt{dict}) := \{\texttt{dict}[x] \mid x \in S\} \setminus \{\perp\}$, $\mathbf{Dom}(\texttt{dict}) := \{x \in S \mid \texttt{dict}[x] \neq \perp\}$.

2.2 The Sponge Construction

Since our construction is based on the plain sponge, we briefly discuss this construction in an XOF mode based on a permutation $\mathcal{P} : \{0,1\}^b \to \{0,1\}^b$, where $b \in \mathbb{N}$. Let $c, r \in \mathbb{N}$ be such that $b = r + c$, $M \in \{0,1\}^*$ be an input message, and $IV \in \{0,1\}^b$ be any fixed initialization vector. The sponge construction requires messages to be of length a multiple of r bits, and such that the last message block is not zero. For that reason we require an injective padding function *pad* which transforms the messages into r-bit blocks with a non-zero last message block. An example of such padding consists of appending to M a one and as many zeros as required to obtain a message of length divisible by r. The sponge construction instantiated with the permutation $\mathcal{P}$, and with domain and codomain $\{0,1\}^*$, is now defined as follows.

1. *Initialization.* M is padded and decomposed into r-bit blocks, and the state of the sponge is initialized by IV;
2. *Absorbing phase.* Every message block is XORed to the upper r bits of the state, and each addition is interleaved with an application of $\mathcal{P}$ to the state;
3. *Squeezing phase.* Once all message blocks have been absorbed, the stream is extracted by blocks of r bits by retrieving the upper r bits of the state. Every extraction is again interleaved by the application of $\mathcal{P}$.

This construction is formalized in Algorithm 1.

Algorithm 1: The sponge construction. The algorithm takes as input the message to hash M and the number of bits of the stream to extract n. $IV \in \{0,1\}^b$ is a fixed initialization vector.

```
Function Sponge^P(M, n): $\{0,1\}^* \times \mathbb{N} \to \{0,1\}^*$
    /* The state $S$ is initialized and the message is padded and split into
       $r$-bit blocks */
    $S \leftarrow IV$;
    $M_1 \| \cdots \| M_k \leftarrow pad(M)$;
    /* Absorbing phase */
    for $i = 1, \ldots, k$ do
        $S \leftarrow \mathcal{P}(S \oplus (M_i \| 0^c))$;
    end
    /* Squeezing phase */
    for $i = 1, \ldots, \lceil n/r \rceil$ do
        $Z_i \leftarrow \mathrm{outer}_r(S)$;
        $S \leftarrow \mathcal{P}(S)$;
    end
    return $(Z_1 \| \ldots \| Z_{\lceil n/r \rceil})[0 : n]$
end
```

2.3 Security Model

2.3.1 Indistinguishability Let S_0, S_1 be two systems. Consider a distinguisher $\mathcal{D}$ which interacts with S_k, for $k \xleftarrow{\$} \{0,1\}$. $\mathcal{D}$ is allowed to make at most q queries book-kept in the so-called query history, and outputs an element in $\{0,1\}$, reflecting with which system it believes to interact with. The advantage of $\mathcal{D}$ is defined as

$$\mathbf{Adv}^{\mathrm{ind}}(S_0, S_1)(\mathcal{D}) = \left|\mathbf{Pr}\left(\mathcal{D}^{S_0} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{S_1} = 1\right)\right| .$$

Moreover, $\mathbf{Adv}^{\mathrm{ind}}(S_0, S_1)(q)$ denotes the supremum of the set of $\mathbf{Adv}^{\mathrm{ind}}(S_0, S_1)(\mathcal{D})$, over all distinguishers $\mathcal{D}$ making at most q queries to the system (note that the quantity q might be refined when S_0 and S_1 give access to several oracles).

2.3.2 Indifferentiability

Indifferentiability is a special type of distinguishing game, where the adversary has access to a keyless primitive. This framework was introduced by Maurer et al. [MRH04], and refined in the context of hash functions by Coron et al. [CDMP05]. Let $\mathcal{H}^{\mathcal{P}}$ be the hash function using an invertible ideal primitive $\mathcal{P}$, and $\mathcal{RO}$ be a random oracle with the same domain and co-domain as $\mathcal{H}$. Let $\mathcal{S} := (\mathcal{S}^{fwd}, \mathcal{S}^{inv})$ be a simulator allowed to make at most $q_{\mathcal{S}}$ queries to $\mathcal{RO}$ such that both $\mathcal{S}^{fwd}$ and $\mathcal{S}^{inv}$ have the same domain and co-domain as $\mathcal{P}$. Let $\mathcal{D}$ be a distinguisher making at most $q_{\mathcal{H}}$ construction queries and $q_{\mathcal{P}}$ primitive queries. The indifferentiability advantage of $\mathcal{D}$ with respect to the simulator $\mathcal{S}$ is defined as follows:

$$\mathbf{Adv}^{\mathrm{indif}}_{\mathcal{H}^{\mathcal{P}},\mathcal{S}}(\mathcal{D}) = \mathbf{Adv}^{\mathrm{ind}}\left([\mathcal{H}^{\mathcal{P}}, (\mathcal{P}, \mathcal{P}^{-1})], [\mathcal{RO}, (\mathcal{S}^{fwd}, \mathcal{S}^{inv})]\right)(\mathcal{D})\,.$$

Moreover, $\mathbf{Adv}^{\mathrm{indif}}_{\mathcal{H}^{\mathcal{P}},\mathcal{S}}(q_{\mathcal{P}}, q_{\mathcal{H}})$ denotes the supremum of the set of $\mathbf{Adv}^{\mathrm{indif}}_{\mathcal{H}^{\mathcal{P}},\mathcal{S}}(\mathcal{D})$, over all distinguishers $\mathcal{D}$ making $q_{\mathcal{P}}$ primitive queries and $q_{\mathcal{H}}$ construction queries. Sometimes, this notion is refined. For example, in our case, calling $\mathcal{H}^{\mathcal{P}}$ with a message M has a practical cost which depends on the size of M (after padding, see Section 3.1), and the desired number of bits of the stream to extract. Then, we define $\mathbf{Adv}^{\mathrm{indif}}_{\mathcal{H}^{\mathcal{P}},\mathcal{S}}(q_{\mathcal{P}}, \sigma)$ to be the supremum of the set of $\mathbf{Adv}^{\mathrm{indif}}_{\mathcal{H}^{\mathcal{P}},\mathcal{S}}(\mathcal{D})$, over all distinguishers $\mathcal{D}$ making $q_{\mathcal{P}}$ primitive queries and construction queries that would require in total σ primitive queries in the world $[\mathcal{H}^{\mathcal{P}}, (\mathcal{P}, \mathcal{P}^{-1})]$.

In the following we define $\mathcal{RO}$-consistency, which captures the fact that the simulator answers match the $\mathcal{RO}$ outputs. This is an essential property to guarantee indifferentiability of the construction.

Definition 1. *A simulator $\mathcal{S}$ for an iterated XOF construction is $\mathcal{RO}$-consistent if $\forall i \in \mathbb{N}, \forall M \in \{0,1\}^*$, whenever one can compute $\mathcal{H}^{\mathcal{S}}(M)[0:i]$ from the query history to the simulator, then it equals $\mathcal{RO}(M)[0:i]$.*

Note that the $\mathcal{RO}$-consistency can be guaranteed under certain conditions, e.g., as long as a bad event does not occur.

3 Double Sponge

In this section we describe our hashing mode, and prove its indifferentiability in Section 4.

3.1 Description

Let $b, r, c \in \mathbb{N}$, with $b = r + c$. On a high level view, the double sponge can be seen as two sponges with a state mixing at each iteration. The construction operates on a state of size $2b$ bits, and requires two cryptographic permutations $\mathcal{P}_{top}$ and $\mathcal{P}_{bot}$ over b bits. One absorption call enables to process r bits of the padded message, and during the squeezing phase, r bits of the digest are extracted at a time. The top and bottom permutations $\mathcal{P}_{top}$ and $\mathcal{P}_{bot}$ will be considered distinct, independent, and uniformly random in the indifferentiability proof. The mixing is performed by applying to the state a 2×2 matrix with coefficients over $\mathrm{GF}(2^b)$, defined as follows:

$$MIX = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}. \tag{1}$$

Our construction is defined in Algorithm 2 and depicted in Fig. 1.

Algorithm 2: The double sponge construction based on $\mathcal{P} := (\mathcal{P}_{top}, \mathcal{P}_{bot})$. The algorithm takes as input the message to hash M and the number of bits of the stream to extract n. *pad* is an injective padding with the same restrictions as for the sponge construction (Section 2.2). IV^{top} and IV^{bot} are two fixed initialization vectors.

```
Function $\mathcal{H}^{\mathcal{P}}(M, n)$: $\{0,1\}^* \times \mathbb{N} \to \{0,1\}^*$
    /* The state is initialized and the message is padded and split into $r$-bit
       blocks */
    $(S^{top}, S^{bot}) \leftarrow (IV^{top}, IV^{bot})$;
    $M_1 \| \cdots \| M_k \leftarrow pad(M)$;
    /* Absorbing phase */
    for $i = 1, \ldots, k$ do
        $(S^{top}, S^{bot}) \leftarrow MIX\left(\mathcal{P}_{top}(S^{top} \oplus (M_i \| 0^c)), \mathcal{P}_{bot}(S^{bot} \oplus (M_i \| 0^c))\right)$;
    end
    /* Squeezing phase */
    for $i = 1, \ldots, \lceil n/r \rceil$ do
        $Z_i \leftarrow \mathrm{outer}_r\left((S^{top}, S^{bot})\right)$;
        $(S^{top}, S^{bot}) \leftarrow MIX\left(\mathcal{P}_{top}(S^{top}), \mathcal{P}_{bot}(S^{bot})\right)$;
    end
    return $(Z_1 \| \ldots \| Z_{\lceil n/r \rceil})[0 : n]$
end
```

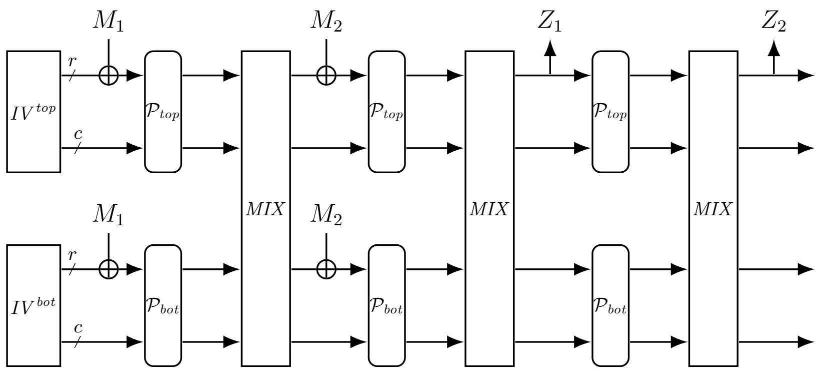


Fig. 1: The double sponge. In this example, $M_1 \| M_2$ is the padded message to hash.

3.2 Design Rationale

For simplicity of the proof, the double sponge uses two independent permutations, but we discuss in Section 4.5 a tweak of the double sponge to use the same permutation. The top and bottom parts of the double sponge should be entangled at the compression function level, as otherwise one can exploit the independency of the top and bottom parts, and find attacks like the ones on the iterated concatenated combiners [Jou04]. For that reason, we require the *MIX* matrix to be MDS, and such that the two coefficients in both rows are not equal. Given these constraints, we opted for a simple choice, i.e., (1). Note that the coefficient 2 in this matrix can be any non-zero coefficient in $\mathrm{GF}(2^b)$ different from 1.

The message is absorbed at both the top *and* bottom parts of the state. This ensures that it influences both top and bottom input permutation calls. It is not possible to absorb two different message blocks during one compression function call, since from i top and bottom queries, one can then build i^2 different states (i.e., rooted nodes, see Section 3.3). The probability of finding full-state collisions would increase to the birthday bound in the capacity. Finally, it is not possible either to squeeze both the top and bottom parts, because looking ahead, doing so makes the linear equation generated by the simulator unsolvable, see point 2 of Section 4.2.

3.3 Graph Representation

In this section we provide a graph representation of the states of the double sponge similarly to the sponge construction. Assume in the following that the permutation outputs are lazily sampled, so that we only know a fraction of the permutation evaluations. The set of nodes is $V = \{0,1\}^{2b}$, and elements are represented using the notation (A^{top}, A^{bot}), where $A^{top} \in \{0,1\}^b$ is called the top part and $A^{bot} \in \{0,1\}^b$ the bottom part. For $(A^{top}, A^{bot}), (B^{top}, B^{bot}) \in V$, and $m \in \{0,1\}^r$, we add the edge $(A^{top}, A^{bot}) \xrightarrow{m} (B^{top}, B^{bot})$ whenever

$$MIX\left(\mathcal{P}_{top}(A^{top} \oplus (m\|0^c)), \mathcal{P}_{bot}(A^{bot} \oplus (m\|0^c))\right) = (B^{top}, B^{bot}),$$

and the two underlying permutation evaluations are defined. A special subset of the nodes is the set of *rooted nodes*, where $(A^{top}, A^{bot}) \in \texttt{Rooted}$ whenever (A^{top}, A^{bot}) is accessible from (IV^{top}, IV^{bot}). Given a rooted node (A^{top}, A^{bot}), one can retrieve the unique[2] XOF call $\mathcal{H}(M)[i : i+r]$ which is associated to this state. Finally, we refer to *partial edges* when there exists $(A^{top}, A^{bot}) \in \texttt{Rooted}, m \in \{0,1\}^r$, and $s \in \{top, bot\}$ such that $\mathcal{P}_s(A^s \oplus (m\|0^c))$ is known, but $\mathcal{P}_{\bar{s}}(A^{\bar{s}} \oplus (m\|0^c))$ is not known.

4 Indifferentiability of the Double Sponge

In this section, we prove that the double sponge is indifferentiable from an $\mathcal{RO}$ with a bound of $\mathcal{O}\left(\frac{q^{\frac{3}{2}}}{2^c}\right)$, as stated in Theorem 1. In Section 4.5 we discuss an extension of our result when using a single permutation.

Theorem 1. *Let $\mathcal{H}^{\mathcal{P}}$ be the construction from Section 3, where $\mathcal{P} := (\mathcal{P}_{top}, \mathcal{P}_{bot})$. Let $q := \sigma + q_{\mathcal{P}}$. If $30 < 2^c$ and $3q < 2^c$. There exists a simulator $\mathcal{S}$ such that*

$$\mathbf{Adv}^{\mathrm{indif}}_{\mathcal{H}^{\mathcal{P}},\mathcal{S}}(q_{\mathcal{P}}, \sigma) \leq \frac{40q^{\frac{3}{2}}}{2^c - 3q}.$$

Moreover, $\mathcal{S}$ makes a total of at most $3q_{\mathcal{P}}$ random oracle queries, and its overall memory storage and runtime are linear in $q_{\mathcal{P}}$.

The proof of this theorem is given in the remainder of this section.

4.1 General Setting of the Proof and Outline

The adversary must distinguish between the real world $W_R := (\mathcal{H}^{\mathcal{P}}, \mathcal{P}, \mathcal{P}^{-1})$ and the ideal world $W_I := (\mathcal{RO}, \mathcal{S}^{fwd}, \mathcal{S}^{inv})$. To do so, it can perform construction queries by using the procedure $\texttt{ConsQuery}()$, which takes as input a message $M \in \{0,1\}^*$, and the index of the stream $k \in \mathbb{N}$, corresponding to $\mathcal{RO}(M)[k \times r : k(r+1)]$ in the ideal world or the kth squeeze call after absorbing M in the real world. Forward (resp., inverse) queries are made via the interface $\texttt{FwdPQuery}()$ (resp., $\texttt{InvPQuery}()$). Both take as input an element $s \in \{top, bot\}$ and the element to query in $\{0,1\}^b$. To prove indifferentiability, we use the simulator described in Section 4.2 which defines on-the-fly permutation-consistent answers and logs its responses in two dictionaries $\texttt{tabP}_{top}$ and $\texttt{tabP}_{bot}$. From these two tables, it is able to keep track of the graph representation defined in Section 3.3. For simplicity of

[2] This call is unique as long as there is no collision on $2c + r$ bits between two rooted nodes.

the proof, we require that the simulator's graph does not contain partial edges. In other words, if (A^{top}, A^{bot}) is a rooted node, and there exists $m \in \{0,1\}^r, s \in \{top, bot\}$ such that $\texttt{tabP}_s[A^s \oplus (m\|0^c)] \neq \bot$, then *necessarily*, $\texttt{tabP}_{\bar{s}}[A^{\bar{s}} \oplus (m\|0^c)] \neq \bot$. In the real world, we assume that the primitive oracle decides outputs using lazy sampling. Thus, in order to ease the comparison between the real world and the ideal world, we introduce world W_{IM2} illustrated in Fig. 3c (see Fig. 3 for a depiction of all worlds considered in this proof), which implements the real world with a supplementary layer called `GraphProc`, ensuring that the graph deducible from the query history of `GraphProc` does not contain partial edges either. In worlds W_{IM2} and W_I, we implicitly consider that the collateral permutation outputs are given *for free*. Now, W_{IM2} and W_I can be differentiated in two ways: consistency with $\mathcal{RO}$, and the statistical difference between the simulator's answers with the ones of a random primitive. Each of these points is addressed separately, thanks to the introduction of another intermediate world W_{IM1}, as illustrated in Fig. 3b. A similar game splitting was done among others in the indifferentiability proof of PHOTON [NO14]. Section 4.3 introduces bad events for the $\mathcal{RO}$-consistency. Section 4.4 explains more rigorously the intermediate worlds W_{IM1} and W_{IM2}. This part also splits the underlying probability computation, which is addressed in detail in Section 5. This eventually leads to the proof's conclusion.

