IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2018, No. 4, pp. 197–228. DOI:10.13154/tosc.v2018.i4.197-228

Sound Hashing Modes of Arbitrary Functions, Permutations, and Block Ciphers

Joan Daemen[1], Bart Mennink[1] and Gilles Van Assche[2]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
joan@cs.ru.nl, b.mennink@cs.ru.nl
[2] STMicroelectronics, Diegem, Belgium
gilles.vanassche@st.com

Abstract. Cryptographic hashing modes come in many flavors, including Merkle-Damgård with various types of strengthening, Merkle trees, and sponge functions. As underlying primitives, these functions use arbitrary functions, permutations, or block ciphers. In this work we provide three simple proofs, one per primitive type, that cover all modes where the input to the primitive consists of message bits, chaining value bits, and bits that only depend on the mode and message length. Our approach generalizes and simplifies over earlier attempts of Dodis et al. (FSE 2009) and Bertoni et al. (Int. J. Inf. Sec. 2014). We prove tight indifferentiability bounds for modes using each of these three primitive types provided that the mode satisfies some easy to verify conditions.

Keywords: Hash functions · tree hashing · generalization · sufficient conditions · indifferentiability · tight

1 Introduction

Cryptographic hash functions are amongst the most-studied and most-used cryptographic functions. Their first appearance dates back to the 70s, when Rabin introduced his iterative hash function design [Rab78] and Merkle his ideas on tree hashing [Mer79], two ideas that later became the predominant approaches in hash function design. Iterative hashing modes of a fixed-input-length compression functions were further investigated and popularized by Merkle and Damgård [Mer89, Dam89]. Taking a compression function that maps $2n$ bits to n bits, Merkle-Damgård partitions an arbitrary length message into pieces of n bits and compresses these pieces one-by-one into an n-bit state. A Merkle tree [Mer79, Mer92, Mer87] likewise partitions the message into pieces of n bits, but places all messages at the leaves of a tree with in-degree two and evaluates the compression function for every branch. The security analysis of both approaches initially focused on preservation of collision resistance: if the underlying compression function is collision resistant, then the hash function is collision resistant as well.

The design and analysis of cryptographic hashing modes has subsequently conceived notable generalizations along three axes:

- The first axis is the generalization of the mode of use. Merkle-Damgård has been considered with "strengthening" or suffix-free encoding [LM92], with prefix-free [CDMP05] or HAIFA [BD07] encoding, with truncation [CDMP05, Luc05], with intermediate transformations [HPY07], with enveloping [BR06], and so on. Merkle trees have likewise seen various generalizations [BR97, SS01, LCL+03, LCL+05, Sar07], and have found adoption in and popularization through the blockchain methodology behind, for example, Bitcoin [Nak08, GKL15]. Tree hashing is also proposed for

Received: 2018-09-01, Accepted: 2018-11-01, Published: 2018-12-13

efficiency purposes. E.g., ParallelHash, defined in NIST SP800-185 [KCP16] is a parallelizable hash function using SHAKE128 or SHAKE256 as underlying compression function.

- The second axis is the diversification of the underlying primitives. Earlier constructions assumed a collision resistant fixed-input-length compression function, but such functions are not readily available and need to be instantiated with something that actually can be built. Rabin [Rab78] already suggested to use a block cipher as compression function, $F(h, m) = \mathrm{DES}_m(h)$, but it was not collision resistant and hence collision resistance preservation was void. This gave rise to the Davies-Meyer construction, basically the addition of a feedforward to remove invertibility, later generalized by Preneel, Govaerts, and Vandewalle in their seminal PGV classification [PGV93, BRS02, Sta09, BRSS10]. Merkle on the other hand constructed a compression function from a truncated permutation in his hash function SNEFRU [Mer90]. The work was followed up by theoretical approaches (most prominently, Black et al. [BCS05]), but also practical constructions were proposed, such as the sponge construction [BDPV07] that underlies SHA-3 winner Keccak.

- The third axis is the type of security analysis. Initial security analyses focused on preservation of collision resistance, and later also of preimage resistance. Rogaway and Shrimpton [RS04] gave a general treatment of security properties of hash functions. Maurer et al. [MRH04] introduced the "indifferentiability framework" that was applied to hash functions by Coron et al. [CDMP05]. It allows to measure the likeness of a hash function to a random oracle assuming a random underlying primitive. A hash function secure in the indifferentiability framework "behaves like" a random oracle, and can replace it in almost all single-stage settings (see Ristenpart et al. [RSS11]). It implies resistance against collision and (second) preimage attacks, among others. As customary, we call a hashing mode "indifferentiable" if there is a proof that the success probability of differentiating the mode from a random function, usually a random oracle, is very close to the birthday bound in the length of the chaining value (CV).

Indifferentiability is a powerful security notion for a hashing mode, but proofs are often complex and error-prone. In addition, in light of the plenitude of hashing modes (axis 1) and generic instantiations (axis 2), every hashing mode comes with its own dedicated indifferentiability proof. For example, rewinding to the SHA-3 competition, indifferentiability of the modes underlying the five finalists BLAKE, Grøstl, JH, Keccak, and Skein was proven in [ALM12, CNY11], [AMP10], [BMN10, MPS16], [BDPV08], and [BKL+09], respectively (see also [AMPŠ12]). The indifferentiability framework does support composability, so an approach to limit the proliferation of dedicated proofs would be to construct an indifferentiable compression function from an underlying primitive such as a block cipher or permutation, and then just apply an indifferentiable mode to that compression function. There are however two problems with this approach. First, most of the proposed compression function constructions, including PGV, simply cannot be proven indifferentiable [KM07]. Second, this two-level approach induces a cost that can be avoided in a dedicated proof. For example, Dodis et al. [DRRS09] proved an indifferentiable bound of a b-to-n-bit compression function built from a b-bit permutation by truncating its output to n bits, but it requires fixing q bits of its input. For achieving indifferentiability, b must be at least $2n$ and q at least $n/2$. Hence, this approach wastes $n/2$ bits of every compression function call.

1.1 Sound Hashing

This systemization of knowledge article is inspired by Dodis et al. [DRRS09], Bertoni et al. [BDPV14b], and Daemen et al. [DDV11]. Dodis et al. [DRRS09] derived five conditions required for a generalized version of the MD6 hashing mode to be secure. Bertoni et al. [BDPV14b] derived three quite general sufficient conditions for sound hashing and proved a tight indifferentiability bound for hashing modes taking a compression function modeled as an arbitrary function (a function that behaves as a random oracle, possibly restricted to having fixed input and/or output length). Daemen et al. [DDV11] demonstrated that a condition must be added if the compression function is a truncated permutation. We revisit, re-factor, and generalize the sufficient conditions of Bertoni et al., in such a way that they are a bit cleaner and slightly extend the more hashing modes. Moreover, by the application of the H-coefficient technique we prove the same bound with a significantly shorter proof. We extend this result in a natural way to hashing modes of truncated permutations and block ciphers (possibly truncated, too), without the Davies-Meyer feedforward. Using the additional condition of Daemen et al. [DDV11], we prove indifferentiability with simple proofs that are incremental to the one for modes of an arbitrary function.

Our sufficient conditions cover all hashing modes where the compression function inputs are concatenations of chaining values, message bits, and *frame* bits. The latter are bits whose values are independent of the message content, such as padding bits and bits encoding the presence of message blocks or chaining values. The inputs to the compression function in a hash function evaluation can be arranged in a so-called *hash tree* where the children of a node are the compression function inputs that lead to the chaining values in that node, and the root of the hash tree is the compression function input that maps to the final hash result. The three conditions that are sufficient for indifferentiability of modes of an arbitrary function are *subtree-freeness*, *radical-decodability*, and *message-decodability*. Subtree-freeness prevents generalizations of length extension attacks by requiring that a hash tree cannot be a subtree of another hash tree. Radical-decodability rules out the existence of hash function collisions in the absence of compression function collisions. Message-decodability ensures that different messages cannot give rise to the same hash tree by requiring that the message can be fully recovered from the compression function inputs. For modes of a truncated permutation or block cipher, the additional condition is *leaf-anchoring*. Leaf-anchoring prevents collision attacks exploiting inverse permutation or block cipher calls by reserving part of the permutation input or block cipher data input to a chaining value for non-leaf nodes and to an initial value for leaf nodes.

The indifferentiability bounds that we derive for hashing modes satisfying these three, resp. four, conditions are given in Table 1. We define the conditions and compare those with the ones of [DRRS09,BDPV14b,DDV11] in Section 3, state the formal security results in Section 4, and provide the corresponding proofs in Sections 5-7.