4.2 Simulator Definition

In this section we provide the high level definition of our simulator, with its formal procedures provided in Algorithm 3.

4.2.1 On Forward Query

For any query `FwdPQuery` (top, X) or query `FwdPQuery` (bot, X), the simulator goes through several phases, as explained in the following.

1. Collection of nodes to expand. The simulator first collects all rooted nodes *after absorption of a message block* where it needs to provide $\mathcal{RO}$-consistent answers. The goal of this procedure is to guarantee that at the end of the query execution, no partial edges exist on the simulator graph. Because the number of queries can go beyond $2^{c/2}$, it is not unlikely that some rooted nodes collide on their top or bottom inner part, thus more than one node may be collected during this step. Example 1 gives a minimal example of such a case.

Example 1. Let $(A^{top}, A^{bot}), (B^{top}, B^{bot}) \in \texttt{Rooted}$ be two distinct nodes such that $A^{bot} \overset{c}{=} B^{bot}$. Assume that the query is `FwdPQuery` $(top, A^{top} \oplus (m\|0^c))$ for $m \in \{0,1\}^r$, and let $m' := m \oplus \text{outer}_r\left(A^{bot} \oplus B^{bot}\right)$. Then the simulator needs to complete an edge from (A^{top}, A^{bot}) with the message m, i.e., decide on $\texttt{tabP}_{top}[A^{top} \oplus (m\|0^c)]$ and $\texttt{tabP}_{bot}[A^{bot} \oplus (m\|0^c)]$. Nevertheless, since $A^{bot} \oplus (m\|0^c) = B^{bot} \oplus (m'\|0^c)$, the query will then also fix $\texttt{tabP}_{bot}[B^{bot} \oplus (m'\|0^c)]$, so that to avoid a partial edge starting from (B^{top}, B^{bot}), the simulator also needs to decide $\texttt{tabP}_{top}[B^{top} \oplus (m'\|0^c)]$.

On a high level view, this procedure can be depicted using a depth-first search algorithm. Indeed, consider a graph (maintained by the simulator, different from the graph representation) where the nodes V' are

$$V' = \left\{(A^{top} \oplus (m\|0^c), A^{bot} \oplus (m\|0^c)) \mid m \in \{0,1\}^r \wedge (A^{top}, A^{bot}) \in \texttt{Rooted}\right\},$$

i.e., the rooted nodes after absorbing all possible message blocks. Then for $s \in \{top, bot\}$, we add the edge $(X_1^{top}, X_1^{bot}) \xrightarrow{s} (X_2^{top}, X_2^{bot})$ whenever $X_1^s = X_2^s$. If the query was `FwdPQuery` (s, X), let **NodesToVisit** $= \{(Y^{top}, Y^{bot}) \in V' \mid Y^s = X\}$. The node collection phase then computes all the nodes accessible from **NodesToVisit**, and returns in the set **NodesToAnswer** all of the nodes found.

To summarize, for every $(X^{top}, X^{bot}) \in$ **NodesToAnswer**, the simulator needs to determine $\texttt{tabP}_{top}[X^{top}]$ and $\texttt{tabP}_{bot}[X^{bot}]$. If the query does not impact any node (i.e.,

Algorithm 3: Simulator $\mathcal{S}$ main functions. UpdateGraph () takes care of updating the set Rooted according to $\texttt{tabP}_{top}$ and $\texttt{tabP}_{bot}$. LinSolve () is described in paragraph 2 from Section 4.2.

```
1  Function Init ()
2      Rooted ← $\{(IV^{top}, IV^{bot})\}$;
3      $\texttt{tabP}_{top}$ ← EmptyDictionary ();
4      $\texttt{tabP}_{bot}$ ← EmptyDictionary ();
5  end
6  Function $\mathcal{S}^{fwd}$ $(s, X)$: $\{top, bot\} \times \{0,1\}^b \to \{0,1\}^b \cup \{\bot\}$
7      if $\texttt{tabP}_s[X] \neq \bot$ then
8          return $\texttt{tabP}_s[X]$;
9      end
10     NodesToAnswer ← NodeCollection $(s, X)$;
11     if NodesToAnswer $= \emptyset$ then // Query is not on the graph
12         $\texttt{tabP}_s[X] \xleftarrow{\$} \{0,1\}^b \setminus \mathbf{Img}(\texttt{tabP}_s)$;
13         return $\texttt{tabP}_s[X]$;
14     end
15     $\texttt{DictOuterParts}_{top}, \texttt{DictOuterParts}_{bot}$ ← LinSolve (NodesToAnswer);
16     if $\exists s, \texttt{DictOuterParts}_s = \bot$ then // Linear system solving failed
17         return $\bot$;
18     end
19     PermConsistentOutputTop ($\texttt{DictOuterParts}_{top}$);
20     PermConsistentOutputBot ($\texttt{DictOuterParts}_{bot}$);
21     UpdateGraph ();
22     EnsureNoPartialEdges ();
23     return $\texttt{tabP}_s[X]$;
24 end
25 Function $\mathcal{S}^{inv}$ $(s, Y)$: $\{top, bot\} \times \{0,1\}^b \to \{0,1\}^b$
26     if $\exists X \in \{0,1\}^b, \texttt{tabP}_s[X] = Y$ then
27         return $X$;
28     end
29     $X \xleftarrow{\$} \{0,1\}^b \setminus \mathbf{Dom}(\texttt{tabP}_s)$;
30     $\texttt{tabP}_s[X] = Y$;
31     UpdateGraph ();
32     EnsureNoPartialEdges ();
33     return $X$;
34 end
```

NodesToAnswer $= \emptyset$), the simulator simply returns a uniform permutation-consistent answer. In Algorithm 3, the node collection procedure is invoked in line 10, and the full procedure is described in Algorithm 4.

2. Linear system solving. Let $(X_1^{top}, X_1^{bot}), (X_2^{top}, X_2^{bot}), \ldots, (X_m^{top}, X_m^{bot})$ be all nodes in **NodesToAnswer**. By construction of the latter set, whenever $m > 1$, every (X_i^{top}, X_i^{bot}) is colliding on its top or bottom part to another (X_j^{top}, X_j^{bot}) in this set. Thanks to the last message block not being equal to zero due to padding, and as long as no full-state collision occurred, for any $(X^{top}, X^{bot}) \in$ **NodesToAnswer**, the simulator knows exactly what $\mathcal{RO}$ calls are necessary to provide consistent answers, i.e., $M \in \{0,1\}^*, k \in \mathbb{N}$ such that in the real world, the knowledge of $\mathcal{P}_{top}(X^{top})$ and $\mathcal{P}_{bot}(X^{bot})$ gives $\mathcal{H}(M)[k : k + r]$. Let $h_1, \ldots, h_m \in \{0,1\}^r$ be the $\mathcal{RO}$ answers corresponding to the nodes in **NodesToAnswer**.

Algorithm 4: Simulator sub-routine to collect the nodes.

```
1  Function NodeCollection($(s, x)$): $\{top, bot\} \times \{0,1\}^b \to \texttt{Set}$
2      NodesToVisit $\leftarrow \{(A^{top} \oplus (m \| 0^c), A^{bot} \oplus (m \| 0^c)) \mid (A^{top}, A^{bot}) \in \texttt{Rooted}$
         $\wedge\, m \in \{0,1\}^r \wedge A^s \oplus (m \| 0^c) = x\}$;
3      NodesToAnswer $\leftarrow \emptyset$;
4      while NodesToVisit $\neq \emptyset$ do
           /* Pick any node $(X^{top}, X^{bot}) \in$ NodesToVisit */;
5          Let $(X^{top}, X^{bot}) \in$ NodesToVisit ;
6          NodesToAnswer $\leftarrow$ NodesToAnswer $\cup \{(X^{top}, X^{bot})\}$;
           /* Collect all rooted nodes which display an $u$-inner collision with
           $(X^{top}, X^{bot})$ */ ;
7          NodesToVisit $\leftarrow$ NodesToVisit $\cup \{(A^{top} \oplus (m \| 0^c), A^{bot} \oplus (m \| 0^c)) \mid m \in$
             $\{0,1\}^r \wedge (A^{top}, A^{bot}) \in \texttt{Rooted} \wedge \exists u \in \{top, bot\}, A^u \oplus (m \| 0^c) = X^u\}$;
8          NodesToVisit $\leftarrow$ NodesToVisit $\setminus$ NodesToAnswer;
9      end
10     return NodesToAnswer;
11 end
```

Then, the simulator aims at providing answers that satisfy

$$\begin{aligned}
\mathrm{outer}_r\left(\texttt{tabP}_{top}[X_1^{top}] \oplus 2\texttt{tabP}_{bot}[X_1^{bot}]\right) &= h_1\,,\\
\mathrm{outer}_r\left(\texttt{tabP}_{top}[X_2^{top}] \oplus 2\texttt{tabP}_{bot}[X_2^{bot}]\right) &= h_2\,,\\
&\vdots\\
\mathrm{outer}_r\left(\texttt{tabP}_{top}[X_m^{top}] \oplus 2\texttt{tabP}_{bot}[X_m^{bot}]\right) &= h_m\,.
\end{aligned}$$

The answers are $\mathcal{RO}$-consistent if and only if they satisfy the equations above. For every $i \in \{1, \ldots, m\}$, either X_i^{top} or X_i^{bot} appears in at least two equations, and therefore it is not always obvious whether the system of linear equations derived has a solution. In examples 2 and 3 below, we assume that $b > 2$, and the X_i^s denote values in $\{0,1\}^b$ where the X_i^{top}'s (resp., X_i^{bot}'s) are all distinct.

Example 2. Assume that

$$\textbf{NodesToAnswer} = \{(X_1^{top}, X_1^{bot}), (X_1^{top}, X_2^{bot}), (X_1^{top}, X_3^{bot})\}\,.$$

Then, after associating variable a_i^{top} (resp., a_i^{bot}) to $\mathrm{outer}_r\left(\texttt{tabP}_{top}[X_i^{top}]\right)$ (resp., $\mathrm{outer}_r\left(2\texttt{tabP}_{bot}[X_i^{bot}]\right)$), the system of consistency equations becomes

$$\begin{aligned}
a_1^{top} \oplus a_1^{bot} &= h_1\,,\\
a_1^{top} \oplus a_2^{bot} &= h_2\,,\\
a_1^{top} \oplus a_3^{bot} &= h_3\,.
\end{aligned}$$

Using a matrix notation, we obtain $\boldsymbol{M}\boldsymbol{a} = \boldsymbol{h}$, with

$$\boldsymbol{M} = \begin{pmatrix} 1\,1\,0\,0 \\ 1\,0\,1\,0 \\ 1\,0\,0\,1 \end{pmatrix}, \qquad \boldsymbol{a} = \begin{pmatrix} a_1^{top} \\ a_1^{bot} \\ a_2^{bot} \\ a_3^{bot} \end{pmatrix}, \qquad \boldsymbol{h} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix}.$$

The kernel of $\boldsymbol{M}$ is spanned by the vector $\langle(\mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1})\rangle$, so $\boldsymbol{M}$ has rank 3. Therefore for every $\boldsymbol{h} \in (\{0,1\}^r)^3$, the equation has a solution (Note that the values a_i^s do not need to be distinct).

Example 3. Assume that

$$\mathbf{NodesToAnswer} = \{(X_1^{top}, X_1^{bot}), (X_1^{top}, X_2^{bot}), (X_2^{top}, X_1^{bot}), (X_2^{top}, X_2^{bot})\}\,.$$

With the same notation as in Example 2, the system of consistency equations becomes

$$\begin{aligned} a_1^{top} \oplus a_1^{bot} &= h_1\,, \\ a_1^{top} \oplus a_2^{bot} &= h_2\,, \\ a_2^{top} \oplus a_1^{bot} &= h_3\,, \\ a_2^{top} \oplus a_2^{bot} &= h_4\,. \end{aligned}$$

Using a matrix notation, we obtain $\boldsymbol{Ma} = \boldsymbol{h}$, with

$$\boldsymbol{M} = \begin{pmatrix} 1\,0\,1\,0 \\ 1\,0\,0\,1 \\ 0\,1\,1\,0 \\ 0\,1\,0\,1 \end{pmatrix}\,, \qquad \boldsymbol{a} = \begin{pmatrix} a_1^{top} \\ a_2^{top} \\ a_1^{bot} \\ a_2^{bot} \end{pmatrix}\,, \qquad \boldsymbol{h} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{pmatrix}\,.$$

This square matrix has a kernel of dimension 1, it is thus non-invertible. In other words, for most vectors $\boldsymbol{h}$, the linear system is unsolvable.

In Section 4.3, we define a **BAD** event such that its absence always guarantees solvable linear systems with *uniform* solutions with *fresh* randomness. In this constrained setting, the simulator fixes the variables appearing in more than one equation to random values.

Example 4. Going back to Example 2, the simulator chooses $a_1^{top} \xleftarrow{\$} \{0,1\}^r$ and sets

$$\begin{aligned} a_1^{bot} &= h_1 \oplus a_1^{top}\,, \\ a_2^{bot} &= h_2 \oplus a_1^{top}\,, \\ a_3^{bot} &= h_3 \oplus a_1^{top}\,. \end{aligned}$$

Because $\boldsymbol{h}$ contains fresh random values and a_1^{top} was sampled uniformly at random, the elements a_k^s all follow the uniform distribution and are independent.

The associated procedure is called in line 15 of Algorithm 3. It returns two elements $\texttt{DictOuterParts}_{top}$ and $\texttt{DictOuterParts}_{bot}$ which, upon success, map the elements $X_i^s \in \{0,1\}^b$ to the associated r-bit constraint of $\texttt{tabP}_s[X_i^s]$. In more detail, for top outputs, this constraint is $\text{outer}_r\left(\texttt{tabP}_{top}[X_i^{top}]\right)$ while for bottom outputs this is $\text{outer}_r\left(2\texttt{tabP}_{bot}[X_i^{bot}]\right)$. When the simulator fails to solve the linear system, it returns $\perp$, therefore allowing the adversary to distinguish easily.

We henceforth assume that the simulator was able to solve the linear system.

3. Choice of the permutation outputs. In this step, the simulator samples permutation-consistent answers among the ones satisfying the constraint found in the previous step. The procedure is invoked in lines 19 and 20 of Algorithm 3 and presented in Algorithm 5, but is thoroughly explained in this paragraph. After the simulator fixed $y_r := \text{outer}_r\left(\texttt{tabP}_{top}[x]\right)$ in the previous step, it samples the answer from the set

$$\{y \in \{0,1\}^b \mid \text{outer}_r\left(y\right) = y_r\} \setminus \mathbf{Img}(\texttt{tabP}_{top}) \tag{2}$$

and similarly for bottom answers with $y_r := \text{outer}_r\left(2\texttt{tabP}_{bot}[x]\right)$ and $\text{outer}_r\left(2y\right) = y_r$. Note that at this stage, the simulator is *always* able to provide permutation-consistent answers (i.e., the set appearing in (2) is never empty), since the total number of queries is smaller than 2^c.