1.2 Application

We map our sufficient conditions to practice in Section 8. First, on the constructive side, we describe minimalistic sequential and tree hashing modes that satisfy our criteria in Section 8.1. By economical use of frame bits, these solutions are more efficient than what is usually proposed. For example, as we explain later on in Section 8.5, our minimalistic example processes more data per primitive evaluation than MD6, and is thus more efficient. In Section 8.2, we discuss an observation on the role of the IV, namely the possibility to relax subtree-freeness in the presence of leaf-anchoring. This observation is subsequently used in the projection of our sufficient conditions to suffix-free Merkle-Damgård and Enveloped Merkle-Damgård in Sections 8.3 and 8.4. Suffix-free Merkle-Damgård turns out *not* to satisfy subtree-freeness and our indifferentiability results do not apply. This should not come as a surprise—it re-confirms a result by Coron et al. [CDMP05]—but it *does* indicate that

Table 1: Indifferentiability bounds for hashing modes of an arbitrary function, a truncated permutation or a (truncated) block cipher. The conditions SF, RD, MD, and LA stand for subtree-freeness, radical-decodability, message-decodability, and leaf-anchoring, respectively. q is the adversarial complexity expressed as the number of primitive queries either direct or indirect, n the CV length, and $b \geq n$ the width of the permutation resp. the block length of the block cipher.

compression function type	SF+RD+MD	LA	bound	reference
arbitrary function	✓	—	$\frac{\binom{q}{2}}{2^n}$	Theorem 1
truncated permutation	✓	✓	$\frac{\binom{q}{2}+1}{2^n} + \frac{\binom{q}{2}}{2^b}$	Theorem 2
truncated block cipher	✓	✓	$\frac{\binom{q}{2}+1}{2^n} + \frac{\binom{q}{2}}{2^b}$	Theorem 3
block cipher	✓	✓	$\frac{2\binom{q}{2}+1}{2^n}$	Theorem 3

subtree-freeness cannot be loosened much. Enveloped Merkle-Damgård [BR06] satisfies the conditions, and similar conclusions can be drawn for HAIFA [BD07, BMN09], various types of Merkle trees [Mer79, Mer87], and ParallelHash as defined in NIST SP800-185 [KCP16], among others.

In Section 8.5 we discuss various tree hashing modes in the wild, in Section 8.6 we give an overview of the Sakura encoding [BDPV14a] relative to our modes and conditions, and in Section 8.7 we consider an application of our conditions to build a message authentication code (MAC) function that for short messages is a factor 4 faster than HMAC [KBC97, Bel06].

Interestingly, our results show that a cryptographic hashing mode of a block cipher *does not need* a feedforward for satisfying indifferentiability as long as the mode itself satisfies our sufficient conditions (see also Section 8.1). Moreover, it covers truncation with $n \leq b$, i.e., taking chaining values shorter than the block length of the block cipher. This allows to have a sound hashing mode without the burden of including frame bits into the key input of the block cipher (see also Section 8.7). On the other hand, a feedforward turns a block cipher into a non-invertible compression function and hence may allow to relax the leaf-anchoring condition. This yields a tradeoff between spending n input bits on an IV to ensure leaf-anchoring and using extra state due to the feedforward. Notably, most standard hash functions, including MD5 [Riv92], SHA-1, and SHA-2 [SHA08]—but not SHA-3 [FIP15]—have both an initial value and a feedforward.

In this paper, we limit ourselves to modes that build the inputs to the underlying primitive by concatenation of chaining values, message bits, and frame bits. This rules out two popular constructions. The first one is the Davies-Meyer construction, where the data input to the underlying block cipher is added to the block cipher output to form the chaining value. The second one is the sponge construction that builds the input to the underlying permutation as the bitwise sum of a permutation output and a message block [BDPV07]. Indifferentiability of the sponge was proven by Bertoni et al. [BDPV08], and a generalized treatment in the indifferentiability framework was presented by Andreeva et al. [AMP12]. Canteaut et al. [CFN+12] considered indifferentiability of sequential hashing based on the broadened Stam block cipher based compression function [Sta09].

2 Parameterized Hashing Mode

Following Bertoni et al. [BDPV14b], we consider parameterized hashing modes. These take as input a message M of arbitrary length and an assignment of parameter values A in some space $\mathcal{A}$ that can be seen as instructions on how to perform the hashing. For example, in a tree hashing mode, A may include chunk length and tree depth. According to the parameter values A, the message bits are distributed over a number of strings and presented to the compression function $\mathcal{F}$. The corresponding outputs of $\mathcal{F}$ are called *chaining values* (CV) and are possibly combined with message bits, subject to yet other calls to $\mathcal{F}$. Finally, the hash result is the output of the application of $\mathcal{F}$ to a string that depends on all the bits of the message, either directly or via chaining values. Note that this definition also covers hashing modes that take no parameters, by just having for $\mathcal{A}$ the empty set.

We define the computation of a hash according to some mode $\mathcal{T}$ of a compression function $\mathcal{F}$ as a two-stage process. For a set of parameters $\mathcal{A}$, the hash function $\mathcal{T}[\mathcal{F}] : \mathbf{Z}_2^* \times \mathcal{A} \rightarrow \mathbf{Z}_2^n$ is defined as

$$\mathcal{T}[\mathcal{F}](M, A) = h, \text{ where } Z = \mathcal{Z}(|M|, A)$$
$$h = \mathcal{Y}[\mathcal{F}](M, Z)\,,$$

where $\mathcal{Z}$ is the *template construction* function (formalized in Section 2.1) and $\mathcal{Y}$ is the *template execution function* (formalized in Section 2.2). So $\mathcal{T}$ consists of a sequential evaluation of:

- A template construction function $\mathcal{Z}$ that is a deterministic algorithm that makes no evaluations of the compression function $\mathcal{F}$ and is specific for a given mode $\mathcal{T}$.
- A template execution function $\mathcal{Y}$ that evaluates the compression function $\mathcal{F}$ and is generic, i.e., the same for all possible modes $\mathcal{T}$.

In case $\mathcal{F}$ is an arbitrary function, $\mathcal{T}[\mathcal{F}]$ may get as additional input $\ell \in \mathbb{N}$ indicating the requested length of the response, a parameter relayed to $\mathcal{Y}[\mathcal{F}]$. There is no a priori limitation on the number of evaluations of $\mathcal{F}$ made by $\mathcal{T}[\mathcal{F}]$.

We adopted the split of hash function processing in template construction and execution from Bertoni et al. [BDPV14b]: it allows us to reason about mode-level processing that is independent from the compression function and the content of the input. This split is conceptual: it is essential for the definition of our conditions (see also the comparison of our conditions with those of Dodis et al. [DRRS09] in Section 3.5), but does not exist in actual implementations.

2.1 Template Construction

The template construction $\mathcal{Z}$ takes as input the length $|M|$ of the message and parameter assignment $A \in \mathcal{A}$, and provides a tree of virtual strings, or *template nodes*, for hashing the message: a *tree template* Z. We write $Z = \mathcal{Z}(|M|, A)$. The nodes in this tree are recipes that specify how to transform the message chunks and chaining values into bit strings to be processed by $\mathcal{F}$. These strings contain three types of virtual bits:

- frame bits: bits with values fully determined by $|M|$ and A. These include padding, IV blocks, and other bits that make the mode compliant to the conditions we will define later on.
- message pointer bits: each such bit specifies a bit of the message M. During template execution a message pointer bit is replaced by the bit in the specified position in the message.

- chaining pointer bits: each such bit specifies a CV bit resulting from earlier evaluations of $\mathcal{F}$. Note that the tree will be evaluated in a hierarchic manner where the output of $\mathcal{F}$ applied to a node is included in a node at a higher level.

A chaining pointer bit has two attributes: the CV index and the bit offset. Chaining pointer bits with the same CV index point to the same CV and in a mode with n-bit CVs there shall be n of them. We will only consider tree hashing modes and their special case of sequential hashing modes. In tree hashing the n bits of a CV are in the same node, and there is exactly one bit for each of the n indices. An example tree template is given in Figure 1a. Further examples of tree templates will be given in Section 8, where we will also link them with the sufficient conditions of Section 3 and the security results of Section 4.

Node x is a child of node y (and y is the parent of x) if y contains chaining pointer bits pointing to the CV of node x. Node x is a descendant of node y if the parent of x is either y or a descendant of y. In a tree template, every node may have a unique parent node, and an arbitrary amount of children nodes. A leaf node is a node with no children (i.e., a node that does not contain chaining pointer bits) and the final node is the unique node that has no parent.

2.2 Template Execution

The template interpreter $\mathcal{Y}$ takes as input a message M and a tree template Z, and executes it using $\mathcal{F}$ to obtain the output h. Starting from the leaves, it instantiates all nodes to obtain the corresponding input string to $\mathcal{F}$ and subsequently evaluates $\mathcal{F}$. In this process, it replaces message pointer bits by the bits at the specified offset in the message and chaining pointer bits by the appropriate CV bits. The template execution renders a *tree* S: $S = \mathcal{Y}[\mathcal{F}](M, Z)$. The final step in the template execution is to evaluate the final node final(S) using $\mathcal{F}$: $h = \mathcal{F}(\text{final}(S))$. In case $\mathcal{T}[\mathcal{F}]$ supports variable output length, h is truncated to the requested length. An example tree is given in Figure 1b.

We represent a tree S (or a subtree thereof) as a list of couples (x, α), each representing a node. The first member of a node x is the binary string as built from the template execution and the second member α identifies the node and the bit positions within that node where the CV $\mathcal{F}(x)$ can be found. Clearly, each α points to a node earlier in the list. The CV of the final node of S does not figure in S, and we indicate this with having $\perp$ as second member. In this representation, attaching a node to a tree S comes down to adding a couple (x', α') to the list, where α' points to a node of S. The list of couples corresponding to the example tree of Figure 1b is given in Figure 1c.

2.3 Definitions of Sets of Hashing Trees

The split in template construction and template execution allows us to define a number of sets of hashing trees that are useful to specify and reason about sufficient conditions. We start by defining the set of tree templates Z and trees S that can potentially arise from a given mode $\mathcal{T}$.

Definition 1 (tree template set [BDPV14b])**.** For a mode of operation $\mathcal{T}$, we define the *tree template set* $\mathcal{Z}_\mathcal{T}$ as the set of all tree templates that can be generated by $\mathcal{T}$:

$$\mathcal{Z}_\mathcal{T} = \{Z \mid \exists(\mu, A) \in \mathbb{N} \times \mathcal{A} \text{ such that } Z = \mathcal{Z}(\mu, A)\}\,.$$

We subsequently define tree and subtree sets.

Definition 2 ((sub)tree set [BDPV14b])**.** A tree S *complies with* a template Z if it has the same tree topology, the corresponding nodes have the same length, and the values of the frame bits in Z match those in S.

For a mode of operation $\mathcal{T}$, we define the following sets:

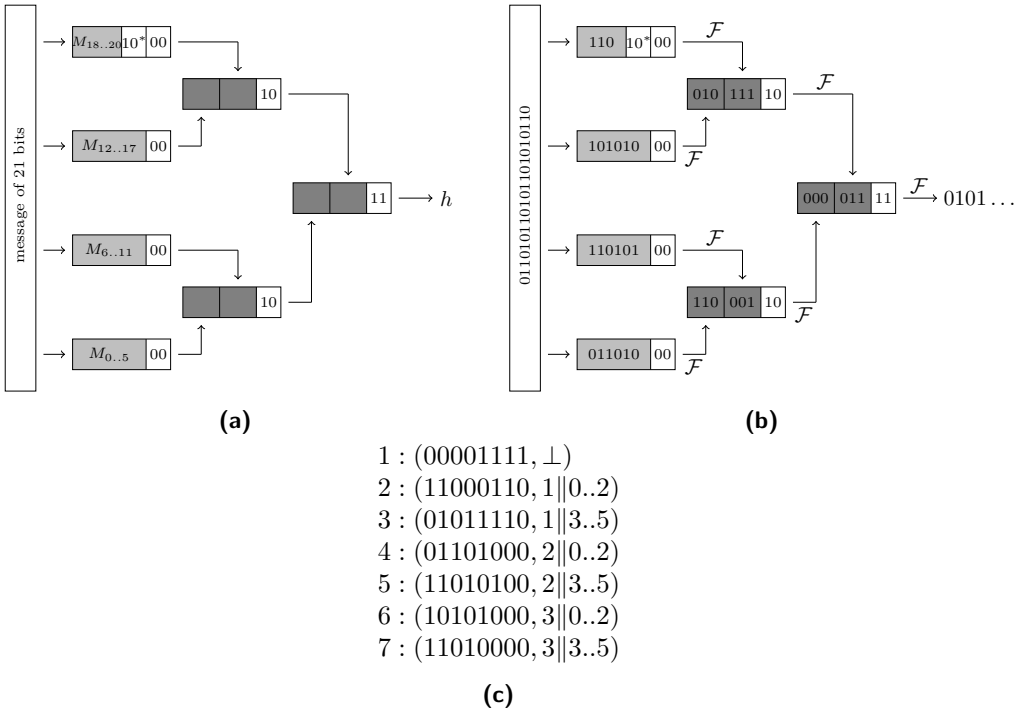

Figure 1: (a) Schematic example of a tree template Z, (b) its corresponding instantiation S for a specific input message M, and (c) its corresponding list of couples. For figures (a) and (b): white blocks contain frame bits, light gray blocks contain message pointer bits, and dark gray chaining blocks contain chaining pointer bits. For figure (c): "$(11000110, 1\|0..2)$" means that the evaluation of $\mathcal{F}$ on 11000110 is represented by bits 0..2 of line 1.

- $\mathcal{S}_\mathcal{T}$ is the set of all trees that comply with a template in $\mathcal{Z}_\mathcal{T}$:

$$\mathcal{S}_\mathcal{T} = \bigcup_{Z \in \mathcal{Z}_\mathcal{T}} \{S \mid S \text{ complies with } Z\}\,.$$

- $\mathcal{S}_\mathcal{T}^{\text{sub}}$ is the set of all proper subtrees of trees in $\mathcal{S}_\mathcal{T}$:

$$\mathcal{S}_\mathcal{T}^{\text{sub}} = \bigcup_{S \in \mathcal{S}_\mathcal{T}} \{S' \mid S' \text{ is proper subtree of } S\}\,.$$

  These are trees that can be constructed as follows. Take a tree S in $\mathcal{S}_\mathcal{T}$ and select one if its nodes. That will be the root of the subtree S'. Then for each of the child nodes of this node, include it or not. Do this recursively for all child nodes of the included nodes. There must be at least one node in S that is not in S'.

- $\mathcal{S}_\mathcal{T}^{\text{leaf}}$ is the subset of $\mathcal{S}_\mathcal{T}^{\text{sub}}$ of trees S' that have the following property: $\exists S \in \mathcal{S}_\mathcal{T}$ with S' a proper subtree of S, and containing a node in S and all its descendants.

- $\mathcal{S}_\mathcal{T}^{\text{final}}$ is the subset of $\mathcal{S}_\mathcal{T}^{\text{sub}}$ of trees S' that have the following property: $\exists S \in \mathcal{S}_\mathcal{T}$ with S' a proper subtree of S, and containing the root of S.

An illustration of the (sub)tree instance sets is given in Figure 2.

A concept that is central to our reasoning is that of a *radical*, and we will illustrate its idea through an example. Consider a tree $S' \in \mathcal{S}_\mathcal{T}^{\text{sub}}$ obtained by taking a tree $S \in \mathcal{S}_\mathcal{T}$ and removing a leaf node. The CV that is the hash of that leaf node is still present in S' but

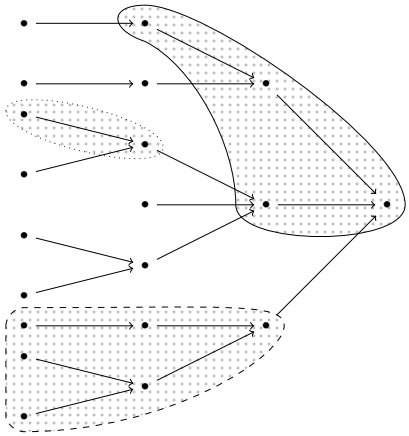

Figure 2: Example of a tree $S \in \mathcal{S}_\mathcal{T}$, along with an identification of three subtrees. The subtree with a dashed line is a leaf subtree, and the one with a solid line is a final subtree. The subtree with a dotted line is neither leaf nor final.

has become a mere stub. If this is the case for all trees $S \in \mathcal{S}_\mathcal{T}$ that S' is a subtree of, we call this a *radical CV*. A *radical* is the position in S' of such a radical CV. We now give a more formal definition.

Definition 3 (radical)**.** A *radical* α in a tree instance $S' \in \mathcal{S}_\mathcal{T}^{\text{sub}}$ identifies a node and a set of bit positions in that node such that, for any tree $S \in \mathcal{S}_\mathcal{T}$, of which S' is a subtree of S, the bits identified by α form a CV that has a node attached to it in S but not in S'. A *radical CV* is the value located at a radical α and is denoted $S'[\alpha]$.