Algorithm 5: Simulator sub-routine to choose the permutation outputs.

```
/* DictOuterParts$_{top}$ (resp. DictOuterParts$_{bot}$) is a dictionary where its
   elements are of form ($x \in \{0,1\}^b \to z \in \{0,1\}^r$), meaning that the outer part
   of tabP$_{top}[x]$ (resp., 2tabP$_{bot}[x]$) is fixed to $z$ */
Function PermConsistentOutputTop (DictOuterParts$_{top}$)
    foreach $x^{top} \to y_r^{top} \in$ DictOuterParts$_{top}$ do
        $y^{top} \xleftarrow{\$} \{y \in \{0,1\}^b \mid \text{outer}_r(y) = y_r^{top}\} \setminus \mathbf{Img}(\texttt{tabP}_{top})$;
        tabP$_{top}[x^{top}] = y^{top}$;
    end
end
Function PermConsistentOutputBot (DictOuterParts$_{bot}$)
    foreach $x^{bot} \to y_r^{bot} \in$ DictOuterParts$_{bot}$ do
        $y^{bot} \xleftarrow{\$} \{y \in \{0,1\}^b \mid \text{outer}_r(2y) = y_r^{bot}\} \setminus \mathbf{Img}(\texttt{tabP}_{bot})$;
        tabP$_{bot}[x^{bot}] = y^{bot}$;
    end
end
```

Algorithm 6: Function ensuring that there is no partial edge.

```
Function EnsureNoPartialEdges ()
    /* Identify the nodes with an inner part in the query history */
    PartialTop $\leftarrow \{A^{top} \oplus (m\|0^c) \mid m \in \{0,1\}^r \wedge \exists A^{bot} \in \{0,1\}^b : (A^{top}, A^{bot}) \in$
      Rooted $\wedge$ tabP$_{top}[A^{top} \oplus (m\|0^c)] = \perp$
      $\wedge$ tabP$_{bot}[A^{bot} \oplus (m\|0^c)] \neq \perp\}$;
    PartialBot $\leftarrow \{A^{bot} \oplus (m\|0^c) \mid m \in \{0,1\}^r \wedge \exists A^{top} \in \{0,1\}^b : (A^{top}, A^{bot}) \in$
      Rooted $\wedge$ tabP$_{bot}[A^{bot} \oplus (m\|0^c)] = \perp$
      $\wedge$ tabP$_{top}[A^{top} \oplus (m\|0^c)] \neq \perp\}$;
    if PartialTop = PartialBot = $\emptyset$ then // Nothing to take care of
        return;
    end
    /* Node collection */
    NodesToAnswer $\leftarrow \{\}$;
    foreach $X^{top} \in$ PartialTop do
        NodesToAnswer = NodesToAnswer $\cup$ NodeCollection $(top, X^{top})$;
    end
    foreach $X^{bot} \in$ PartialBot do
        NodesToAnswer = NodesToAnswer $\cup$ NodeCollection $(bot, X^{bot})$;
    end
    /* Note that the linear system solving takes into account the already
    defined permutation outputs */
    DictOuterParts$_{top}$, DictOuterParts$_{bot}$ $\leftarrow$ LinSolve (NodesToAnswer);
    if $\exists s$, DictOuterParts$_s = \perp$ then // Linear system solving failed
        return;
    end
    /* Permutation consistency, graph update */
    PermConsistentOutputTop (DictOuterParts$_{top}$);
    PermConsistentOutputBot (DictOuterParts$_{bot}$);
    UpdateGraph ();
    EnsureNoPartialEdges () // Recursive call
end
```

4. Extension to several iterations. The simulator is not done yet. It is possible that the newly created edges point towards nodes with one of their inner parts appearing in the query history. In this case, the simulator needs to complete the partial edges with $\mathcal{RO}$-consistent answers. In the worse case where *both* parts appear in the query history (modulo xoring by a message block), then the simulator has lost. The underlying procedure, called `EnsureNoPartialEdges` () and presented in Algorithm 6, is similar to steps 1, 2, and 3 combined, unless that the consistency equation has some of its variables instantiated with particular values. One of the bad events defined in Section 4.3 ensures that the simulator does not require more than two iterations of `EnsureNoPartialEdges` () per query.

4.2.2 On Inverse Query On an inverse query, the simulator provides an answer uniformly at random among the permutation-consistent ones. If this query happens to hit a rooted node, then the simulator has to run the same procedure as the one described in step 4. It is noteworthy that the probability that an inverse query hits the tree is the same as the probability that a fresh node has one of its inner part appearing in the query history.

4.3 Bad Events and Discussion

Two main undesirable behaviors can occur when running the simulator algorithm. The first one happens when the linear system derived does not have a solution. In this case, regardless of the simulator's answer, the adversary can differentiate easily between W_I and W_{IM1}, since the difference between these two worlds lies in the $\mathcal{RO}$-consistency (see Section 4.4). The second undesirable behavior is when the simulator runs over a large number of iterations, i.e., step 4 is repeated many times. Besides the algorithm termination, this is a problem in the sense that every subsequent permutation output decided by the simulator is considered to be given *for free*, giving therefore a large factor in the upper bounds. In the following we define bad events which ensure that the simulator does not run into these problems.

4.3.1 Events on Edge Addition We first start by defining bad events that can be triggered at any iteration, whenever an edge is added to the graph.

i) **Double2Col**: there exist $(A^{top}, A^{bot}), (B^{top}, B^{bot}), (C^{top}, C^{bot}) \in$ `Rooted` such that $(A^{top}, A^{bot}) \neq (B^{top}, B^{bot})$, $(A^{top}, A^{bot}) \neq (C^{top}, C^{bot})$, and

$$\begin{aligned} \text{inner}_c\left(A^{top}\right) &= \text{inner}_c\left(B^{top}\right) \\ \text{and } \text{inner}_c\left(A^{bot}\right) &= \text{inner}_c\left(C^{bot}\right) . \end{aligned}$$

This event is illustrated in Fig. 2a;

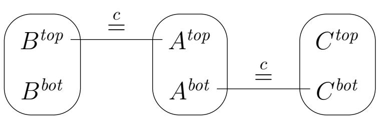
ii) **3Col**: there exist distinct $(A^{top}, A^{bot}), (B^{top}, B^{bot}), (C^{top}, C^{bot}) \in$ `Rooted` such that

$$\begin{aligned} \text{inner}_c\left(A^{top}\right) &= \text{inner}_c\left(B^{top}\right) = \text{inner}_c\left(C^{top}\right) \\ \text{or } \text{inner}_c\left(A^{bot}\right) &= \text{inner}_c\left(B^{bot}\right) = \text{inner}_c\left(C^{bot}\right) . \end{aligned}$$

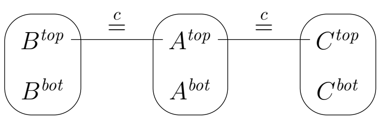
This event is illustrated in Fig. 2b;

iii) **BadNode**: there exist $i \in \{1, \dots, q\}, m, m_1, m_2 \in \{0,1\}^r$, and $(A^{top}, A^{bot}), (B^{top}, B^{bot}) \in$ `Rooted`, such that (i) before query i, $(B^{top}, B^{bot}) \notin$ `Rooted`, (ii) during query i, the edge $(A^{top}, A^{bot}) \xrightarrow{m} (B^{top}, B^{bot})$ is produced, (iii) *both* $B^{top} \oplus (m_1 \| 0^c)$ and $B^{bot} \oplus (m_2 \| 0^c)$ were already in the top (resp., bottom) query history *before* the edge was produced. This event is illustrated in Fig. 2d;

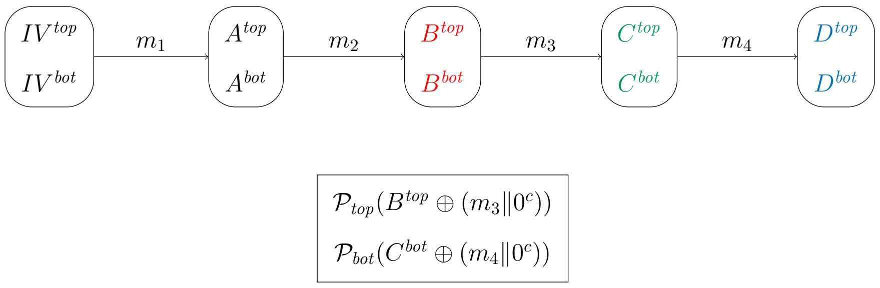
iv) **BadNode_Strong**: there exist $i \in \{1, \dots, q\}, m, m' \in \{0,1\}^r$, and $(A^{top}, A^{bot}), (B^{top}, B^{bot}) \in$ `Rooted` such that (i) before query i, $(B^{top}, B^{bot}) \notin$ `Rooted`, (ii) during query i, the edge $(A^{top}, A^{bot}) \xrightarrow{m} (B^{top}, B^{bot})$ is produced, (iii) *both* $B^{top} \oplus (m' \| 0^c)$ and $B^{bot} \oplus (m' \| 0^c)$ were already in the top (resp., bottom) query history *before* the edge was produced. In short, **BadNode_Strong** is set when a new edge is added from (B^{top}, B^{bot}) to the graph without the simulator being able to guarantee $\mathcal{RO}$-consistency.



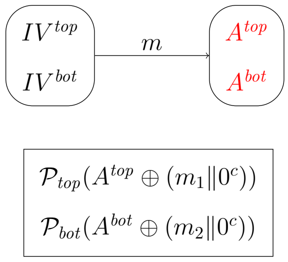
(a) **Double2Col**.



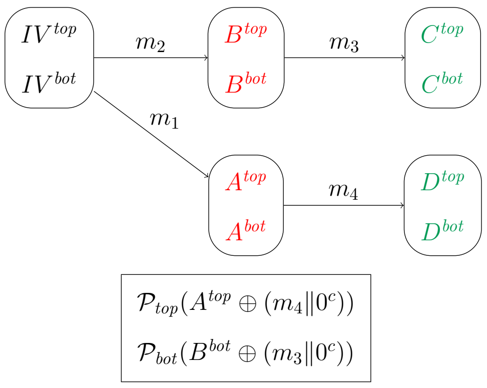
(b) **3Col**.



(c) **ThreeRounds**.


IV^{top}
IV^{bot}
m
A^{top}
A^{bot}
$\mathcal{P}_{top}(A^{top} \oplus (m_1\|0^c))$
$\mathcal{P}_{bot}(A^{bot} \oplus (m_2\|0^c))$

(d) **BadNode**.



(e) **AUX**^{3}.

Fig. 2: Illustration of the main **BAD** events. In Figures 2a and 2b, the drawn nodes are rooted. In Figures 2c to 2e, one color represents one iteration of the simulator and the rectangles contain particular permutation outputs already known by the adversary before the query.

4.3.2 Second-Iteration Event

We define a bad event **ThreeRounds** specific to the second iteration of the algorithm, i.e., the second (resp., third) call of `EnsureNoPartialEdges` () in step 4 for a forward (resp. inverse) query. At a high level view, this event means that one single query triggers the addition of three consecutive edges $(A^{top}, A^{bot}) \xrightarrow{m} (B^{top}, B^{bot}) \xrightarrow{m'} (C^{top}, C^{bot}) \xrightarrow{m''} (D^{top}, D^{bot})$, where $(A^{top}, A^{bot}) \in \texttt{Rooted}$. For a forward query `FwdPQuery` (s, X), it means that there exist $i \in \{1, \ldots, q\}$, $(A^{top}, A^{bot}) \in \texttt{Rooted}$, $(B^{top}, B^{bot}), (C^{top}, C^{bot}) \in \{0,1\}^{2b}$, $m, m', m'' \in \{0,1\}^r$ such that at query i, the following sequence of events happens:

1. $X = A^s \oplus (m\|0^c)$, so that the first iteration of `NodeCollection` () gives $(A^{top} \oplus (m\|0^c), A^{bot} \oplus (m\|0^c)) \in$ **NodesToAnswer**;
2. The edge $(A^{top}, A^{bot}) \xrightarrow{m} (B^{top}, B^{bot})$ is added, making (B^{top}, B^{bot}) a rooted node;
3. $B^{top} \oplus (m'\|0^c)$ or $B^{bot} \oplus (m'\|0^c)$ was already in the top (resp., bottom) query history, so that another iteration of `NodeCollection` () is necessary;

4. The edge $(B^{top}, B^{bot}) \xrightarrow{m'} (C^{top}, C^{bot})$ is added, making (C^{top}, C^{bot}) a rooted node;
5. $C^{top} \oplus (m'' \| 0^c)$ or $C^{bot} \oplus (m'' \| 0^c)$ appears in the top (resp., bottom) query history, so that another iteration of `NodeCollection` () is needed.

For an inverse query `InvPQuery` (s, Y), additionally to the sequence of events above, the response of the inverse query must beforehand hit one of the inner parts of (A^{top}, A^{bot}). **ThreeRounds** is illustrated in Fig. 2c.

4.3.3 Auxiliary Events Finally, we introduce the following events, which allow to simultaneously eliminate rare but undesirable cases, simplify the proof, and reduce the constant factors in the bounds.

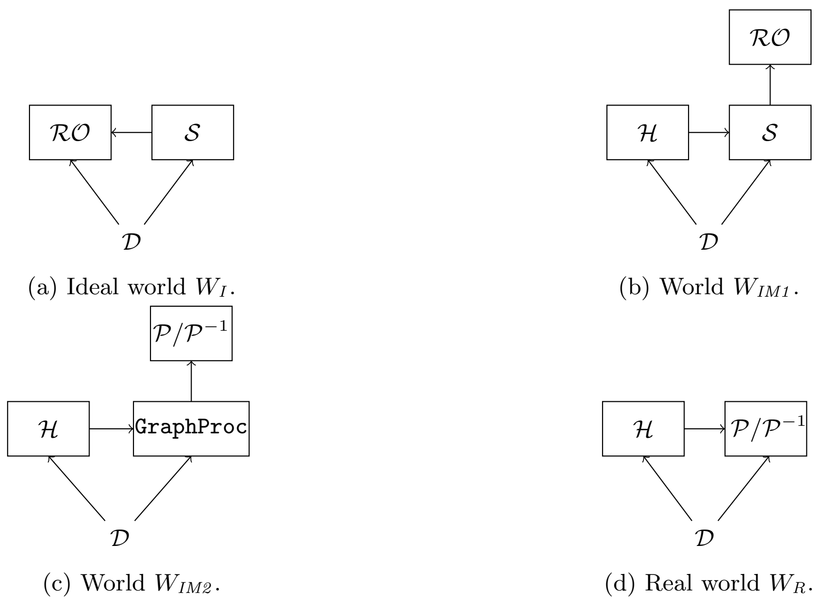
i) $\mathbf{AUX}^1$: There exist $i \in \{1, \ldots, q\}$, and distinct (A^{top}, A^{bot}), $(B^{top}, B^{bot}) \in \{0,1\}^{2b}$ such that the query i adds (A^{top}, A^{bot}) and (B^{top}, B^{bot}) to the set `Rooted` and such that $A^s \overset{c}{=} B^s$ for some $s \in \{top, bot\}$;
ii) $\mathbf{AUX}^2$: There exist $s \in \{top, bot\}$, $i \in \{1, \ldots, q\}$, $X \in \{0,1\}^b$, and distinct (A^{top}, A^{bot}), $(B^{top}, B^{bot}) \in$ `Rooted` such that query i is of the form `InvPQuery` (s, X) and the answer Y is such that $Y \overset{c}{=} A^s$ and (A^{top}, A^{bot}) displays an inner collision with (B^{top}, B^{bot}) on their s part for $s \in \{top, bot\}$;
iii) $\mathbf{AUX}^3$: There exists $i \in \{1, \ldots, q\}$ such that query i requires a second iteration of the simulator during which the set **NodesToAnswer** in Algorithm 6 has a size greater than 1. This event is illustrated in Fig. 2e.