3 Sufficient Conditions

We formulate three conditions for a tree hashing mode $\mathcal{T}$ to satisfy in order to be indifferentiable: *subtree-freeness*, *radical-decodability*, and *message-decodability* (in Sections 3.1-3.3, respectively). In Section 3.4, we formulate a fourth condition, *leaf-anchoring*, that is relevant if the compression function is a truncated permutation or block cipher. It prevents the adversary to perform length extension attacks at the leaves using inverse permutation of block cipher queries.

The first three conditions are very similar to the three sufficient conditions for sound tree hashing introduced by Bertoni et al. [BDPV14b] in 2014: *tree-decodability*, *message-completeness*, and *final-node separability*. The fourth condition, leaf-anchoring, is taken from Daemen et al. [DDV11]. We will discuss the relation between our conditions and those of [BDPV14b, DDV11] along the way. A summary of the comparison of the conditions is given in Section 3.5, where we will also relate them to the five conditions of Dodis et al. [DRRS09].

3.1 Subtree-Freeness

This property ensures the absence of trees S that are both in $\mathcal{S}_\mathcal{T}$ and in $\mathcal{S}_\mathcal{T}^{\text{sub}}$. If violated, an adversary may use a tree S in $\mathcal{S}_\mathcal{T} \cap \mathcal{S}_\mathcal{T}^{\text{sub}}$ to mount a length extension attack, by building a tree that has S as a leaf subtree and/or by attaching one or more trees to leaf node(s) of S.

Definition 4 (subtree-free)**.** A mode of operation $\mathcal{T}$ is *subtree-free* if

$$\mathcal{S}_\mathcal{T} \cap \mathcal{S}_\mathcal{T}^{\text{sub}} = \emptyset\,. \tag{1}$$

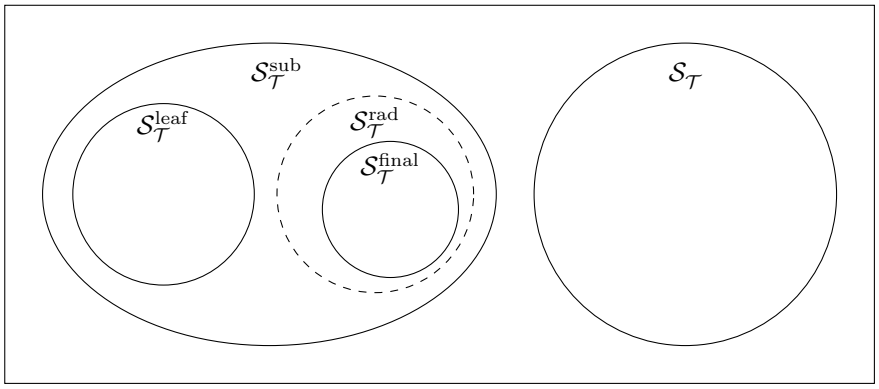

Figure 3: Venn diagram of the sets $\mathcal{S}_\mathcal{T}^{\text{leaf}}$, $\mathcal{S}_\mathcal{T}^{\text{final}}$, $\mathcal{S}_\mathcal{T}^{\text{rad}}$, $\mathcal{S}_\mathcal{T}^{\text{sub}}$, and $\mathcal{S}_\mathcal{T}$.

This condition was not explicitly present in Bertoni et al. [BDPV14b], but it is implied by those: their tree-decodability explicitly states $\mathcal{S}_\mathcal{T} \cap \mathcal{S}_\mathcal{T}^{\text{final}} = \emptyset$, and their final-node separability implies $\mathcal{S}_\mathcal{T} \cap (\mathcal{S}_\mathcal{T}^{\text{sub}} \setminus \mathcal{S}_\mathcal{T}^{\text{final}}) = \emptyset$.

Subtree-freeness covers all possible length extension attacks but is not strictly necessary: as we will explain in Section 8.2, in the presence of leaf-anchoring (of Section 3.4) one can relax subtree-freeness slightly, as leaf-anchoring assures that leaves can be identified and no length extension attack can be performed by prepending data.

3.2 Radical-Decodability

This property ensures that any tree $S \in \mathcal{S}_\mathcal{T}^{\text{final}}$ contains a radical and that this radical can be found efficiently.

Definition 5 (radical-decodability)**.** A mode of operation $\mathcal{T}$ is *radical-decodable* if there exists a set $\mathcal{S}_\mathcal{T}^{\text{rad}}$ such that all trees $S \in \mathcal{S}_\mathcal{T}^{\text{rad}}$ have a radical, and there exists an efficient deterministic function radical() that returns a radical upon presentation of an $S \in \mathcal{S}_\mathcal{T}^{\text{rad}}$, and $\perp$ otherwise. The set $\mathcal{S}_\mathcal{T}^{\text{rad}}$ must satisfy $\mathcal{S}_\mathcal{T}^{\text{final}} \subseteq \mathcal{S}_\mathcal{T}^{\text{rad}} \subseteq \mathcal{S}_\mathcal{T}^{\text{sub}} \setminus \mathcal{S}_\mathcal{T}^{\text{leaf}}$.

Definition 5 deals with four sets: $\mathcal{S}_\mathcal{T}^{\text{leaf}}$, $\mathcal{S}_\mathcal{T}^{\text{final}}$, $\mathcal{S}_\mathcal{T}^{\text{rad}}$, and $\mathcal{S}_\mathcal{T}^{\text{sub}}$, where $\mathcal{S}_\mathcal{T}^{\text{leaf}} \cap \mathcal{S}_\mathcal{T}^{\text{rad}} = \emptyset$ and $\mathcal{S}_\mathcal{T}^{\text{leaf}} \cup \mathcal{S}_\mathcal{T}^{\text{rad}} \subseteq \mathcal{S}_\mathcal{T}^{\text{sub}}$. See also the Venn diagram in Figure 3. By requiring that a radical can be found in any final subtree, we rule out the existence of multiple trees in $\mathcal{S}_\mathcal{T}$ with the same final node, except if they include a collision in the compression function. It is easy to see why this is the case. Assume that S and S' are two trees with the same final node, and let S'' be the largest tree that is a subtree of both. Due to radical-decodability, S'' has a radical, and due to the assumption that S'' is the largest common subtree, the nodes in S and S' that hash to the corresponding radical value must be different and thus represent a compression function collision.

The conditions radical-decodability and subtree-freeness overlap, but one does not imply the other. For seeing that subtree-freeness does not imply radical-decodability, consider a simple hashing mode that *only* takes three message blocks and *only* allows for the two templates as given in Figure 4. It is easy to verify that the mode of operation is subtree-free but not radical-decodable. Namely, the common final subtree indicated in Figure 4 has no radical, namely *a set of bits that corresponds to a CV in all trees in* $\mathcal{S}_\mathcal{T}$ *it is a subtree of*. Note that the hashing mode permits collision attacks, as an adversary can choose message blocks in one mode that match the chaining values in the other mode. Also, radical-decodability does not imply subtree-freeness: a radical-decodable mode where the intersection of $\mathcal{S}_\mathcal{T}$ and $\mathcal{S}_\mathcal{T}^{\text{leaf}}$ is not empty would clearly not be subtree-free. An example of such a mode is a sequential mode like Merkle-Damgård with a frame bit at the end of each node indicating whether it is a leaf or not.

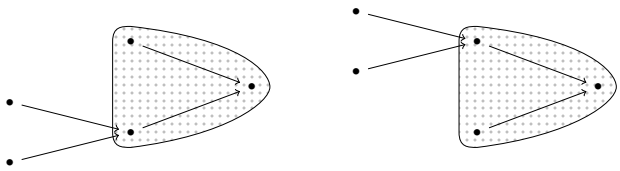

Figure 4: Example of a hashing mode that is subtree-free but not radical-decodable. The hashing mode only allows for the two depicted templates with three message blocks. The subtree with a solid line is a common final subtree.

When building a mode that is radical-decodable, the designer has some freedom in choosing $\mathcal{S}^{\text{rad}}_{\mathcal{T}}$. One extreme is choosing $\mathcal{S}^{\text{rad}}_{\mathcal{T}} = \mathcal{S}^{\text{final}}_{\mathcal{T}}$. In that case, the function radical() must return a radical for any single-node tree consisting of a final node and return $\perp$ for any single-node tree consisting of a non-final node. This can only be achieved by domain separation between final and non-final nodes, hence the condition of final-node separability by Bertoni et al. [BDPV14b]. Indeed, their tree-decodability maps to the combination of our subtree-freeness and radical-decodability with $\mathcal{S}^{\text{rad}}_{\mathcal{T}} = \mathcal{S}^{\text{final}}_{\mathcal{T}}$. At the other end of the spectrum, one may choose $\mathcal{S}^{\text{rad}}_{\mathcal{T}} = \mathcal{S}^{\text{sub}}_{\mathcal{T}} \setminus \mathcal{S}^{\text{leaf}}_{\mathcal{T}}$. This option has the advantage that domain separation between final and non-final nodes is not required, but has the disadvantage that there must be domain separation between leaf and non-leaf nodes and that any single non-leaf node that can occur in any valid tree in $\mathcal{S}_{\mathcal{T}}$ must have a radical that is easy to identify.

3.3 Message-Decodability

This property states that from a tree S that is the result of the hashing process of an input (M, A), the message M can be recovered. We formalize this condition by *message-decodability*.

Definition 6 (message-decodability)**.** A mode of operation $\mathcal{T}$ is *message-decodable* if there is an efficient function extract() that on input of $S \in \mathcal{S}_{\mathcal{T}}$ returns the template Z it complies with and the message M, and on input of $S \notin \mathcal{S}_{\mathcal{T}}$ returns $\perp$.

Clearly, by Definition 2, for any $S \in \mathcal{S}_{\mathcal{T}}$ there is at least one message that could have resulted in S. If message-decodability is not satisfied, there must be two distinct messages M, M' resulting in the same tree instance S. This, in particular, happens if the mode of operation $\mathcal{T}$ does not process all bits of its input message.

3.4 Leaf-Anchoring

This condition states that all leaves have an IV at some fixed position and non-leaf nodes have a CV at that position.

Definition 7 (leaf-anchoring)**.** A mode of operation $\mathcal{T}$ is *leaf-anchored* if for every template $Z \in \mathcal{Z}_{\mathcal{T}}$, the first n bits of every leaf node encode IV as frame bits and the first n bits of every non-leaf node are chaining pointer bits.

Leaf-anchoring is necessary if the compression function allows efficiently computing collisions or pre-images of a CV. That is the case for a truncated permutation or block cipher as this can easily be done by making inverse queries to the primitive. The former trivially allows finding message pairs leading to equal hash results while the latter allows extending a tree at its leaves. By fixing part of the permutation input or the data part input of the block cipher to an IV, inverse queries are only harmful if they hit the IV, and this is hard. Leaf-anchoring is not a necessary condition in the strict sense of the word,

as alternative strategies (like requiring that IV is encoded at a different position or in a different manner) work as well. We consider the current definition of leaf-anchoring to be the simplest and most intuitive choice.

3.5 Detailed Comparison with Earlier Conditions

As we have explained in the previous sections, our conditions of subtree-freeness and radical-decodability correspond, in the special case $\mathcal{S}_{\mathcal{T}}^{\text{rad}} = \mathcal{S}_{\mathcal{T}}^{\text{final}}$, to the earlier tree-decodability and final-node separability from Bertoni et al. [BDPV14b]. The conditions are more general and cleaner. Message-decodability is identical to their message-completeness (but we have re-named it to better reflect what it stands for), and leaf-anchoring is identical to Daemen et al. [DDV11]'s leaf-node anchoring. These are summarized in Table 2.

Dodis et al. [DRRS09] listed five properties required for sound hashing modes of an arbitrary function that has fixed input and output length and do not take parameters. Bertoni et al. compared these with their own conditions and we will here consider how the conditions compare to ours, and summarize the relations in Table 2.

First, the property of *unique parsing* is similar to, but more specific than, Bertoni et al.'s tree-decodability. In detail, unique parsing requires that it is possible to identify frame bits, message pointer bits, and chaining pointers bits with just access to the node instance, making it de facto a restricted case of tree-decodability.

Second, the property of *root predicate* is identical to Bertoni et al.'s final-node separability. As such, the conditions of unique parsing plus root predicate of Dodis et al. are comparable to our definitions of subtree-freeness plus radical-decodability.

Third, the property of *message reconstruction* is equivalent to our condition of message-decodability.

Fourth, *final output processing* says that the output of applying the inner hash function to the final node is transformed using an "efficiently computable, regular function" ζ for which for each h "the set of all preimages $\zeta^{-1}(h)$ must be efficiently sampleable." It seems that this function is introduced as generalization of truncation, in order to cover the case that the outer hash function has a different output length than the inner hash function. In our description, the compression function $\mathcal{F}$ may have arbitrary output length, and the need for such a condition does not appear.

Fifth, Dodis et al. pose the condition of *straight-line program structure*, roughly meaning that the mode of operation can be properly evaluated by a sequential evaluation of the underlying primitive. There is no equivalence of the notion in the conditions of Dodis et al. and ours. Rather, it corresponds to our definition of a tree hashing mode in Section 2: the slightly more elaborate definition of tree hashing (compared to Dodis et al.) is just a matter of presentation. We distinguish two parts in the input to the mode of operation: a message M that only impacts the tree template through its length, and parameter values A that together with $|M|$ determines the tree template. The generalized treatment allows for more flexibility and covers a larger amount of modes of operation.

4 Security of Hashing Modes

In this section we formulate indifferentiability bounds for hashing modes for three types of compression function: an arbitrary function in Section 4.2, a truncated permutation in Section 4.3, and a block cipher in Section 4.4. We start by discussing the security model in Section 4.1.

Table 2: Sufficient conditions from Dodis et al. [DRRS09], Bertoni et al. [BDPV14b], Daemen et al. [DDV11], and current work. The order of the conditions in the earlier works is reshuffled for sake of comparison.

[DRRS09]		[BDPV14b, DDV11]		current work
unique parsing	$\Longrightarrow$	tree-decodability	$\Longrightarrow$	subtree-freeness
root predicate	$\Longleftrightarrow$	final-node separability	$\Longrightarrow$	radical-decodability
message reconstruction	$\Longleftrightarrow$	message-completeness	$\Longleftrightarrow$	message-decodability
—		leaf-node anchoring*	$\Longleftrightarrow$	leaf-anchoring*
final output processing**		—		—
straight-line program structure**		—		—

* only needed if the compression function is a truncated permutation or block cipher. The work of [DRRS09] focuses on hashing modes of arbitrary functions (with fixed input and output length) and the issue did not apply.
** the condition does not appear in [BDPV14b, DDV11] and ours, but is implicit in the description of the hashing mode.

4.1 Security Model

Preliminaries. In our setup we assume *random* primitives. For the case of an arbitrary function, it corresponds to the original definition of random oracle by Bellare and Rogaway [BR93]. For the permutation case, we assume the random permutation $\mathcal{P} : \mathbf{Z}_2^b \to \mathbf{Z}_2^b$ with width b to be selected uniformly from the set of all b-bit permutations $\mathsf{perm}(b)$. For the block cipher case, we assume the random block cipher $\mathcal{E} : \mathbf{Z}_2^\kappa \times \mathbf{Z}_2^b \to \mathbf{Z}_2^b$ with key length κ and block length b to be selected uniformly from the set of all block ciphers of those dimensions, $\mathsf{bc}(\kappa, b)$. For a bit string x of size at least n bits, the function $\lfloor x \rfloor_n$ outputs the n first bits of x. The uniform random drawing of an element x from a finite set X is denoted $x \xleftarrow{\$} X$.

Indifferentiability Framework. Maurer et al. [MRH04] introduced the indifferentiability framework to capture the security of a function whose underlying primitives are publicly available as a random function, and Coron et al. [CDMP05] adapted it to hash functions. Indifferentiability measures the distance between a mode of operation $\mathcal{T}$ of a random component $\mathcal{R}$ and a random oracle $\mathcal{RO}$. In our case, $\mathcal{R}$ is either a function $\mathcal{F}$, a permutation $\mathcal{P}$, or a block cipher $\mathcal{E}$.

The mode of operation $\mathcal{T}$ takes a message M and a parameter assignment A, and returns independent responses for different messages, but for the same message hashed under different parameters the responses may be equal. The reason for this is that the parameters tell how to shape the tree, but not all parameters are significant in all cases: e.g., the maximum height of the tree may not be reached for short messages. So the parameters should not be seen as additional input but rather instructions on how to perform the hashing process, and we tolerate the hash function in the simulated world to return equal responses for different input pairs (M, A) and (M, A'). Recalling that $\mathcal{T}$ consists of a sequential evaluation of a deterministic template construction function $\mathcal{Z}$ (that makes no evaluations of the compression function) and a template execution function $\mathcal{Y}$ (that does evaluate the compression function $\mathcal{F}$), we formalize this by "encapsulating" the deterministic function $\mathcal{Z}$ into $\mathcal{D}$. Rather than making arbitrary construction queries of the form (M, A) to $\mathcal{T}$, the distinguisher makes construction queries of the form (M, Z) to $\mathcal{Y}$, with the restriction that any such query satisfies that $Z = \mathcal{Z}(|M|, A)$ for some A.