4.3.4 Interpretation Intuitively, **Double2Col** and **3Col** ensure that the set **NodesToAnswer** is of size at most 2 and that the linear system always has a solution for first iteration nodes. For the subsequent iterations, we additionally need the event **BadNode_Strong**. However, **BadNode** is implied by **BadNode_Strong**, and the former event is useful for the other parts of the proof, thus we will only make use of **BadNode**. Note that **Double2Col** and **3Col** are suboptimal for the linear system solvability, e.g., in Example 2, the linear system is solvable while **3Col** is set. The real bad event here would be to have a cycle in the graph defined in step 1, Section 4.2. Nevertheless, **Double2Col** and **3Col** are very useful for the probability computation of **ThreeRounds**, which allows to upper bound the size of the tree. Note that when assuming $\neg\mathbf{AUX}^3_i$, $\mathbf{ThreeRounds}_i$ is the only scenario where the simulator requires more than two iterations of `NodeCollection` () during query i. Finally, $\mathbf{AUX}^1$, $\mathbf{AUX}^2$, and $\mathbf{AUX}^3$ are bad events that alleviate the proof complexity and eliminate bad cases. In the following, let

$$\mathbf{BAD} := \mathbf{Double2Col} \vee \mathbf{3Col} \vee \mathbf{BadNode} \vee \mathbf{ThreeRounds} \vee \bigvee_j \mathbf{AUX}^j .$$

4.4 World Decomposition

Remember that the ideal world W_I consists of $(\mathcal{RO}, \mathcal{S}^{fwd}, \mathcal{S}^{inv})$, and the real world W_R consists of $(\mathcal{H}^{\mathcal{P}}, \mathcal{P}, \mathcal{P}^{-1})$. Our indifferentiability proof uses two intermediate worlds as shown in Fig. 3. The first one, called W_{IM1}, gives access to $(\mathcal{H}^{\mathcal{S}}, \mathcal{S}^{fwd}, \mathcal{S}^{inv})$, where $\mathcal{S}$ relies on a random oracle. It is a natural intermediate step between the real and the ideal world, since it allows to separate the $\mathcal{RO}$-consistency (c.f., Definition 1) from the simulator's quality of randomness. Note that since the **BAD** event is defined on the simulator $\mathcal{S}$, the former therefore also applies in W_{IM1}. However, while the distance from W_I to W_{IM1} is now easier to analyze, this is not the case for the distance between W_{IM1} and W_R. Indeed, in W_R, the answers are returned using lazy sampling, so that the notion of "iterations" does not exist in this world, thus preventing the usage of the bad event **ThreeRounds**. For that reason, we introduce another intermediate world W_{IM2}, which implements the real world with a supplementary layer mimicking the simulator's node collection phase and step 4 of Section 4.2. This



(a) Ideal world W_I. (b) World W_{IM1}. (c) World W_{IM2}. (d) Real world W_R.

Fig. 3: Definition of the different worlds. $\mathcal{P}$ returns responses with lazy sampling.

layer is called `GraphProc` and is described as follows: on a forward query or inverse query, an answer is drawn using lazy sampling. Then, the procedure `EnsureNoPartialEdges` () is run, with `PermConsistentOutputTop` () and `PermConsistentOutputBot` () replaced by a permutation-consistent lazy sampling.

For $x \in \{I, IM1, IM2, R\}$, let $\mathcal{D}^{W_x}$ denote the fact that distinguisher $\mathcal{D}$ is in world W_x. Recall that our goal is to bound $\mathbf{Adv}^{\text{indif}}_{\mathcal{H}^P,\mathcal{S}}(\mathcal{D})$. By the triangle inequality:

$$\begin{aligned}
\mathbf{Adv}^{\text{indif}}_{\mathcal{H}^P,\mathcal{S}}(\mathcal{D}) &= \left|\mathbf{Pr}\left(\mathcal{D}^{W_I}=1\right) - \mathbf{Pr}\left(\mathcal{D}^{W_R}=1\right)\right| \\
&\le \left|\mathbf{Pr}\left(\mathcal{D}^{W_I}=1\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}=1\right)\right| && \text{(3a)} \\
&+ \left|\mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}=1\right) - \mathbf{Pr}\left(\mathcal{D}^{W_R}=1\right)\right| . && \text{(3b)}
\end{aligned}$$

For the distance in (3a):

$$\begin{aligned}
\text{(3a)} = \Big| &\mathbf{Pr}\left(\mathcal{D}^{W_I}=1 \wedge \mathbf{BAD}\right) + \mathbf{Pr}\left(\mathcal{D}^{W_I}=1 \wedge \neg\mathbf{BAD}\right) \\
&- \mathbf{Pr}\left(\mathcal{D}^{W_{IM1}}=1 \wedge \neg\mathbf{BAD}\right) + \mathbf{Pr}\left(\mathcal{D}^{W_{IM1}}=1 \wedge \neg\mathbf{BAD}\right) \\
&- \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}=1 \wedge \mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}=1 \wedge \neg\mathbf{BAD}\right) \Big| \\
\le &\left|\mathbf{Pr}\left(\mathcal{D}^{W_I}=1 \wedge \mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}=1 \wedge \mathbf{BAD}\right)\right| \\
+ &\left|\mathbf{Pr}\left(\mathcal{D}^{W_I}=1 \wedge \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM1}}=1 \wedge \neg\mathbf{BAD}\right)\right| \\
+ &\left|\mathbf{Pr}\left(\mathcal{D}^{W_{IM1}}=1 \wedge \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}=1 \wedge \neg\mathbf{BAD}\right)\right| \\
\le \Big| &\mathbf{Pr}\left(\mathcal{D}^{W_I}\text{ sets }\mathbf{BAD}\right)\mathbf{Pr}\left(\mathcal{D}^{W_I}=1 \mid \mathbf{BAD}\right) - \\
&\qquad \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}\text{ sets }\mathbf{BAD}\right)\mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}=1 \mid \mathbf{BAD}\right)\Big| && \text{(4a)} \\
+ &\left|\mathbf{Pr}\left(\mathcal{D}^{W_I}=1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM1}}=1 \mid \neg\mathbf{BAD}\right)\right| && \text{(4b)} \\
+ &\left|\mathbf{Pr}\left(\mathcal{D}^{W_{IM1}}=1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}}=1 \mid \neg\mathbf{BAD}\right)\right| . && \text{(4c)}
\end{aligned}$$

The distance in (4a) can eventually be bounded as follows:

$$(4a) \leq \max\left\{ \mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{BAD}\right), \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} \text{ sets } \mathbf{BAD}\right) \right\}. \tag{5}$$

In conclusion, we find that $\mathbf{Adv}^{\text{indif}}_{\mathcal{H}^P,\mathcal{S}}(\mathcal{D})$ of (3) is bounded by the sum of the terms in (5), (4b), (4c), and (3b):

$$\mathbf{Adv}^{\text{indif}}_{\mathcal{H}^P,\mathcal{S}}(\mathcal{D}) \leq \max\left\{ \mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{BAD}\right), \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} \text{ sets } \mathbf{BAD}\right) \right\} \tag{6a}$$

$$+ \left| \mathbf{Pr}\left(\mathcal{D}^{W_I} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM1}} = 1 \mid \neg\mathbf{BAD}\right) \right| \tag{6b}$$

$$+ \left| \mathbf{Pr}\left(\mathcal{D}^{W_{IM1}} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} = 1 \mid \neg\mathbf{BAD}\right) \right| \tag{6c}$$

$$+ \left| \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{W_R} = 1\right) \right|. \tag{6d}$$

We will discuss these two probabilities and three distances below. In the following, let $q := \sigma + q_{\mathcal{P}}$.

Bad events. We upper bound the **BAD** event probabilities in Lemma 2 (Section 5.1) and obtain the upper bound

$$(6a) \leq \frac{249q^3}{(2^c - 3q)^2} + \frac{21q}{2^c - 3q}.$$

Ideal world W_I versus W_{IM1} as long as no **BAD**. Here, the adversary has access to $(\mathcal{RO}, \mathcal{S}^{fwd}, \mathcal{S}^{inv})$ or $(\mathcal{H}^{\mathcal{S}}, \mathcal{S}^{fwd}, \mathcal{S}^{inv})$, where in W_{IM1}, $\mathcal{S}^{fwd}$ and $\mathcal{S}^{inv}$ are built upon a random oracle. Therefore, as long as the simulator is $\mathcal{RO}$-consistent with respect to the double sponge, the two worlds are indistinguishable. The $\mathcal{RO}$-consistency is guaranteed by the success of the linear system solving phase of the simulator. More formally, in Lemma 3 (Section 5.2), we prove that as long as $\neg$**BAD** holds, this phase never fails. The proof relies on a simple enumeration of all possible cases regarding the structure of the set **NodesToAnswer**. Therefore,

$$(6b) = 0.$$

W_{IM1} versus W_{IM2} as long as no **BAD**. Here, the adversary has access to $(\mathcal{H}^{\mathcal{S}}, \mathcal{S}^{fwd}, \mathcal{S}^{inv})$ or $(\mathcal{H}^{\mathcal{P}}, \mathcal{P}, \mathcal{P}^{-1})$. Now, since the construction component is the same in both worlds, the distinguisher can convert the construction queries into primitive queries. The game thus reduces to a distinguishing game between $(\mathcal{S}^{fwd}, \mathcal{S}^{inv})$ and $(\mathcal{P}, \mathcal{P}^{-1})$. The full proof is in Lemma 4 (Section 5.3), and it gives an upper bound

$$(6c) \leq \frac{3q^{\frac{3}{2}}}{2^c - 3q}.$$

W_{IM2} versus real world W_R. Similarly to the real world, world W_{IM2} implements a double sponge using a permutation, the only difference lies in the timing at which the permutation samplings are performed. A permutation with answers being lazily sampled is equivalent to a permutation drawn at the beginning of the game. These two worlds are thus equivalent, and hence

$$(6d) = 0.$$

From (6), we eventually obtain

$$\begin{aligned}\mathbf{Adv}^{\mathrm{indif}}_{\mathcal{H}^{\mathcal{P}},\mathcal{S}}(q_{\mathcal{P}},\sigma) &\leq \frac{249q^3}{(2^c-3q)^2}+\frac{21q}{2^c-3q}+\frac{3q^{\frac{3}{2}}}{2^c-3q}\\ &\leq \frac{19q^{\frac{3}{2}}}{2^c-3q}+\frac{21q}{2^c-3q}\\ &\leq \frac{40q^{\frac{3}{2}}}{2^c-3q}\,,\end{aligned}$$

which concludes Theorem 1.

4.5 Extension to Single Permutation

One natural question arises on whether it is possible to extend the double sponge construction by replacing $\mathcal{P}_{top}$ and $\mathcal{P}_{bot}$ by the same permutation, but with different initialization vectors IV^{top} and IV^{bot}. From a proof perspective, such an extension would significantly complicate the analysis, and particularly increase drastically the number of cases to be considered. On the other hand, using a single permutation is desirable in practice. Thus, as a trade-off, we propose the following modification to the double sponge construction: we replace the permutations $\mathcal{P}_{top}$ and $\mathcal{P}_{bot}$ by respectively $\mathcal{P}(0\|\cdot)[1:b]$ and $\mathcal{P}(1\|\cdot)[1:b]$. In that setting, the impact on the simulator definition, bad event analysis, and statistical distance computation is relatively mild.

The presence of domain separator bits allows the simulator to determine whether it needs to consider the top or bottom part of the nodes during a forward query. As a result, the procedure `NodeCollection` () does not change, apart from a parsing adjustment. The procedure `LinSolve` () does not need any changes. The procedures `PermConsistentOutputTop` () and `PermConsistentOutputBot` () can be merged into a single procedure. Next, if (X^{top}, X^{bot}) is a newly produced node (after absorption of a message block), then the updated version of `EnsureNoPartialEdges` () runs a new iteration only when either $0\|X^{top}$ or $1\|X^{bot}$ appears in the query history. Note that having $1\|X^{top}$ or $0\|X^{bot}$ in the query history does not require one other iteration, since if X^{top} (resp., X^{bot}) appears later as a bottom (resp., top) node, it will be generated from a fresh source of randomness. Finally, the algorithm run by the simulator on an inverse query does not change.

Regarding the upper bounding of distances, only the bad event analysis (equation (6a)) and the quality of randomness of the simulator (equation (6c)) are affected. In the bad event analysis, the nodes are now linear combinations of truncated permutation outputs instead of permutation outputs, but this modification does not affect the upper bounds derived in the analysis.[3] On the other hand, we need an additional bad event which states that one query to the simulator results in the addition of two rooted nodes (A^{top}, A^{bot}) and (B^{top}, B^{bot}) such that $A^{top} \overset{c}{=} B^{bot}$. This is a rare event, and the auxiliary bad events ensure that the simulator generates no more than three nodes in a single query. Therefore, this event occurs with a probability of form $\mathcal{O}\left(\frac{q}{2^c}\right)$. Thus, modulo the loss of one bit and small constant factors, the bad event upper bounding does not change significantly. Now, regarding the quality of randomness of the simulator, the permutation's randomness is exhausted twice as fast, resulting in further constant factor losses.

5 Probability Computation

In this section, we establish upper bounds for the three terms from equations ($6a$), ($6b$), and ($6c$) in Sections 5.1, 5.2, and 5.3 respectively.

[3] To be precise, the bounding in equation (9) still holds.

5.1 BAD Event Probability Analysis

Before stating an upper bound on the **BAD** probabilities in Lemma 2, we define the trimmed tree in Definition 2, and state a result on the size of the latter. This result ensures the algorithm termination and allows to upper bound the number of queries given for free by the simulator.

Definition 2. *The trimmed tree is a subset of* `Rooted` *defined in an inductive way:*

- *Initialization:* `Trimmed` $\leftarrow \{(IV^{top}, IV^{bot})\}$*;*
- *On query* i*, whenever* $(A^{top}, A^{bot}) \xrightarrow{m} (B^{top}, B^{bot})$ *is added to the graph with* $(A^{top}, A^{bot}) \in$ `Trimmed`*; if* (B^{top}, B^{bot}) *does not trigger* **BAD***, then* `Trimmed` $\leftarrow$ `Trimmed`$\cup\{(B^{top}, B^{bot})\}$.

In other words, the trimmed tree is the set of rooted nodes deprived of the ones (and their descendants) that would trigger **BAD** *if added.*

Lemma 1. *For any* $i \in \{0, \ldots, q\}$*, the size of the trimmed tree after* i *queries is upper bounded by* $3i + 1$*.*

Proof. First observe that the nodes returned by `NodeCollection` () must display a collision on their top or bottom part with another node in the same set. Therefore, having more than two nodes returned implies that there is either a 3-collision or a double collision among the rooted nodes, which correspond to respectively **3Col** and **Double2Col**. Therefore, as long as no **BAD** happens, `NodeCollection` () returns at most two nodes each time.

Now, the trimmed tree is initialized with a single node (IV^{top}, IV^{bot}), thus has size of 1. Then, at query i, if the procedure `NodeCollection` () is run using `Trimmed` instead of `Rooted`, we know from the above observation that $|\mathbf{NodesToAnswer}|$ is at most 2. In addition, because of **ThreeRounds**, the nodes produced after more than two iterations are not added to the trimmed tree. Finally, $\mathbf{AUX}^3$ guarantees that at most one node has two consecutive edges added. We conclude that in total, at most 3 nodes are added to the trimmed tree per query. □

Lemma 2. *For any distinguisher* $\mathcal{D}$ *making* $q_{\mathcal{P}}$ *primitive queries and construction queries which would correspond to a total of* σ *primitive queries in world* W_R*, if* $30 < 2^c$*, one has*

$$\mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{BAD}\right) \leq \frac{249 q_{\mathcal{P}}^3}{(2^c - 3q_{\mathcal{P}})^2} + \frac{21 q_{\mathcal{P}}}{2^c - 3q_{\mathcal{P}}},$$
$$\mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} \text{ sets } \mathbf{BAD}\right) \leq \frac{249 q^3}{(2^c - 3q)^2} + \frac{21 q}{2^c - 3q},$$

where $q := \sigma + q_{\mathcal{P}}$*.*

Proof. For an event $\mathbf{E}$, we denote by $\mathbf{Pr}_I(\mathbf{E})$ (resp., $\mathbf{Pr}_{IM2}(\mathbf{E})$) the probability that $\mathbf{E}$ occurs in world W_I (resp., W_{IM2}). Let q be the total number of primitive queries made, so that $q = q_{\mathcal{P}}$ in world W_I, and $q \leq \sigma + q_{\mathcal{P}}$ in world W_{IM2}. Let

$$\begin{aligned} S_{\mathbf{2Col}} &:= \Big\{(A^{top}, A^{bot}) \in \texttt{Trimmed} \mid \exists (B^{top}, B^{bot}) \in \texttt{Trimmed} \setminus \{(A^{top}, A^{bot})\}, \\ &\qquad s \in \{top, bot\} \text{ s.t., } A^s \overset{c}{=} B^s \Big\}, \\ N_{\mathbf{2Col}} &:= |S_{\mathbf{2Col}}| \,. \end{aligned}$$

In other words, $N_{\mathbf{2Col}}$ counts the number of nodes displaying an inner collision on their top or bottom part in the trimmed tree. Note that, in particular, $N_{\mathbf{2Col}}$ cannot be larger than $3q + 1$.