Formally, we consider a distinguisher $\mathcal{D}$ that has oracle access to either the *real world* $(\mathcal{Y}[\mathcal{R}], \mathcal{R})$ or the *simulated world* $(\mathcal{RO}, \mathcal{S}[\mathcal{RO}])$, where $\mathcal{S}$ is a *simulator*: an algorithm with the same interface as $\mathcal{R}$, with query access to $\mathcal{RO}$, and that aims to behave in such a

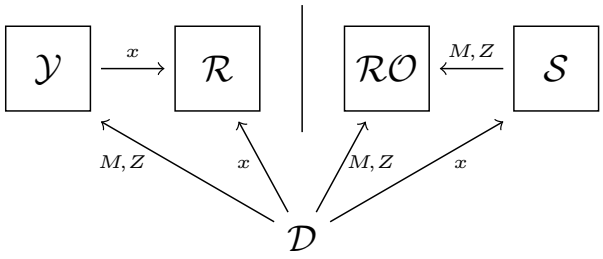

Figure 5: The indifferentiability setting. Distinguisher $\mathcal{D}$ is confined to making construction queries (M, Z) such that $Z = \mathcal{Z}(|M|, A)$ for some A. The labels indicate the input to the oracles. Oracle $\mathcal{R}/\mathcal{S}$ receives as additional input the requested length of the response (if it is an arbitrary function), the direction of the query (if it is a permutation), or the key input and the direction of the query (if it is a block cipher).

way that the real world and simulated world are hard to distinguish. The goal of the distinguisher is to guess which world it is conversing with. It can make two kinds of queries:

- Construction queries on input of a tuple (M, Z) return an n-bit string. In the case of a mode of an arbitrary function the caller may specify the length ℓ of the response and the response is an ℓ-bit string.
- Primitive queries depend on the type of primitive:
  - For an arbitrary function, the input is an arbitrary-length string x and the requested length ℓ of the response, and the output is an ℓ-bit string.
  - For a permutation, the input is a b-bit string x and an indicator whether the permutation or its inverse must be applied, and the output is a b-bit string.
  - For a block cipher the input is a κ-bit string k, a b-bit string x, and an indicator whether the block cipher or its inverse must be applied, and the output is a b-bit string.

The formal definition is given below.

Definition 8. Let $\mathcal{T}$ be a hashing mode of a random primitive $\mathcal{R}$. We denote its template construction function by $\mathcal{Z}$ and template execution by $\mathcal{Y}$. Let $\mathcal{RO}$ be a random oracle with the same domain and range as $\mathcal{Y}$, and let $\mathcal{S}$ be a simulator with oracle access to $\mathcal{RO}$. The indifferentiability advantage of a distinguisher $\mathcal{D}$ is defined as

$$\mathsf{Adv}^{\mathrm{diff}}_{\mathcal{T}[\mathcal{R}],\mathcal{S}}(\mathcal{D}) = \left|\Pr\left[\mathcal{D}^{\mathcal{Y}[\mathcal{R}],\mathcal{R}} = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{RO},\mathcal{S}[\mathcal{RO}]} = 1\right]\right| ,$$

where $\mathcal{D}$ is confined to only making construction queries (M, Z) such that $Z = \mathcal{Z}(|M|, A)$ for some A.

Note that the model is formally applied to $\mathcal{Y}[\mathcal{R}]$, but the subscript of Adv still includes $\mathcal{T}[\mathcal{R}]$. This is for completeness and clarity of reasoning.

Patarin's H-Coefficient Technique. We make use of the H-coefficient technique by Patarin [Pat08, CS14]. Consider any information-theoretic deterministic distinguisher $\mathcal{D}$ whose goal it is to distinguish $\mathcal{O}_1 := \big(\mathcal{Y}[\mathcal{R}], \mathcal{R}\big)$ from $\mathcal{O}_2 := \big(\mathcal{RO}, \mathcal{S}[\mathcal{RO}]\big)$. Assume that it makes a finite amount of queries and these are gathered in a transcript, or *view*, ν. Denote by $D_{\mathcal{O}_1}$ the probability distribution of views of interactions with $\mathcal{O}_1$, and likewise by $D_{\mathcal{O}_2}$ the probability distribution of views of interactions with $\mathcal{O}_2$. A view ν is called *attainable*

if $\Pr[D_{\mathcal{O}_2} = \nu] > 0$, i.e., if it can be attained in the *simulated world*, and we denote by $\mathcal{V}$ the set of attainable views. Patarin's H-coefficient technique states the following for $\mathsf{Adv}^{\mathrm{diff}}_{\mathcal{T}[\mathcal{R}],\mathcal{S}}(\mathcal{D})$:

Lemma 1 (H-coefficient Technique [Pat08, CS14]). *Let* $\mathcal{O}_1 := \big(\mathcal{Y}[\mathcal{R}], \mathcal{R}\big)$ *and* $\mathcal{O}_2 := \big(\mathcal{RO}, \mathcal{S}[\mathcal{RO}]\big)$. *Consider a fixed deterministic distinguisher* $\mathcal{D}$, *and let* $\mathcal{V} = \mathcal{V}_{\mathrm{good}} \cup \mathcal{V}_{\mathrm{bad}}$ *be a partition of the set of views into good and bad views. Let* $\varepsilon \geq 0$ *be a rational number such that for all* $\nu \in \mathcal{V}_{\mathrm{good}}$,

$$\frac{\Pr[D_{\mathcal{O}_1} = \nu]}{\Pr[D_{\mathcal{O}_2} = \nu]} \geq 1 - \varepsilon\,. \tag{2}$$

Then, $\mathsf{Adv}^{\mathrm{diff}}_{\mathcal{T}[\mathcal{R}],\mathcal{S}}(\mathcal{D}) \leq \varepsilon + \Pr[D_{\mathcal{O}_2} \in \mathcal{V}_{\mathrm{bad}}]$.

For a given view $\nu = \{(x_1, y_1), \ldots, (x_q, y_q)\}$, an oracle $\mathcal{O}$ is said to *extend* ν, denoted $\mathcal{O} \vdash \nu$, if $\mathcal{O}(x_i) = y_i$ for all $i = \{1, \ldots, q\}$.

4.2 Mode of an Arbitrary Function

We consider a mode $\mathcal{T}$ of an arbitrary function $\mathcal{F}$, where for chaining values the output is truncated to n bits and for hashing the final node, the output is truncated to the number of bits requested by the caller.

Theorem 1. *Consider a hashing mode* $\mathcal{T}$ *of a random arbitrary function* $\mathcal{F}$, *and assume that it is* subtree-free, radical-decodable, *and* message-decodable. *There exists a simulator* $\mathcal{S}$ *such that for any distinguisher* $\mathcal{D}$ *with total complexity at most* q,

$$\mathsf{Adv}^{\mathrm{diff}}_{\mathcal{T}[\mathcal{F}],\mathcal{S}}(\mathcal{D}) \leq \frac{\binom{q}{2}}{2^n}\,.$$

The simulator $\mathcal{S}$ *makes at most* q *queries to* $\mathcal{RO}$.

The total complexity of $\mathcal{D}$ is counted as the number of evaluations of $\mathcal{F}$ both in construction and primitive queries, noting that any evaluation of $\mathcal{Y}[\mathcal{F}]$ corresponds to a certain amount of evaluations of $\mathcal{F}$. Multiple queries to $\mathcal{Y}[\mathcal{F}]$ or $\mathcal{F}$ for the same input but different requested output lengths count as one query (namely, for the maximum of the lengths). We prove this theorem in Section 5.

To see that the bound of Theorem 1 is tight, consider a mode and an input length $|M|$, where the tree has at least one chaining value with corresponding leaf node x that contains a message chunk m_i. An attacker can make q queries to the primitive $\mathcal{F}$ for different leaf nodes x by taking different values for the message chunk m_i. The probability of finding a collision in the chaining value is $\binom{q}{2}/2^n$ for two different values of message chunk m_i, say α and β. In the real world, any pair of messages of given length $|M|$ that has values α and β respectively in m_i and are equal for the remainder, must collide in a construction query. In the ideal world, construction queries are presented to a random oracle and different inputs collide with negligible probability.

4.3 Mode of a Truncated Permutation

We consider $\mathcal{T}$ with its compression function $\mathcal{F} : \mathbf{Z}_2^b \to \mathbf{Z}_2^n$ being a truncated permutation $\mathcal{P} \xleftarrow{\$} \mathsf{perm}(b)$ for $b \geq n$:

$$\mathcal{F}(s) = \lfloor \mathcal{P}(s) \rfloor_n\,. \tag{3}$$

Theorem 2. *Consider a hashing mode $\mathcal{T}$ of a truncated permutation with $\mathcal{P} \xleftarrow{\$} \mathsf{perm}(b)$, and assume that it is* subtree-free*,* radical-decodable*,* message-decodable*, and* leaf-anchored*. There exists a simulator $\mathcal{S}$ such that for any distinguisher $\mathcal{D}$ with total complexity at most q,*

$$\mathsf{Adv}^{\mathrm{diff}}_{\mathcal{T}[\mathcal{P}],\mathcal{S}}(\mathcal{D}) \leq \frac{\binom{q}{2}+1}{2^n} + \frac{\binom{q}{2}}{2^b}\,.$$

The simulator $\mathcal{S}$ makes at most q queries to $\mathcal{RO}$.