Remark 1. As long as **BAD** is not set, $S_{\mathbf{2Col}}$ can be split as the disjoint union of $S_{\mathbf{2Col}}(top)$ and $S_{\mathbf{2Col}}(bot)$ defined as follows for $s \in \{top, bot\}$.

$$S_{\mathbf{2Col}}(s) = \Big\{(A^{top}, A^{bot}) \in \texttt{Trimmed} \mid \\ \exists (B^{top}, B^{bot}) \in \texttt{Trimmed} \setminus \{(A^{top}, A^{bot})\} \text{ s.t., } A^s \overset{c}{=} B^s \Big\} .$$

As long as **BAD** does not occur, the size of $S_{\mathbf{2Col}}$ is the sum of the two set sizes.

Now, by basic probability theory, for $x \in \{I, IM2\}$,

$$\mathbf{Pr}_x(\mathbf{BAD}) \leq \sum_{m=0}^{3q+1} \mathbf{Pr}_x(\mathbf{BAD} \mid N_{\mathbf{2Col}} = m) \cdot \mathbf{Pr}_x(N_{\mathbf{2Col}} = m) \tag{7}$$

$$\leq \sum_{m=0}^{3q+1} \sum_{i=1}^{q} \mathbf{Pr}_x(\mathbf{BAD}_i \mid N_{\mathbf{2Col}} = m \wedge \neg\mathbf{BAD}_{i-1}) \cdot \mathbf{Pr}_x(N_{\mathbf{2Col}} = m) , \tag{8}$$

where we remind that $\mathbf{BAD}_i$ denotes that **BAD** is set after i queries.

We start by the conditioned probabilities $\mathbf{Pr}_x(\mathbf{BAD}_i \mid N_{\mathbf{2Col}} = m \wedge \neg\mathbf{BAD}_{i-1})$ for $x \in \{I, IM2\}$ and any $i \in \{1, \ldots, q\}$ and $m \in \{0, \ldots, 3q+1\}$. An important observation is that, because of $\neg\mathbf{BAD}_{i-1}$, the trimmed tree recoverable from the first $i-1$ queries and the actual tree coincide. Therefore, from $N_{\mathbf{2Col}} = m$ and $\neg\mathbf{3Col}_{i-1} \wedge \neg\mathbf{Double2Col}_{i-1}$, we know that the tree has exactly $m/2$ colliding pairs on their inner top or bottom part.

Remark 2. In world W_{IM2}, the responses are drawn uniformly at random from a set of size at least $2^b - 3i$, while in the ideal world W_I, the inner parts of the responses are sampled uniformly at random in a set of size at least $2^c - 3i$. Therefore, for any forward query with answer Y, $y \in \{0,1\}^b$, and $x \in \{I, IM2\}$,

$$\mathbf{Pr}_x\left(Y \overset{c}{=} y\right) \leq \frac{1}{2^c - 3q} . \tag{9}$$

Looking ahead, the probability computation will give the same upper bound in both worlds.

Since **BAD** groups several sub-events, we split accordingly the probability computation, and when it is meaningful each sub-event is studied as many times as the number of iterations of the simulator that we consider. For instance, remember that $\mathbf{Double2Col}_i$ is an event that can occur when an edge is added. It is split into $(\mathbf{Double2Col}_i[k])_{k \in \mathbb{N}\setminus\{0\}}$, where $\mathbf{Double2Col}_i[k]$ means that the k^{th} iteration of the simulator during the query i sets **Double2Col**. Note that the order in which we consider the bad events matters. In more detail, if we compute the probability of event **E**, then **F**, and then **G**, in the last computation $\neg\mathbf{E} \wedge \neg\mathbf{F}$ is implicitly assumed.

The remainder of the proof is organized as follows. Sections 5.1.1 to 5.1.3 are dedicated to forward queries events, where each section concerns one iteration of the simulator. Section 5.1.4 looks at inverse queries events. Finally, Section 5.1.5 gathers the found upper bounds and plugs them into (8).

5.1.1 First Iteration Events, Forward Query The first iteration on a forward query is the easiest case, since the newly created edges provide the maximal possible randomness. Because of $\neg\mathbf{BAD}_{i-1}$, and from the remark at the beginning of the proof of Lemma 1, the node collection phase returns at most two nodes (after absorption). In the following, we focus on the hardest case where $|\mathbf{NodesToAnswer}| = 2$, so that $\mathbf{NodesToAnswer} := \{(X_1^{top}, X_1^{bot}), (X_2^{top}, X_2^{bot})\}$, where $X_1^s = X_2^s$ for $s \in \{top, bot\}$. Thus, three permutation outputs need to be fixed. For $k \in \{1,2\}$ and $s \in \{top, bot\}$, let Y_k^s be the random variable equal to $\texttt{tabP}_s[X_k^s]$. After the first iteration, the graph has two more rooted nodes that we denote by $(A^{top}, A^{bot}), (B^{top}, B^{bot})$. Now we can look at the probability that (A^{top}, A^{bot}) and/or (B^{top}, B^{bot}) triggers **BAD** using the condition $N_{\mathbf{2Col}} = m$.

1. **AUX**^{1}. This event implies that

$$\begin{aligned} Y_1^{top} + 2Y_1^{bot} &\overset{c}{=} Y_2^{top} + 2Y_2^{bot} \\ \text{or } \ 2Y_1^{top} + Y_1^{bot} &\overset{c}{=} 2Y_2^{top} + Y_2^{bot}, \end{aligned}$$

where we know that $Y_1^{top} = Y_2^{top}$ or $Y_1^{bot} = Y_2^{bot}$, but the other elements are distinct. By Remark 2, in both worlds, this event happens with probability at most $\dfrac{2}{2^c - 3q}$.

Now, $\neg\mathbf{AUX}_i^1$ allows to study separately the probability that the node (A^{top}, A^{bot}) (resp., (B^{top}, B^{bot})) triggers a first iteration event. Since the probabilities are the same, in the following we focus only on (A^{top}, A^{bot}).

2. **AUX**^{2}. The event **AUX**^{2} only concerns inverse queries, thus it does not apply here.

3. **AUX**^{3}. The event **AUX**^{3} only concerns the second iteration, thus it does not apply here.

4. **3Col**. This event means that there exist distinct $(N_1^{top}, N_1^{bot}), (N_2^{top}, N_2^{bot}) \in \texttt{Rooted}$ such that

$$\begin{aligned} A^{top} &\overset{c}{=} N_1^{top} \overset{c}{=} N_2^{top} \\ \text{or } \ A^{bot} &\overset{c}{=} N_1^{bot} \overset{c}{=} N_2^{bot}. \end{aligned}$$

Because of the condition $N_{\mathbf{2Col}} = m$, there are $m/2$ tuples $\{(N_1^{top}, N_1^{bot}), (N_2^{top}, N_2^{bot})\}$. From Remark 1, we know that the latter are disjointly split into $S_{\mathbf{2Col}}(bot)$ and $S_{\mathbf{2Col}}(top)$. Therefore the probability is upper bounded by $\dfrac{m}{2(2^c - 3q)}$.

5. **Double2Col**. There are two possibilities:

– The inserted node *displays* a double collision. In other words, there exist (N_1^{top}, N_1^{bot}), $(N_2^{top}, N_2^{bot}) \in \texttt{Rooted}$ (not necessarily distinct) such that

$$\begin{aligned} A^{top} &\overset{c}{=} N_1^{top} \\ \text{and } A^{bot} &\overset{c}{=} N_2^{bot}. \end{aligned}$$

Let $(M_1^{top}, M_1^{bot}) := MIX^{-1}(N_1^{top}, N_1^{bot})$, and $(M_2^{top}, M_2^{bot}) := MIX^{-1}(N_2^{top}, N_2^{bot})$. Then, this case reduces to

$$\begin{aligned} Y_1^{top} &\overset{c}{=} M_1^{top} \\ \text{and } Y_1^{bot} &\overset{c}{=} M_2^{bot}. \end{aligned} \tag{10}$$

Now, because of $\neg\mathbf{BAD}_{i-1}$, there are at most $(3(i-1)+1)^2 \leq (3i)^2$ tuples $(N_1^{top}, N_1^{bot}), (N_2^{top}, N_2^{bot}) \in \texttt{Rooted}$, and for each tuple, (10) is satisfied with probability at most $\left(\dfrac{1}{2^c - 3q}\right)^2$. Therefore, this event happens with probability at most $\left(\dfrac{3i}{2^c - 3q}\right)^2$.

– The inserted node is *part of* a double collision: in other words there exist $(N_1^{top}, N_1^{bot}), (N_2^{top}, N_2^{bot}) \in \texttt{Rooted}$ such that

$$\begin{cases} N_1^{top} \overset{c}{=} A^{top}, \\ N_1^{bot} \overset{c}{=} N_2^{bot}, \end{cases} \quad \text{or} \quad \begin{cases} N_1^{top} \overset{c}{=} N_2^{top}, \\ N_1^{bot} \overset{c}{=} A^{bot}. \end{cases}$$

Similarly to **3Col**, this gives a probability of $\dfrac{m}{2^c - 3q}$.

6. **BadNode***.* This event means that there exist $x^{top} \in \mathbf{Dom}(\texttt{tabP}_{top})$ and $x^{bot} \in \mathbf{Dom}(\texttt{tabP}_{bot})$ such that

$$A^{top} \overset{c}{=} x^{top}$$
$$\text{and } A^{bot} \overset{c}{=} x^{bot} .$$

For every fixed tuple (x^{top}, x^{bot}), this happens with probability at most $\left(\frac{1}{2^c - 3q}\right)^2$, and there are at most $9i^2$ such tuples. Therefore we obtain the upper bound $\left(\frac{3i}{2^c - 3q}\right)^2$.

Now we are done with the first iteration events. We can then compute the same probabilities for second-iteration events (if a second iteration is required) and compute the probability that a third iteration is needed.

5.1.2 Second-Iteration Events, Forward Query Thanks to $\neg$**BadNode** for the first iteration nodes, neither (A^{top}, A^{bot}) nor (B^{top}, B^{bot}) have the inner part of *both* their top and bottom part in the query history. Nevertheless, it is possible that only their top or bottom part displays an inner collision with the query history, and this is not a bad event. Since the size of the top (resp., bottom) query history is upper bounded by $3i$, and four different cases can happen, i.e., with node (A^{top}, A^{bot}) or (B^{top}, B^{bot}), on top or bottom part, this event happens with probability at most $\frac{12i}{2^c - 3q}$. In the following, we first get rid of the case where the simulator or `GraphProc` has to take care of more than one node.

1. $\mathbf{AUX}^3$*.* This event can be achieved in two different ways:

– *Both* (A^{top}, A^{bot}) and (B^{top}, B^{bot}) have their upper or lower part appearing in the query history. The probability analysis is similar to the one made at the first paragraph of Section 5.1.2, and we obtain an upper bound of $\left(\frac{6i}{2^c - 3q}\right)^2$;

– (A^{top}, A^{bot}) (or (B^{top}, B^{bot})) has its upper or lower part appearing in the query history *and* displays an inner collision with another rooted rode. For $s \in \{top, bot\}$, let

$$\xi^s = \left\{ j \in \{1, \ldots, q\} \mid (A_j^{top}, A_j^{bot}) \in S_{\mathbf{2Col}}(s) \vee (B_j^{top}, B_j^{bot}) \in S_{\mathbf{2Col}}(s) \right\} , \qquad (11)$$

where we abuse notation for (A_j^{top}, A_j^{bot}) and (B_j^{top}, B_j^{bot}) to refer to the nodes produced during the first iteration of query j. Then, as long as **BAD** does not occur, ξ^{top} and ξ^{bot} are disjoint and the size of $\xi^{top} \cup \xi^{bot}$ is at most m. Now, this case of $\mathbf{AUX}^3$ happens with a non-zero probability if and only if $i \in \xi^s$ for $s \in \{top, bot\}$, in which case the probability can be upper bounded by $\frac{6i}{2^c - 3q}$.

We henceforth assume $\neg\mathbf{AUX}_i^3$, so that the procedure `NodeCollection`() returns only one node (X^{top}, X^{bot}). In the ideal world, the simulator has no problem to solve the consistency equation, and inserts to the graph one more node (C^{top}, C^{bot}). Now, (C^{top}, C^{bot}) does not contain as much randomness as the first iteration nodes, since one of the permutation calls is already fixed. However, we are going to make use of the fact that second iteration edges are *rarely* produced. W.l.o.g., we can assume that X^{top} appeared in the query history, and let $y^{top} := \texttt{tabP}_{top}[X^{top}]$, and $Y^{bot} := \texttt{tabP}_{bot}[X^{bot}]$; in particular, the only fresh random value is Y^{bot}. In the following we compute the probability that this node triggers **BAD** using the condition $N_{\mathbf{2Col}} = m$.

2. **AUX**^{1}*.* This event implies that (C^{top}, C^{bot}) collides with (A^{top}, A^{bot}) or (B^{top}, B^{bot}), thus

$$\begin{aligned} y^{top} + 2Y^{bot} &\overset{c}{=} Y_k^{top} + 2Y_k^{bot} \\ \text{or } 2y^{top} + Y^{bot} &\overset{c}{=} 2Y_k^{top} + Y_k^{bot} \end{aligned}$$

for some $k \in \{1, 2\}$, where we remind that Y_k^s is the random variable equal to $\texttt{tabP}_s[X_k^s]$, fixed during the first node collection. This event happens with probability at most $\dfrac{4}{2^c - 3q}$.

3. **AUX**^{2}*.* The event **AUX**^{2} only concerns inverse queries, thus it does not apply here.

4. **3Col***.* This event means that there exist $(N_1^{top}, N_1^{bot}), (N_2^{top}, N_2^{bot}) \in \texttt{Rooted}$ distinct such that

$$\begin{aligned} C^{top} &\overset{c}{=} N_1^{top} \overset{c}{=} N_2^{top} \\ \text{or } C^{bot} &\overset{c}{=} N_1^{bot} \overset{c}{=} N_2^{bot} , \end{aligned}$$

or in more detail,

$$\begin{aligned} y^{top} + 2Y^{bot} &\overset{c}{=} N_1^{bot} \overset{c}{=} N_2^{bot} \\ \text{or } 2y^{top} + Y^{bot} &\overset{c}{=} N_1^{top} \overset{c}{=} N_2^{top} . \end{aligned}$$

Similarly to the first iteration nodes, this event happens with probability at most $\dfrac{m}{2(2^c - 3q)}$.

5. **Double2Col***.* Again, there are two possibilities:

– (C^{top}, C^{bot}) *displays* a double collision: there exist (N_1^{top}, N_1^{bot}), $(N_2^{top}, N_2^{bot}) \in \texttt{Rooted}$ (not necessarily distinct) such that

$$\begin{aligned} C^{top} &\overset{c}{=} N_1^{top} \\ \text{and } C^{bot} &\overset{c}{=} N_2^{bot} . \end{aligned}$$

Denoting $(M_1^{top}, M_1^{bot}) = MIX^{-1}(N_1^{top}, N_1^{bot})$, and $(M_2^{top}, M_2^{bot}) = MIX^{-1}(N_2^{top}, N_2^{bot})$, this reduces to

$$\begin{aligned} y^{top} &\overset{c}{=} M_1^{top} \\ \text{and } Y^{bot} &\overset{c}{=} M_2^{bot} . \end{aligned} \tag{12}$$

Now, the first term of the equation does not provide any randomness. Nevertheless, **Double2Col** with a second iteration node implies that the following two sub-events happened:

- $\mathbf{E}_1$: The node (A^{top}, A^{bot}) or (B^{top}, B^{bot}) has the inner part of its top part already in the query history. By the first paragraph of Section 5.1.2, the probability is upper bounded by $\dfrac{6i}{2^c - 3q}$;
- $\mathbf{E}_2$: There exists M_2^{bot} as above such that Y^{bot} satisfies the second equation of (12). This happens with probability at most $\dfrac{3i}{2^c - 3q}$.

$\mathbf{E}_1$ and $\mathbf{E}_2$ are independent, and we need to consider the symmetric case where the bottom part of (A^{top}, A^{bot}) or (B^{top}, B^{bot}) was already in the query history. Therefore this case of **Double2Col** happens with probability at most $\left(\dfrac{6i}{2^c - 3q}\right)^2$.

– (C^{top}, C^{bot}) is *part* of a double collision: there exists $(N_1^{top}, N_1^{bot}), (N_2^{top}, N_2^{bot}) \in \texttt{Rooted}$ (not necessarily distinct) such that

$$\begin{cases} N_1^{top} \overset{c}{=} C^{top}\,, \\ N_1^{bot} \overset{c}{=} N_2^{bot}\,, \end{cases} \quad \text{or} \quad \begin{cases} N_1^{top} \overset{c}{=} N_2^{top}\,, \\ N_1^{bot} \overset{c}{=} C^{bot}\,. \end{cases}$$

Similarly to the first iteration nodes, this gives a probability of at most $\dfrac{m}{2^c - 3q}$.

6. **BadNode***.* Since $\mathbf{BadNode}_i$ on the node (C^{top}, C^{bot}) implies $\mathbf{ThreeRounds}_i$, we can skip this probability computation.

5.1.3 Third Iteration Events, Forward Query Now, to guarantee $\neg\mathbf{ThreeRounds}_i$, we need that (C^{top}, C^{bot}) (if it exists) does not have any of its inner parts appearing in the query history. Again, this event implies the following two sub-events:

– An inner part of a first iteration node appears in the query history: by the first paragraph of Section 5.1.2, this event happens with probability at most $\dfrac{12i}{2^c - 3q}$;
– An inner part of a second iteration node appears in the query history: since $\neg\mathbf{AUX}_i^3$ guarantees that at most one node was produced during the second iteration, this event happens with probability at most $\dfrac{6i}{2^c - 3q}$.

Both of these events are independent, therefore $\mathbf{ThreeRounds}_i$ happens with probability at most $2\left(\dfrac{6i}{2^c - 3q}\right)^2$.

5.1.4 The Case of Inverse Queries This case is similar to the second-iteration events, since inverse queries *rarely* hit the tree, i.e., they hit one of the inner parts of the tree with probability at most $\dfrac{6i}{2^c - 3q}$. Moreover, the event $\mathbf{AUX}^2$ is set with probability at most $\dfrac{m}{2^c - 3q}$. Assuming then $\neg\mathbf{AUX}^2$, **NodesToAnswer** contains only one node and the behavior of the simulator and `GraphProc` is the same as if a second iteration node was produced with one of its top part in the query history, and the probability computation is the same. Consequently,

$$\mathbf{Pr}\left(\mathbf{BAD}_i \mid N_{\mathbf{2Col}} = m \wedge \neg\mathbf{BAD}_{i-1} \wedge \text{query } i \text{ is inverse}\right) \leq \\ \mathbf{Pr}\left(\mathbf{BAD}_i \mid N_{\mathbf{2Col}} = m \wedge \neg\mathbf{BAD}_{i-1} \wedge \text{query } i \text{ is forward}\right)\,.$$

5.1.5 Conclusion From Sections 5.1.1 to 5.1.4, we know that

$$\mathbf{Pr}_x\left(\mathbf{BAD}_i \mid N_{\mathbf{2Col}} = m \wedge \neg\mathbf{BAD}_{i-1}\right) \\ \leq \begin{cases} 180\left(\frac{i}{2^c-3q}\right)^2 + \frac{6}{2^c-3q} + \frac{9m}{2(2^c-3q)} & \text{if } i \notin \xi^{top} \cup \xi^{bot}\,, \\ 180\left(\frac{i}{2^c-3q}\right)^2 + \frac{6}{2^c-3q} + \frac{9m}{2(2^c-3q)} + \frac{6q}{2^c-3q} & \text{otherwise}\,, \end{cases} \tag{13}$$

where we remind that ξ^s is the number of queries such that there exists a node produced during the first iteration in $S_{\mathbf{2Col}}(s)$ (see (11)). The conditions under which we study the corresponding **BAD** events guarantee that ξ^{top} and ξ^{bot} are disjoint, and $|\xi^{top}|+|\xi^{bot}| = m$. Therefore, this gives

$$\begin{aligned}\mathbf{Pr}_x\left(\mathbf{BAD} \mid N_{\mathbf{2Col}} = m\right) \leq \sum_{i=1}^{q} \left(180\left(\frac{i}{2^c-3q}\right)^2 + \frac{6}{2^c-3q} + \frac{9m}{2(2^c-3q)}\right) + \\ \sum_{i\in\xi^{top}\cup\xi^{bot}} \frac{6q}{2^c-3q} \\ \leq \frac{60q^3}{(2^c-3q)^2} + \frac{90q^2}{(2^c-3q)^2} + \frac{30q}{(2^c-3q)^2} + \frac{6q}{2^c-3q} + \\ \frac{21mq}{2(2^c-3q)}\,.\end{aligned}$$

Now, plugging this equation into (7) gives

$$\begin{aligned}\mathbf{Pr}_x\left(\mathbf{BAD}\right) &\leq \frac{60q^3}{(2^c-3q)^2} + \frac{90q^2}{(2^c-3q)^2} + \frac{30q}{(2^c-3q)^2} + \frac{6q}{2^c-3q} + \\ &\sum_{m=0}^{3q+1} \mathbf{Pr}_x\left(N_{\mathbf{2Col}} = m\right)\frac{21mq}{2(2^c-3q)} \\ &\leq \frac{60q^3}{(2^c-3q)^2} + \frac{90q^2}{(2^c-3q)^2} + \frac{30q}{(2^c-3q)^2} + \frac{6q}{2^c-3q} + \\ &\frac{21q}{2(2^c-3q)}\mathsf{E}\left(N_{\mathbf{2Col}}\right)\,. \qquad (14)\end{aligned}$$

It now remains to compute $\mathsf{E}\left(N_{\mathbf{2Col}}\right)$. For $i \in \{1,\ldots,|\texttt{Trimmed}|\}$, denote by (N_i^{top}, N_i^{bot}) the i^{th} element in `Trimmed`. Moreover, we define the Bernoulli variable V_i equal to 1 if and only if $(N_i^{top}, N_i^{bot}) \in S_{\mathbf{2Col}}$. Then,

$$\begin{aligned}\mathsf{E}\left(N_{\mathbf{2Col}}\right) = \mathsf{E}\left(\sum_{i=1}^{|\texttt{Trimmed}|} V_i\right) &= \sum_{i=1}^{|\texttt{Trimmed}|} \mathsf{E}\left(V_i\right) \\ &= \sum_{i=1}^{|\texttt{Trimmed}|} 2\mathbf{Pr}\left(\exists j < i, s \in \{top, bot\} \text{ such that } N_i^s \overset{c}{=} N_j^s\right) \\ &\leq 2\sum_{i=1}^{|\texttt{Trimmed}|} \frac{2(i-1)}{2^c-3q} \\ &= \frac{4}{2^c-3q}\,\frac{|\texttt{Trimmed}|\left(|\texttt{Trimmed}|-1\right)}{2} \\ &\leq \frac{18q^2}{2^c-3q} + \frac{6q}{2^c-3q}\,, \qquad (15)\end{aligned}$$

where the last inequality uses $|\texttt{Trimmed}| \leq 3q+1$. Finally, plugging (15) into (14), we obtain

$$\begin{aligned}\mathbf{Pr}_x\left(\mathbf{BAD}\right) &\leq \frac{60q^3}{(2^c-3q)^2} + \frac{90q^2}{(2^c-3q)^2} + \frac{30q}{(2^c-3q)^2} + \frac{6q}{2^c-3q} + \\ &\frac{21q}{2(2^c-3q)}\left(\frac{18q^2}{2^c-3q} + \frac{6q}{2^c-3q}\right) \\ &\leq \frac{249q^3}{(2^c-3q)^2} + \frac{153q^2}{(2^c-3q)^2} + \frac{30q}{(2^c-3q)^2} + \frac{6q}{2^c-3q} \\ &\leq \frac{249q^3}{(2^c-3q)^2} + \frac{30q}{(2^c-3q)^2} + \frac{19q}{2^c-3q}\,.\end{aligned}$$

If we assume that $30 < 2^c$, then we obtain the upper bound

$$\mathbf{Pr}_x\left(\mathbf{BAD}\right) \leq \frac{249q^3}{(2^c - 3q)^2} + \frac{21q}{2^c - 3q}\,.$$

When $x = I$, $q = q_{\mathcal{P}}$, while when $x = \mathit{IM2}$, $q \leq \sigma + q_{\mathcal{P}}$, hence the result. $\square$

5.2 World W_I Versus World W_{IM1} as Long as no BAD

Lemma 3 argues that as long as no **BAD** happens, the ideal and intermediate worlds are indistinguishable.

Lemma 3. *For any distinguisher $\mathcal{D}$,*

$$\left|\mathbf{Pr}\left(\mathcal{D}^{W_I} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM1}} = 1 \mid \neg\mathbf{BAD}\right)\right| = 0\,.$$

Proof. The only difference between the worlds W_I and W_{IM1} lies in the consistency of the responses. In more detail, as long as the simulator is $\mathcal{RO}$-consistent with respect to $\mathcal{H}$ (c.f., Definition 1), then W_I and W_{IM1} are perfectly indistinguishable. We stress that for $s \in \{top, bot\}$, an s inverse query hitting the tree does not necessarily break the consistency, since the simulator uses the $\bar{s}$ part to guarantee consistency. Therefore, the $\mathcal{RO}$-consistency boils down to ensuring that the procedure `LinSolve` () never returns $\perp$, which we argue in the following.

From the remark at the beginning of the proof of Lemma 1, we know that as long as no **BAD** happens, $|\mathbf{NodesToAnswer}| \leq 2$ for any iteration of the simulator. When $|\mathbf{NodesToAnswer}| = 1$, if **BadNode** is not set, the linear system comprises only one equation with at least one unknown, thus the simulator can always output consistent answers. Therefore we henceforth consider the case where $|\mathbf{NodesToAnswer}| = 2$. W.l.o.g., assume that $\mathbf{NodesToAnswer} = \{(X_1^{top}, X_1^{bot}), (X_2^{top}, X_2^{bot})\}$ where $X_1^{top} = X_2^{top}$. Now, the form of the underlying linear system depends on the direction of the query and the iteration run by the simulator. In the following we study each case.

Forward query, first iteration. The consistency equation is of the form

$$\begin{aligned} a_1^{top} \oplus a_1^{bot} &= h_1\,, \\ a_1^{top} \oplus a_2^{bot} &= h_2\,, \end{aligned} \tag{16}$$

where a_1^{top} is the variable corresponding to $\mathrm{outer}_r\left(\mathtt{tabP}_{top}[X_1^{top}]\right) = \mathrm{outer}_r\left(\mathtt{tabP}_{top}[X_2^{top}]\right)$, a_1^{bot} to $\mathrm{outer}_r\left(\mathtt{tabP}_{bot}[2X_1^{bot}]\right)$, and a_2^{bot} to $\mathrm{outer}_r\left(\mathtt{tabP}_{bot}[2X_2^{bot}]\right)$. This equation has a solution, since the simulator can choose $a_1^{top} \xleftarrow{\$} \{0,1\}^r$, and infer a_1^{bot}, a_2^{bot}.

Inverse query, first iteration. In this case, since **BAD** (and in particular $\mathbf{AUX}^2$) does not happen, $|\mathbf{NodesToAnswer}| \leq 1$.

Second iteration. Because of $\neg\mathbf{AUX}^3$, $|\mathbf{NodesToAnswer}| \leq 1$.

Therefore, the simulator can always produce consistent answers. $\square$

5.3 World W_{IM1} Versus World W_{IM2} as Long as no BAD

In this section we upper bound the distinguisher's advantage between W_{IM1} and W_{IM2} without **BAD**.

Lemma 4. *For any distinguisher $\mathcal{D}$ making $q_{\mathcal{P}}$ primitive queries and construction queries which would correspond to a total of σ primitive queries in world W_R, one has*

$$\left|\mathbf{Pr}\left(\mathcal{D}^{W_{IM1}} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} = 1 \mid \neg\mathbf{BAD}\right)\right| \leq \frac{3q^{\frac{3}{2}}}{2^c - 3q},$$

where $q := \sigma + q_{\mathcal{P}}$.

The proof is deferred to Section 5.3.3. First, we introduce a distinguishing game useful for the proof in Section 5.3.1, and upper bound the statistical distance between the two underlying distributions in Section 5.3.2.

5.3.1 A Simpler Distinguishing Game

We first introduce a simple distinguishing game whose statistical distance computation is crucial for the proof of Lemma 4. In the following, let $\phi : \{0,1\}^b \to \{0,1\}^b$ be a bijection.

World W_1. In this world, the elements in $\{0,1\}^b$ are partitioned into 2^r buckets $(B_k)_{k=0,\ldots,2^r-1}$ in the following way:

$$\forall x \in \{0,1\}^b,\ x \in B_k \iff \mathrm{outer}_r\left(\phi(x)\right) = k\,.$$

Each bucket contains 2^c elements. Then, on a forward query, an element $k \in \{0,1\}^r$ is drawn uniformly at random, and the answer is the result of a sampling without replacement in bucket B_k. On an inverse query with $y \in \{0,1\}^b$, a permutation-consistent element with lazy sampling is returned, and y is withdrawn from bucket $B_{\mathrm{outer}_r(y)}$.

World W_2. This world implements a random permutation over $\{0,1\}^b$, where the elements are returned using lazy sampling.

Intuitively, W_2 corresponds to the permutations $\mathcal{P}_{top}$ and $\mathcal{P}_{bot}$, and we will argue later that W_1 is an abstraction of the simulator way of sampling the responses.

5.3.2 Upper Bounding the Statistical Distance Using the χ^2 Method

Lemma 5 upper bounds the statistical distance between W_1 and W_2. The proof uses the χ^2 method by Dai et al. [DHT17] that we describe in the following. Let $\Omega := \{0,1\}^b$ be a sample space, and consider two probability distributions P_1 and P_2 with values in Ω^q. Let $\boldsymbol{x} := (x_1, \ldots, x_q)$ be a random vector following P_1, and for any $i \in \{1, \ldots, q\}$, let $\boldsymbol{x}^i := (x_1, \ldots, x_i)$. The χ^2 divergence is defined as follows:

$$\chi^2(\boldsymbol{x}^{i-1}) := \sum_{\substack{x\in\Omega,\\ \mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)\neq 0}} \frac{\left(\mathbf{Pr}_1\left(x_i = x \mid \boldsymbol{x}^{i-1}\right) - \mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)\right)^2}{\mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)}\,.$$

Moreover, assume that the support of P_1 is included in the support of P_2. Then the χ^2 method upper bounds the statistical distance between P_1 and P_2 as follows:

$$\mathbf{Adv}^{\mathrm{ind}}\left(P_1, P_2\right)(q) \leq \left(\frac{1}{2}\sum_{i=1}^{q} \mathsf{E}\left(\chi^2\left(\boldsymbol{x}^{i-1}\right)\right)\right)^{\frac{1}{2}}, \tag{17}$$

where we abuse notation for $\mathbf{Adv}^{\mathrm{ind}}\left(P_1, P_2\right)(q)$.