We prove this theorem in Section 6.

4.4 Mode of a Block Cipher

We consider $\mathcal{T}$ with its compression function $\mathcal{F} : \mathbf{Z}_2^{\kappa+b} \to \mathbf{Z}_2^n$ being a truncated block cipher $\mathcal{E} \xleftarrow{\$} \mathsf{bc}(\kappa, b)$ for $b \geq n$. The $(b+\kappa)$-bit compression function input s is parsed as $s = s_{\mathrm{data}} \| s_{\mathrm{key}}$ with $|s_{\mathrm{data}}| = b$ and $|s_{\mathrm{key}}| = \kappa$ and the compression function is defined as:

$$\mathcal{F}(s) = \mathcal{F}(s_{\mathrm{data}} \| s_{\mathrm{key}}) = \lfloor \mathcal{E}(s_{\mathrm{key}}, s_{\mathrm{data}}) \rfloor_n\,, \tag{4}$$

noting that the IV/CV in the first n bits of input s to $\mathcal{F}$ is in s_{data} to prevent abuse of inverse block cipher queries.

Theorem 3. *Consider a hashing mode $\mathcal{T}$ of a block cipher $\mathcal{E} \xleftarrow{\$} \mathsf{bc}(\kappa, b)$, and assume that it is* subtree-free*,* radical-decodable*,* message-decodable*, and* leaf-anchored*. There exists a simulator $\mathcal{S}$ such that for any distinguisher $\mathcal{D}$ with total complexity at most q,*

$$\mathsf{Adv}^{\mathrm{diff}}_{\mathcal{T}[\mathcal{E}],\mathcal{S}}(\mathcal{D}) \leq \frac{\binom{q}{2}+1}{2^n} + \frac{\binom{q}{2}}{2^b}\,.$$

The simulator $\mathcal{S}$ makes at most q queries to $\mathcal{RO}$.

We prove this theorem in Section 7. The expression of the bound is identical to that of Theorem 2. This is because the proof of Theorem 2 considers a simulator that responds uniformly at random for every query (allowing accidental collisions), and as such the idea generalizes to Theorem 3 almost verbatim. In the case of a permutation, however, b is the width of the full permutation input, while in the case of a block cipher, b is only the *data* part of the input and the full width is $\kappa + b$.

5 Proof of Theorem 1: Mode of an Arbitrary Function

We specify a simulator in Section 5.1. Next, we refine the distinguisher to suit our analysis in Section 5.2, and summarize what views look like in Section 5.3. Finally, in alignment with Patarin's H-coefficient technique, we analyze bad views in Section 5.4 and good views in Section 5.5.

5.1 Simulator

Simulator $\mathcal{S}$ has an arbitrary function interface and its main feature is to be consistent with the construction oracle $\mathcal{RO}$ the same way $\mathcal{F}$ does with $\mathcal{Y}[\mathcal{F}]$. In other words, the simulator should behave in such a way that the joint distributions of $\big(\mathcal{RO}, \mathcal{S}[\mathcal{RO}]\big)$ and $\big(\mathcal{Y}[\mathcal{F}], \mathcal{F}\big)$ are close.

Algorithm 1 radicalExtend[Ł](S) and radicalValue[Ł](x)

Interface: radicalExtend[Ł](S)
  if radical(S) returns $\perp$ **then**
    return S
  end if
  $S' \leftarrow S$ ▷ default return value if radical CV not found in Ł
  for all $(x, y) \in$ Ł : $\lfloor y \rfloor_n = S[\text{radical}(S)]$ **do**
    $S' \leftarrow \text{radicalExtend}[Ł](S \cup \{(x, \text{radical}(S))\})$
    if radical(S') returns $\perp$ **then return** S' ▷ stop as soon as no more radical
  end for
  return S' ▷ ultimately return the last tree found

Interface: radicalValue[Ł](x)
  $S \leftarrow \text{radicalExtend}[Ł](\{(x, \perp)\})$
  return $S[\text{radical}(S)]$

It maintains an initially empty list Ł to store all of its query-response tuples, and for $(x, y) \in Ł$, we write $Ł(x) = y$. The input value ℓ specifying the requested length of the response is implicit in the elements. We denote

$$\text{domŁ} = \{x \mid \exists y \text{ such that } (x, y) \in Ł\},$$
$$\text{rngŁ} = \{y \mid \exists x \text{ such that } (x, y) \in Ł\}.$$

In Algorithm 1, we define a recursive function radicalExtend[Ł](S) that extends a tree with nodes constructed from elements in Ł:

$$S \leftarrow \text{radicalExtend}[Ł](S).$$

For each query (x, ℓ) the simulator receives, it applies radicalExtend[Ł](·) to the single-node tree $(x, \perp)$, with abuse of notation denoted as radicalExtend[Ł](x). This function then recursively extends $(x, \perp)$ through radicals. Each extension step consists of adding $\{(x', \text{radical}(S))\}$, where x' is taken from an entry $(x', y) \in Ł$ with $\lfloor y \rfloor_n$ equal to the radical CV in S, or more formally with $\lfloor y \rfloor_n = S[\alpha]$ and $\alpha = \text{radical}(S)$. The function radical($S$), in turn, exists by virtue of radical-decodability (see Definition 5): it returns a radical for any tree in $\mathcal{S}_{\mathcal{T}}^{\text{rad}}$ and $\perp$ otherwise. If there are multiple entries in Ł that are compliant to the radical CV, all possibilities are explored in a tree search fashion.[1] The algorithm returns the first tree S it encounters that is not in $\mathcal{S}_{\mathcal{T}}^{\text{rad}}$. If all trees it encounters are in $\mathcal{S}_{\mathcal{T}}^{\text{rad}}$, this implies that all trees have a radical. In that case it returns the tree S that it last visited.

Our simulator is given in Algorithm 2. It is inspired by Bertoni et al. [BDPV14b] but is simpler. Consider any query $\mathcal{S}(x, \ell)$ for which either $x \notin \text{domŁ}$ or $|Ł(x)| < \ell$. The simulator only checks whether or not x forms the final node of a tree in $\mathcal{S}_{\mathcal{T}}$ with all descendant nodes in Ł by performing radical extension radicalExtend[Ł](x). To avoid ambiguities in this radical extension, it always generates and stores at least n bits in $Ł(x)$. If x (radical-)extends to a tree in $\mathcal{S}_{\mathcal{T}}$, $\mathcal{S}$ extracts the message M and tree template Z, and queries its random oracle on input (M, Z).[2] Otherwise, it generates uniformly random bits and returns those. In case x was queried before, the new random bits are appended to the old response so as to guarantee self-consistency: old queries are never overwritten.

[1] So this is a tree of trees.
[2] Formally, it already calls extract(S) to efficiently learn whether $S \in \mathcal{S}_{\mathcal{T}}$.

Algorithm 2 Simulator for proof of Theorem 1

Interface: $\mathcal{S} : \mathbf{Z}_2^* \to \mathbf{Z}_2^\infty$, $(x, \ell) \mapsto y$
  if $x \notin \text{domŁ}$ **then**
    $\text{Ł} \xleftarrow{\cup} \{(x, \text{emptystring})\}$
  end if
  $\ell' = \max\{\ell, n\}$ ▷ generate $\ell' \geq n$ bits
  if $|\text{Ł}(x)| < \ell'$ **then**
    $S \leftarrow \text{radicalExtend}[\text{Ł}](x)$ ▷ radical-extend tree from single node
    if $S \in \mathcal{S}_\mathcal{T}$ **then** ▷ query completes tree
      $(M, Z) \leftarrow \text{extract}(S)$ ▷ extract message and tree template
      $z_1 \| z_2 = z \leftarrow \mathcal{RO}(M, Z, \ell')$ ▷ $|z_1| = |\text{Ł}(x)|$
      $\text{Ł}(x) \leftarrow \text{Ł}(x) \| z_2$
    else ▷ query does not complete tree
      $z \xleftarrow{\$} \mathbf{Z}_2^{\ell'}$ ▷ at most ℓ' bits need to be generated
      $\text{Ł}(x) \leftarrow \text{Ł}(x) \| z$
    end if
  end if
  return $\lfloor \text{Ł}(x) \rfloor_\ell$