Lemma 5. *For any $q \leq 2^c$,*

$$\mathbf{Adv}^{\mathrm{ind}}\left(W_1, W_2\right)(q) \leq \frac{1}{2}\frac{q^{\frac{3}{2}}}{2^c - q}\,.$$

Proof. In the following, if $\boldsymbol{x}$ is a vector of length $k > 1$ with values in $\{0,1\}^b$, let

$$\mathcal{P}\mathrm{Cons}(\boldsymbol{x}) := \{y \in \{0,1\}^b \mid \forall i \in \{1,\ldots,k\}, y \neq \boldsymbol{x}_i\}\,.$$

Because the inverse queries in W_1 and W_2 trigger the same sampling procedure (thus a zero term in the chi-squared divergence), we can w.l.o.g., focus on forward queries. Denote by P_1 (resp., P_2) the associated distribution in W_1 (resp., W_2). In both cases, let `tabP` be the dictionary that logs the query history, i.e., if $\texttt{tabP}[x] = y$, then a forward query with x gives y, and vice versa for inverse queries. For $i \in \{1,\ldots,q\}$, let $\boldsymbol{x}^{i-1}$ be a random vector of $i-1$ elements sampled according to P_1. Let $x \in \mathcal{P}\mathrm{Cons}(\boldsymbol{x}^{i-1})$, we then define $S(x)$ as follows:

$$S(x) = |\{y \in \mathbf{Img}(\texttt{tabP}) \mid \mathrm{outer}_r(y) = \mathrm{outer}_r(\phi(x))\}|\ .$$

i.e., the number of elements that have already been withdrawn from bucket $B_{\mathrm{outer}_r(\phi(x))}$. Now, in order to obtain x in W_1, the adversary must first sample $\mathrm{outer}_r(\phi(x))$, and then draw the appropriate element in a bucket of size $2^c - S(x)$. Thus

$$\begin{aligned}\mathbf{Pr}_1\left(x_i = x \mid \boldsymbol{x}^{i-1}\right) &= \frac{1}{2^r}\frac{1}{2^c - S(x)}\,,\\ \mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right) &= \frac{1}{2^b - (i-1)}\,.\end{aligned}$$

Therefore, using that $0 \le S(x) \le i-1 \le q \le 2^c$,

$$\begin{aligned}&\left|\frac{\mathbf{Pr}_1\left(x_i = x \mid \boldsymbol{x}^{i-1}\right) - \mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)}{\mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)}\right|\\ &= \left|\frac{2^b - (i-1) - 2^b + 2^r S(x)}{2^b - 2^r S(x)}\right|\\ &= \left|\frac{2^r S(x) - (i-1)}{2^b - 2^r S(x)}\right|\\ &\le \frac{i-1}{2^c - q}\,.\end{aligned}$$

Using the obtained equation, we can compute the χ^2 divergence:

$$\begin{aligned}\chi^2(\boldsymbol{x}^{i-1}) &= \sum_{x \in \mathcal{P}\mathrm{Cons}(\boldsymbol{x}^{i-1})} \mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)\left(\frac{\mathbf{Pr}_1\left(x_i = x \mid \boldsymbol{x}^{i-1}\right) - \mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)}{\mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)}\right)^2\\ &\le \sum_{x \in \mathcal{P}\mathrm{Cons}(\boldsymbol{x}^{i-1})} \mathbf{Pr}_2\left(x_i = x \mid \boldsymbol{x}^{i-1}\right)\left(\frac{i-1}{2^c - q}\right)^2\\ &= \left(\frac{i-1}{2^c - q}\right)^2\,.\end{aligned}$$

Now, we can apply the χ^2 technique to obtain

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ind}}(W_1, W_2)(q) &\le \left(\frac{1}{2}\sum_{i=1}^{q} \mathsf{E}\left(\chi^2\left(\boldsymbol{x}^{i-1}\right)\right)\right)^{\frac{1}{2}}\\ &\le \left(\frac{1}{2}\sum_{i=1}^{q}\left(\frac{i-1}{2^c - q}\right)^2\right)^{\frac{1}{2}}\\ &\le \frac{1}{2}\frac{q^{\frac{3}{2}}}{2^c - q}\,.\end{aligned}$$

□

Remark 3. This distinguishing problem is similar to the one of Bhattacharya and Nandi's work [BN18, Theorem 2]. They have an upper bound of the form

$$\mathcal{O}\left(\frac{q}{2^{\frac{b+c}{2}}}\right).$$

However, in their setting the adversary only makes forward queries. In our case, allowing inverse queries introduces a supplementary bias in the distribution that can be upper bounded by $\frac{q^{\frac{3}{2}}}{2^c}$.

5.3.3 Proof of Lemma 4 Using Lemma 5, we can now prove Lemma 4. Consider $\mathcal{D}$ which aims at distinguishing W_{IM1} and W_{IM2} without **BAD**. We decompose the $q_{\mathcal{P}}$ primitive queries into $q_{\mathcal{P}}^{top}$ top queries and $q_{\mathcal{P}}^{bot}$ bottom queries, so that $q_{\mathcal{P}} = q_{\mathcal{P}}^{top} + q_{\mathcal{P}}^{bot}$. From $\mathcal{D}$, we build another distinguisher $\mathcal{D}_1$ which converts all of the construction queries into the appropriate primitive queries, so that $\mathcal{D}_1$ makes at most $\sigma/2+q_{\mathcal{P}}^{top}$ (resp., $\sigma/2+q_{\mathcal{P}}^{bot}$) top (resp., bottom) primitive queries. Since the construction component is the same in both worlds, doing this provides at least the same amount of information to the adversary. Define W_a as W_{IM1} without the component $\mathcal{H}$, and similarly, let W_d comprise W_{IM2} without the component $\mathcal{H}$. Then, we have

$$\left|\mathbf{Pr}\left(\mathcal{D}^{W_{IM1}} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} = 1 \mid \neg\mathbf{BAD}\right)\right| \leq \\ \left|\mathbf{Pr}\left(\mathcal{D}_1^{W_a} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}_1^{W_d} = 1 \mid \neg\mathbf{BAD}\right)\right|.$$

Now, we remark that the simulator embeds two distinct primitives, each one simulating the top or bottom permutation. Given the current simulator definition, it is not clear whether these two primitives are independent. Indeed, a top (resp., bottom) query can trigger a bottom (resp., top) output being defined, and the outer parts are chosen according to an equation involving top and bottom outer parts. To address these points, we introduce two intermediate worlds W_b and W_c, as illustrated in Fig. 4.

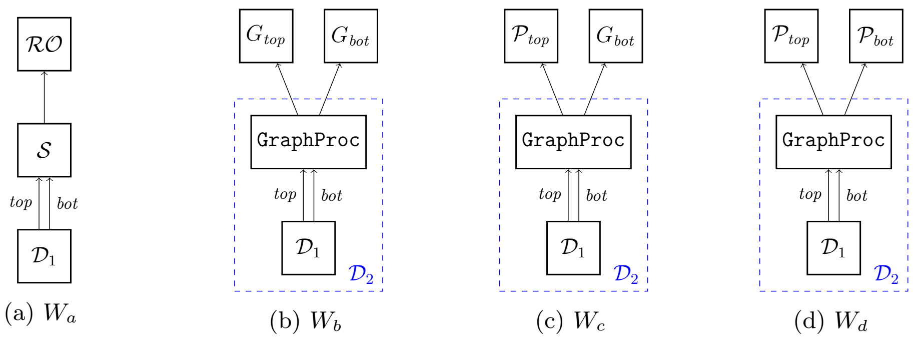


Fig. 4: World decomposition for the proof of Lemma 4. W_a and W_d correspond respectively to W_{IM1} and W_{IM2} (without the construction oracle). $\mathcal{D}_2$ is a distinguisher built from $\mathcal{D}_1$ when distinguishing W_b, W_c, and W_d.

World W_b. W_b differs from W_a in the way the dictionaries $\texttt{DictOuterParts}_{top}$ and $\texttt{DictOuterParts}_{bot}$ are generated. Namely, the function LinSolve () is replaced by a uniform sampling of the outer parts. Given this description, it is now clear that the two primitives have independent sampling procedures. In Fig. 4b, they are represented by the components G_{top} and G_{bot} which take as parameter a bit $\beta \in \{0,1\}$, and for every fresh query, G_s operates as follows:

- If $\beta = 0$ or the query is in the inverse direction, it returns a permutation-consistent answer uniformly at random;
- Otherwise it samples $h \xleftarrow{\$} \{0,1\}^r$, and returns a permutation-consistent answer y uniformly at random with the restriction that

$$h = \begin{cases} \text{outer}_r(y) & \text{if } s = top\,, \\ \text{outer}_r(2y) & \text{if } s = bot\,. \end{cases}$$

The component `GraphProc` invokes the procedures G_{top} and G_{bot} with $\beta = 0$ whenever the node collection phase returns an empty set (corresponding to line 11 of Algorithm 3). Looking ahead, G_{top} and G_{bot} with β always equal to 1 correspond to W_1 defined in Section 5.3.1.

Now, recall that in W_a the outer parts returned by `LinSolve`() come from $\mathcal{RO}$ calls combined with uniform sampling. In particular, because no **BAD** event occurs, these values are all freshly random, so that W_b is just a rearrangement of W_a. Therefore,

$$\left|\mathbf{Pr}\left(\mathcal{D}_1^{W_a} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}_1^{W_b} = 1 \mid \neg\mathbf{BAD}\right)\right| = 0\,.$$

New distinguisher. From $\mathcal{D}_1$ we build yet another distinguisher $\mathcal{D}_2$ which obtains for free all of the subsequent queries defined by the simulator or the primitives G_{top} and G_{bot}. In other words, $\mathcal{D}_2$ runs the component `GraphProc` on its own (see Fig. 4). Since no **BAD** happens, $\mathcal{D}_2$ makes at most $3(\sigma/2+q_{\mathcal{P}}^{top})$ (resp., $3(\sigma/2+q_{\mathcal{P}}^{bot})$) top (resp., bottom) primitive queries, and clearly,

$$\left|\mathbf{Pr}\left(\mathcal{D}_1^{W_b} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}_1^{W_d} = 1 \mid \neg\mathbf{BAD}\right)\right| \le \left|\mathbf{Pr}\left(\mathcal{D}_2^{W_b} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}_2^{W_d} = 1 \mid \neg\mathbf{BAD}\right)\right|\,.$$

World W_c. In this world, the left sampler G_{top} is replaced by an ideal permutation. Since G_{top} and G_{bot} are independent and $\mathcal{D}_2$ runs `GraphProc` by itself, we obtain

$$\left|\mathbf{Pr}\left(\mathcal{D}_2^{W_b} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}_2^{W_c} = 1 \mid \neg\mathbf{BAD}\right)\right| \le \mathbf{Adv}^{\text{ind}}\left(G_{top}, \mathcal{P}_{top}\right)\left(3(\sigma/2 + q_{\mathcal{P}}^{top})\right)\,.$$

Likewise, in W_d, the right sampler G_{bot} is replaced by an ideal permutation, and we obtain

$$\left|\mathbf{Pr}\left(\mathcal{D}_2^{W_c} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}_2^{W_d} = 1 \mid \neg\mathbf{BAD}\right)\right| \le \mathbf{Adv}^{\text{ind}}\left(G_{bot}, \mathcal{P}_{bot}\right)\left(3(\sigma/2 + q_{\mathcal{P}}^{bot})\right)\,.$$

Conclusion. Using the triangle inequality and the results above,

$$\left|\mathbf{Pr}\left(\mathcal{D}^{W_{IM1}} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} = 1 \mid \neg\mathbf{BAD}\right)\right| \le \mathbf{Adv}^{\text{ind}}\left(G_{top}, \mathcal{P}_{top}\right)\left(3(\sigma/2 + q_{\mathcal{P}}^{top})\right) + \mathbf{Adv}^{\text{ind}}\left(G_{bot}, \mathcal{P}_{bot}\right)\left(3(\sigma/2 + q_{\mathcal{P}}^{bot})\right)\,.$$

We remark that the chi-squared divergence of $G_{top}/\mathcal{P}_{top}$ (resp. $G_{bot}/\mathcal{P}_{bot}$) is always smaller than the one of W_1/W_2 from Section 5.3.1. Indeed, in the distinguishing game $G_s/\mathcal{P}_s$, when $\beta = 1$, the conditional probabilities are the same ones as in the game W_1/W_2, while when $\beta = 0$, the conditional probabilities in G_{top} and $\mathcal{P}_{top}$ are equal, giving a zero term in the chi-squared divergence. Therefore, we can use the result of Lemma 5, and obtain

$$\mathbf{Adv}^{\text{ind}}\left(G_{top}, \mathcal{P}_{top}\right)\left(3(\sigma/2 + q_{\mathcal{P}}^{top})\right) \le \frac{1}{2}\frac{(3(\sigma/2 + q_{\mathcal{P}}^{top}))^{\frac{3}{2}}}{2^c - (3(\sigma/2 + q_{\mathcal{P}}))}\,,$$

$$\mathbf{Adv}^{\text{ind}}\left(G_{bot}, \mathcal{P}_{bot}\right)\left(3(\sigma/2 + q_{\mathcal{P}}^{bot})\right) \le \frac{1}{2}\frac{(3(\sigma/2 + q_{\mathcal{P}}^{bot}))^{\frac{3}{2}}}{2^c - (3(\sigma/2 + q_{\mathcal{P}}))}\,,$$

Finally, defining $q := \sigma + q_{\mathcal{P}}$, this gives

$$\left|\mathbf{Pr}\left(\mathcal{D}^{W_{IM1}} = 1 \mid \neg\mathbf{BAD}\right) - \mathbf{Pr}\left(\mathcal{D}^{W_{IM2}} = 1 \mid \neg\mathbf{BAD}\right)\right| \leq \frac{3q^{\frac{3}{2}}}{2^c - 3q},$$

which concludes the proof.

6 Tightness of the Bound

In this section, we argue about the tightness of the bound of Theorem 1 by providing two attacks. The first one, described in Section 6.1, has an advantage of $\mathcal{O}\left(\frac{q^2}{2^{2c+r}}\right)$. This attack is not specific to our simulator, since it can be used to mount collision and second preimages (which can then be used to differentiate). The second attack, as explained in Section 6.2, has an advantage of $\Omega\left(\frac{q^3}{2^{2c+r}}\right)$ and only works against our simulator of Section 4.2.

6.1 Simple Birthday Attack

The birthday attack is similar to the best-known differentiability attack of the sponge construction [BDPV07]. It consists of finding a full-state collision in the nodes, i.e., find (A^{top}, A^{bot}), $(B^{top}, B^{bot}) \in$ `Rooted`, $m \in \{0,1\}^r$ such that

$$(A^{top}, A^{bot}) = (B^{top} \oplus (m\|0^c), B^{bot} \oplus (m\|0^c)) \,. \tag{18}$$

To do so, the distinguisher performs q top and bottom queries to expand rooted paths, so that it obtains a set of rooted nodes $\{(N_i^{top}, N_i^{bot})\}_{i=1,\ldots,q}$, and searches for (A^{top}, A^{bot}), (B^{top}, B^{bot}) in this list that satisfy (18). The probability to find such nodes is then $\approx \dfrac{q^2}{2^{2c+r}}$. Once this is done, the distinguisher adds one edge from (A^{top}, A^{bot}) with a zero message. Consequently, an edge from (B^{top}, B^{bot}) with the message m is also added. The distinguisher can then retrieve the unique messages and indexes M_A, k_A and M_B, k_B associated to (A^{top}, A^{bot}) and $(B^{top} \oplus (m\|0^c), B^{bot} \oplus (m\|0^c))$. Finally, it queries `ConsQuery` (M_A, k_A) and `ConsQuery` (M_B, k_B) and checks whether the two responses are equal. If this is the case, then the distinguisher returns W_R, otherwise it returns W_I. Indeed, consistency is always satisfied in the real world, while in the ideal world, with high probability the two $\mathcal{RO}$ calls are distinct. This attack succeeds with high probability when $q = 2^{c+r/2}$. Note that this attack works against any simulator, and therefore one cannot get better than $c + r/2$ bits of security.

6.2 A Simulator-Specific Attack

We show a differentiability attack with an advantage of $\approx \left(1 - \frac{1}{2^r}\right) \frac{q^3}{2^{2c+r}}$. Consider the following distinguisher $\mathcal{D}$:

1. $\mathcal{D}$ first starts by making q forward top and bottom primitive queries and obtains for $i \in \{1, \ldots, q\}$ the queries $X_i^{top} \to Y_i^{top}$ and $X_i^{bot} \to Y_i^{bot}$;
2. Then it makes q top and bottom primitive queries to expand rooted paths. The top queries are of the form $(A_j^{top} + (m\|0^c))$, and the bottom queries of form $(A_j^{bot} + (m\|0^c))$, where $(A_j^{top}, A_j^{bot}) \xleftarrow{\$}$ `Rooted`, and $m \xleftarrow{\$} \{0,1\}^r$. After this step, $\mathcal{D}$ obtains q rooted notes, and for any $i \in \{1, \ldots, q\}$, the ith node is denoted by (A_i^{top}, A_i^{bot});
3. The distinguisher checks whether the event **BadNode_Strong** is set, i.e., there exist $m \in \{0,1\}^r$, $i, j, k \in \{1, \ldots, q\}$ such that $A_i^{top} = X_j^{top} \oplus (m\|0^c)$ and $A_i^{bot} = X_k^{bot} \oplus (m\|0^c)$;

4. If such a collision is found, the distinguisher makes the construction query associated to $(A_i^{top} \oplus (m\|0^c), A_i^{bot} \oplus (m\|0^c))$, and returns 1 if the answer is consistent and 0 otherwise;
5. Otherwise, if **BadNode_Strong** is not set, $\mathcal{D}$ returns 0.

In the following, we abbreviate **BadNode_Strong** to **BNS**. $\mathcal{D}$ makes at most $4q$ primitive queries and one construction query. In the real world, the answers are always consistent, and in the ideal world, the answer is consistent with probability $\frac{1}{2^r}$, therefore

$$\begin{aligned}\mathbf{Pr}\left(\mathcal{D}^{W_R} = 1 \mid \mathbf{BNS}\right) &= 1\,,\\ \mathbf{Pr}\left(\mathcal{D}^{W_I} = 1 \mid \mathbf{BNS}\right) &= \frac{1}{2^r}\,.\end{aligned}$$

Moreover, in W_R and W_I, the probability that one fresh edge sets $\mathbf{BNS}_i$ is $\approx \frac{q^2}{2^{2c+r}}$, thus

$$\mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{BNS}\right) \approx \mathbf{Pr}\left(\mathcal{D}^{W_R} \text{ sets } \mathbf{BNS}\right) \approx \frac{q^3}{2^{2c+r}}\,.$$

Therefore, noting that $\mathcal{D}$ outputs 0 when **BNS** is not set,

$$\begin{aligned}\mathbf{Pr}\left(\mathcal{D}^{W_R} = 1\right) &= \mathbf{Pr}\left(\mathcal{D}^{W_R} = 1 \mid \mathbf{BNS}\right)\mathbf{Pr}\left(\mathcal{D}^{W_R} \text{ sets } \mathbf{BNS}\right) \approx \frac{q^3}{2^{2c+r}}\,,\\ \mathbf{Pr}\left(\mathcal{D}^{W_I} = 1\right) &= \mathbf{Pr}\left(\mathcal{D}^{W_I} = 1 \mid \mathbf{BNS}\right)\mathbf{Pr}\left(\mathcal{D}^{W_I} \text{ sets } \mathbf{BNS}\right) \approx \frac{1}{2^r}\frac{q^3}{2^{2c+r}}\,.\end{aligned}$$

So that

$$\begin{aligned}\mathbf{Adv}^{\text{indif}}_{\mathcal{H}^{\mathcal{P}_{top},\mathcal{P}_{bot}},\mathcal{S}}(\mathcal{D}) &= \left|\mathbf{Pr}\left(\mathcal{D}^{W_R} = 1\right) - \mathbf{Pr}\left(\mathcal{D}^{W_I} = 1\right)\right|\\ &\approx \frac{q^3}{2^{2c+r}}\left(1 - \frac{1}{2^r}\right)\,.\end{aligned}$$

Setting $q = 2^{\frac{2c+r}{3}}$ thus gives a high probability success. This means that our simulator cannot give a security bound better than $\frac{q^3}{2^{2c+r}}$.

Note that there is a (small) gap of $r/3$ bits of security between this attack and our proof. This means that either the attack can be improved or (more likely) our simulator has a better security bound. However proving a better security bound would be extremely challenging. Indeed, in our proof, we do allow inner collisions (as we aim for security beyond the birthday bound), but restrict ourselves to nodes which do not have more than one single collision on their inner part. This was possible thanks to the introduction of dedicated bad events (namely **Double2Col** and **3Col**) which are set after approximately $2^{2c/3}$ queries. In order to obtain beyond $2c/3$ bits of security, we would have to release this restriction, thus allow higher degree collisions, and possibly allow partial edges to be added to the graph. This would incur a complex tree structure and new, sophisticated bad events. Furthermore, a better bound on the simulator's quality of randomness must also be established.

7 Conclusion

In this work, we propose a new permutation-based hashing mode that achieves security beyond the birthday bound in the capacity. It also achieves security beyond the birthday bound in the permutation size if $c > 3b/4$.

We proved $2c/3$ bits of security in the powerful indifferentiability framework, while Section 6.2 upper-bounds the achievable security bound with respect to our simulator to $2c/3 + r/3$ bits. As explained in Section 6.2, reaching a tight bound with respect to our simulator is extremely hard. This is because inner collisions of degree more than two are difficult to deal with. It should be noted that the attack of Section 6.2 is for our specific simulator; it may be that there exists a simulator which defeats this attack.[4] However, the challenges involved in describing such simulator and proving its security make it a difficult undertaking. Indeed, in this case the higher-degree inner collisions would not only become intractable from a proof side (as already explained in Section 6.2), but also from the simulator side, as the latter has to keep track of them.

Acknowledgements. We would like to thank the anonymous reviewers for their valuable comments. Charlotte Lefevre is supported by the Netherlands Organisation for Scientific Research (NWO) under grant OCENW.KLEIN.435. Bart Mennink is supported by the Netherlands Organisation for Scientific Research (NWO) under grant VI.Vidi.203.099.

References

ADMV15. Elena Andreeva, Joan Daemen, Bart Mennink, and Gilles Van Assche. Security of Keyed Sponge Constructions Using a Modular Proof Approach. In Gregor Leander, editor, *Fast Software Encryption - 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, volume 9054 of *Lecture Notes in Computer Science*, pages 364–384. Springer, 2015.

AFK$^+$11. Frederik Armknecht, Ewan Fleischmann, Matthias Krause, Jooyoung Lee, Martijn Stam, and John P. Steinberger. The Preimage Security of Double-Block-Length Compression Functions. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 233–251. Springer, 2011.

AMP10. Elena Andreeva, Bart Mennink, and Bart Preneel. On the indifferentiability of the Grøstl hash function. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2010.

AMP12. Elena Andreeva, Bart Mennink, and Bart Preneel. The parazoa family: generalizing the sponge hash functions. *Int. J. Inf. Sec.*, 11(3):149–165, 2012.

BCDM20. Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. Dumbo, Jumbo, and Delirium: Parallel Authenticated Encryption for the Lightweight Circus. *IACR Trans. Symmetric Cryptol.*, 2020(S1):5–30, 2020.

BDPV07. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. Ecrypt Hash Workshop 2007, May 2007.

BDPV08. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

BDPV11a. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

BDPV11b. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, January 2011.

BKL$^+$11. Andrey Bogdanov, Miroslav Knezevic, Gregor Leander, Deniz Toz, Kerem Varici, and Ingrid Verbauwhede. Spongent: A Lightweight Hash Function. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011*

[4] More precisely, we can tweak our simulator $\mathcal{S}$ in a way such that its correctness does not depend on $\neg\mathbf{BNS}$, but on a more specific event.

- *13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 312–325. Springer, 2011.

BMN10. Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Security Analysis of the Mode of JH Hash Function. In Seokhie Hong and Tetsu Iwata, editors, *Fast Software Encryption, 17th International Workshop, FSE 2010, Seoul, Korea, February 7-10, 2010, Revised Selected Papers*, volume 6147 of *Lecture Notes in Computer Science*, pages 168–191. Springer, 2010.

BN18. Srimanta Bhattacharya and Mridul Nandi. A note on the chi-square method: A tool for proving cryptographic security. *Cryptogr. Commun.*, 10(5):935–957, 2018.

BR93. Mihir Bellare and Phillip Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993*, pages 62–73. ACM, 1993.

CDH$^+$12. Donghoon Chang, Morris Dworkin, Seokhie Hong, John Kelsey, and Mridul Nandi. A keyed sponge construction with pseudorandomness in the standard model. NIST SHA–3 Workshop, March 2012.

CDMP05. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer, 2005.

CJN20. Bishwajit Chakraborty, Ashwin Jha, and Mridul Nandi. On the Security of Sponge-type Authenticated Encryption Modes. *IACR Trans. Symmetric Cryptol.*, 2020(2):93–119, 2020.

DHT17. Wei Dai, Viet Tung Hoang, and Stefano Tessaro. Information-Theoretic Indistinguishability via the Chi-Squared Method. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 497–523. Springer, 2017.

DM19. Christoph Dobraunig and Bart Mennink. Leakage Resilience of the Duplex Construction. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 225–255. Springer, 2019.

DMP22. Christoph Dobraunig, Bart Mennink, and Robert Primas. Leakage and Tamper Resilient Permutation-Based Cryptography. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 859–873. ACM, 2022.

DMV17. Joan Daemen, Bart Mennink, and Gilles Van Assche. Full-State Keyed Duplex with Built-In Multi-user Support. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 606–637. Springer, 2017.

FGL09. Ewan Fleischmann, Michael Gorski, and Stefan Lucks. Security of Cyclic Double Block Length Hash Functions. In Matthew Geoffrey Parker, editor, *Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Proceedings*, volume 5921 of *Lecture Notes in Computer Science*, pages 153–175. Springer, 2009.

GIM22. Chun Guo, Tetsu Iwata, and Kazuhiko Minematsu. New indifferentiability security proof of MDPH hash function. *IET Inf. Secur.*, 16(4):262–281, 2022.

GKM$^+$11. Praveen Gauravaram, Lars Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren Thomsen. Grøstl – a SHA-3 candidate, 2011. Submission to NIST's SHA-3 competition.

GPP11. Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.

GPT15. Peter Gaži, Krzysztof Pietrzak, and Stefano Tessaro. The Exact PRF Security of Truncation: Tight Bounds for Keyed Sponges and Truncated CBC. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 368–387. Springer, 2015.

Hir04. Shoichi Hirose. Provably Secure Double-Block-Length Hash Functions in a Black-Box Model. In Choonsik Park and Seongtaek Chee, editors, *Information Security and Cryptology - ICISC 2004, 7th International Conference, Seoul, Korea, December 2-3, 2004, Revised Selected Papers*, volume 3506 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2004.

Hir06. Shoichi Hirose. Some Plausible Constructions of Double-Block-Length Hash Functions. In Matthew J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2006.

IKMP20. Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the Titans: The Romulus and Remus Families of Lightweight AEAD Algorithms. *IACR Trans. Symmetric Cryptol.*, 2020(1):43–120, 2020.

JLM14. Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond $2^{c/2}$ Security in Sponge-Based Authenticated Encryption Modes. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 85–104. Springer, 2014.

JLM$^+$19. Philipp Jovanovic, Atul Luykx, Bart Mennink, Yu Sasaki, and Kan Yasuda. Beyond Conventional Security in Sponge-Based Authenticated Encryption Modes. *J. Cryptol.*, 32(3):895–940, 2019.

JÖS12. Dimitar Jetchev, Onur Özen, and Martijn Stam. Collisions Are Not Incidental: A Compression Function Exploiting Discrete Geometry. In Ronald Cramer, editor, *Theory of Cryptography - 9th Theory of Cryptography Conference, TCC 2012, Taormina, Sicily, Italy, March 19-21, 2012. Proceedings*, volume 7194 of *Lecture Notes in Computer Science*, pages 303–320. Springer, 2012.

Jou04. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

KP97. Lars R. Knudsen and Bart Preneel. Fast and Secure Hashing Based on Codes. In Burton S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*, volume 1294 of *Lecture Notes in Computer Science*, pages 485–498. Springer, 1997.

KRT07. Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2007.

LK11. Jooyoung Lee and Daesung Kwon. The Security of Abreast-DM in the Ideal Cipher Model. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 94-A(1):104–109, 2011.

LM92. Xuejia Lai and James L. Massey. Hash Function Based on Block Ciphers. In Rainer A. Rueppel, editor, *Advances in Cryptology - EUROCRYPT '92, Workshop on the Theory and Application of of Cryptographic Techniques, Balatonfüred, Hungary, May 24-28,* *1992, Proceedings*, volume 658 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 1992.

LM22. Charlotte Lefevre and Bart Mennink. Tight preimage resistance of the sponge construction. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 185–204. Springer, 2022.

LS15. Jooyoung Lee and Martijn Stam. MJH: a faster alternative to MDC-2. *Des. Codes Cryptogr.*, 76(2):179–205, 2015.

LSS11a. Jooyoung Lee, Martijn Stam, and John P. Steinberger. The Collision Security of Tandem-DM in the Ideal Cipher Model. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 561–577. Springer, 2011.

LSS11b. Jooyoung Lee, Martijn Stam, and John P. Steinberger. The preimage security of double-block-length compression functions. *IACR Cryptol. ePrint Arch.*, page 210, 2011.

Men12. Bart Mennink. Optimal Collision Security in Double Block Length Hashing with Single Length Key. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 526–543. Springer, 2012.

Men13. Bart Mennink. Indifferentiability of Double Length Compression Functions. In Martijn Stam, editor, *Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings*, volume 8308 of *Lecture Notes in Computer Science*, pages 232–251. Springer, 2013.

Men14. Bart Mennink. On the collision and preimage security of MDC-4 in the ideal cipher model. *Des. Codes Cryptogr.*, 73(1):121–150, 2014.

Men23. Bart Mennink. Understanding the Duplex and Its Security. *IACR Trans. Symmetric Cryptol.*, 2023(2):1–46, 2023.

MP12. Bart Mennink and Bart Preneel. Hash Functions Based on Three Permutations: A Generic Security Analysis. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 330–347. Springer, 2012.

MPS16. Dustin Moody, Souradyuti Paul, and Daniel Smith-Tone. Improved indifferentiability security bound for the JH mode. *Des. Codes Cryptogr.*, 79(2):237–259, 2016.

MRH04. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.

MRV15. Bart Mennink, Reza Reyhanitabar, and Damian Vizár. Security of Full-State Keyed Sponge and Duplex: Applications to Authenticated Encryption. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 465–489. Springer, 2015.

MS88. Carl H Meyer and Michael Schilling. Secure program load with manipulation detection code. In *Proc. Securicom*, volume 88, pages 111–130, 1988.

Nai17. Yusuke Naito. Indifferentiability of Double-Block-Length Hash Function Without Feed-Forward Operations. In Josef Pieprzyk and Suriadi Suriadi, editors, *Information Security and Privacy - 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3-5, 2017, Proceedings, Part II*, volume 10343 of *Lecture Notes in Computer Science*, pages 38–57. Springer, 2017.

Nai19. Yusuke Naito. Optimally Indifferentiable Double-Block-Length Hashing Without Post-processing and with Support for Longer Key Than Single Block. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology - LATINCRYPT 2019 - 6th International Conference on Cryptology and Information Security in Latin America,* *Santiago de Chile, Chile, October 2-4, 2019, Proceedings*, volume 11774 of *Lecture Notes in Computer Science*, pages 65–85. Springer, 2019.

NIS19. NIST. Lightweight Cryptography, February 2019. https://csrc.nist.gov/Projects/Lightweight-Cryptography.

NO14. Yusuke Naito and Kazuo Ohta. Improved Indifferentiable Security Analysis of PHOTON. In Michel Abdalla and Roberto De Prisco, editors, *Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings*, volume 8642 of *Lecture Notes in Computer Science*, pages 340–357. Springer, 2014.

NSS21. Yusuke Naito, Yu Sasaki, and Takeshi Sugawara. Double-Block-Length Hash Function for Minimum Memory Size. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 376–406. Springer, 2021.

NY16. Yusuke Naito and Kan Yasuda. New Bounds for Keyed Sponges with Extendable Output: Independence Between Capacity and Message Length. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2016.

ÖS09. Onur Özen and Martijn Stam. Another Glance at Double-Length Hashing. In Matthew Geoffrey Parker, editor, *Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009, Cirencester, UK, December 15-17, 2009. Proceedings*, volume 5921 of *Lecture Notes in Computer Science*, pages 176–201. Springer, 2009.

RS08. Phillip Rogaway and John P. Steinberger. Constructing Cryptographic Hash Functions from Fixed-Key Blockciphers. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 433–450. Springer, 2008.

RSS11. Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with Composition: Limitations of the Indifferentiability Framework. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer, 2011.

SS08. Thomas Shrimpton and Martijn Stam. Building a Collision-Resistant Compression Function from Non-compressing Primitives. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 643–654. Springer, 2008.

SSY12. John P. Steinberger, Xiaoming Sun, and Zhe Yang. Stam's Conjecture and Threshold Phenomena in Collision Resistance. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 384–405. Springer, 2012.

Sta08. Martijn Stam. Beyond Uniformity: Better Security/Efficiency Tradeoffs for Compression Functions. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 397–412. Springer, 2008.

Sta09. Martijn Stam. Blockcipher-Based Hashing Revisited. In Orr Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2009.

Ste10. John P. Steinberger. Stam's Collision Resistance Conjecture. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference* *on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 597–615. Springer, 2010.

SY15. Yu Sasaki and Kan Yasuda. How to Incorporate Associated Data in Sponge-Based Authenticated Encryption. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*, volume 9048 of *Lecture Notes in Computer Science*, pages 353–370. Springer, 2015.

Wu11. Hongjun Wu. The Hash Function JH, 2011. Submission to NIST's SHA-3 competition.