5.2 Distinguisher

Consider any distinguisher $\mathcal{D}$ against the indifferentiability of $\mathcal{T}[\mathcal{F}]$. We build a distinguisher $\mathcal{D}'$ on top of $\mathcal{D}$. Distinguisher $\mathcal{D}'$ operates exactly as $\mathcal{D}$: it makes the same queries, in the same order, and outputs the same decision at the end. However, before $\mathcal{D}'$ outputs its final decision, it takes each construction query-response tuple (M, Z, h) that $\mathcal{D}$ made, and makes all primitive queries to $\mathcal{F}$ resp. $\mathcal{S}$ corresponding to the computation of $\mathcal{Y}[\mathcal{F}/\mathcal{S}](M, Z)$. Clearly, as we are considering total complexity only, this change is only administrative, the total complexity of $\mathcal{D}'$ matches that of $\mathcal{D}$. In addition, as $\mathcal{D}'$ relays the decision by $\mathcal{D}$, its advantage is unchanged:

$$\mathsf{Adv}^{\text{diff}}_{\mathcal{T}[\mathcal{F}],\mathcal{S}}(\mathcal{D}') = \mathsf{Adv}^{\text{diff}}_{\mathcal{T}[\mathcal{F}],\mathcal{S}}(\mathcal{D})\,.$$

The transition from $\mathcal{D}$ to $\mathcal{D}'$ simplifies our analysis significantly, in that all queries that $\mathcal{D}$ makes to the construction *or* primitive oracle, are summarized in just the primitive oracles made by $\mathcal{D}'$. The idea of this transition has appeared before in [CN08, MPN10, MP15], among others.

5.3 Views

We denote by $\mathcal{M} = \{(M_1, Z_1, h_1), \ldots, (M_r, Z_r, h_r)\}$ the view seen by $\mathcal{D}'$ on interaction with the construction oracle, and by $\text{Ł} = \{(x_1, y_1), \ldots, (x_q, y_q)\}$ the view seen by $\mathcal{D}'$ on interaction with the primitive oracle. For $i = 1, \ldots, q$, we denote by $\ell_i = |y_i|$ the amount of bits learned by $\mathcal{D}'$ in primitive query i. Split Ł into

$$\begin{aligned}\mathcal{L}^{\text{rad}} &= \{(x_i, y_i) \in \text{Ł} \mid S = \text{radicalExtend}[\text{Ł}_{i-1}](x_i) \in \mathcal{S}^{\text{rad}}_\mathcal{T}\},\\ \mathcal{L}^{\text{other}} &= \text{Ł} \setminus \mathcal{L}^{\text{rad}}\,.\end{aligned}$$

Denote $\nu = (\mathcal{M}, \mathcal{L}^{\text{rad}}, \mathcal{L}^{\text{other}})$. The set $\mathcal{V}$ denotes any attainable view that can be observed by $\mathcal{D}'$.

An attainable view ν is called *bad* if:

(i) There exist distinct $(x_i, y_i), (x_j, y_j) \in \mathcal{L}^{\text{rad}}$ with $\lfloor y_i \rfloor_n = \lfloor y_j \rfloor_n$;

(ii) There exist distinct $(x_i, y_i), (x_j, y_j) \in \mathcal{L}^{\text{other}}$ with $\lfloor y_i \rfloor_n = \lfloor y_j \rfloor_n$;

(iii) There exist $(x_i, y_i) \in \mathcal{L}^{\text{rad}}$ and $(x_j, y_j) \in \mathcal{L}^{\text{other}}$ with $i < j$ such that $\lfloor y_j \rfloor_n = \text{radicalValue}[Ł_{i-1}](x_i)$.

The first badness condition assures that the simulator's evaluation of the procedure radicalExtend[Ł](x) is efficient, the second condition assures that incomplete trees never collide, and the third condition assures that if a tree has a radical, it never gets extended in a later query.

We will analyze $\Pr[D_{\mathcal{O}_2} \in \mathcal{V}_{\text{bad}}]$ in Section 5.4 and $\frac{\Pr[D_{\mathcal{O}_1}=\nu]}{\Pr[D_{\mathcal{O}_2}=\nu]}$ for good views $\nu \in \mathcal{V}_{\text{good}}$ in Section 5.5, and the bound of Theorem 1 immediately follows.

5.4 Analysis of Bad Views

Badness condition (i) is satisfied with probability $\binom{|\mathcal{L}^{\text{rad}}|}{2}/2^n$, badness condition (ii) is satisfied with probability $\binom{|\mathcal{L}^{\text{other}}|}{2}/2^n$, and badness condition (iii) is satisfied with probability at most $|\mathcal{L}^{\text{rad}}| \cdot |\mathcal{L}^{\text{other}}|/2^n$. As $|\mathcal{L}^{\text{rad}}| + |\mathcal{L}^{\text{other}}| = |Ł| = q$, the probability of a bad view in the simulated world is hence easily computed as

$$\Pr[D_{\mathcal{O}_2} \in \mathcal{V}_{\text{bad}}] \le \frac{\binom{|\mathcal{L}^{\text{rad}}|}{2} + \binom{|\mathcal{L}^{\text{other}}|}{2} + |\mathcal{L}^{\text{rad}}| \cdot |\mathcal{L}^{\text{other}}|}{2^n} = \frac{\binom{|Ł|}{2}}{2^n} = \frac{\binom{q}{2}}{2^n}\,.$$

5.5 Analysis of Good Views

Consider any good view $\nu = (\mathcal{M}, \mathcal{L}^{\text{rad}}, \mathcal{L}^{\text{other}})$.

In the real world $\mathcal{O}_1 = \big(\mathcal{Y}[\mathcal{F}], \mathcal{F}\big)$, for each evaluation $\mathcal{Y}[\mathcal{F}](M_i, Z_i, |h_i|)$ distinguisher $\mathcal{D}'$ has made all primitive queries individually. This means that Ł contains all queries to "construct" $\mathcal{Y}[\mathcal{F}](M_i, Z_i, |h_i|)$, i.e., such that

$$\lfloor Ł(\text{final}(S_i)) \rfloor_{|h_i|} = h_i\,,$$

where S_i is the tree coming from the evaluation of $\mathcal{Y}[\mathcal{F}](M_i, Z_i, |h_i|)$. Therefore, $\mathcal{M}$ does not contain additional information relative to Ł. We obtain that

$$\Pr[D_{\mathcal{O}_1} = \nu] = \Pr[\mathcal{F} \vdash Ł \wedge \mathcal{Y}[\mathcal{F}] \vdash \mathcal{M}] \overset{(i)}{=} \Pr[\mathcal{F} \vdash Ł] = \prod_{i=1}^{q} \frac{1}{2^{\ell_i}}\,,$$

where the randomness is taken over the drawing of $\mathcal{F}$, where $\overset{(i)}{=}$ holds as $\mathcal{M}$ is properly represented by Ł, and where we recall that $|y_i| = \ell_i$ for $i = 1, \ldots, q$.

We will prove that also in the simulated world $\mathcal{O}_2 = \big(\mathcal{RO}, \mathcal{S}[\mathcal{RO}]\big)$ the list $\mathcal{M}$ does not contain extra information relative to Ł. Notice that this de facto boils down to demonstrating that the primitive queries made by $\mathcal{D}'$ do not conflict with those that $\mathcal{D}$ already made by itself (the queries may overlap, though).

Lemma 2. *For every tuple $(M_i, Z_i, h_i) \in \mathcal{M}$ with S_i being the tree coming from the evaluation of $\mathcal{Y}[\mathcal{S}[\mathcal{RO}]](M_i, Z_i, |h_i|)$,*

$$\lfloor Ł(\text{final}(S_i)) \rfloor_{|h_i|} = h_i\,. \tag{5}$$

Proof. Consider any tuple $(M_i, Z_i, h_i) \in \mathcal{M}$. Clearly, $h_i = \mathcal{RO}(M_i, Z_i, |h_i|)$. Distinguisher $\mathcal{D}'$ made all primitive queries corresponding to this construction query tuple, and particularly, $\text{final}(S_i) \in \text{dom}Ł$ for S_i the tree coming from the evaluation of $\mathcal{Y}[\mathcal{S}[\mathcal{RO}]](M_i, Z_i, |h_i|)$.

Write $x_i = \text{final}(S_i)$. Consider an evaluation $S' = \text{radicalExtend}[Ł_{i-1}](x_i)$ on input of x_i, where $S' \in \mathcal{S}_{\mathcal{T}}$. We can distinguish between the following cases.

(1) $S' \neq S_i$. This implies that there are two complete trees of two distinct (M_i, Z_i) and (M', Z') whose final nodes collide: $\text{final}(S_i) = \text{final}(S') = x_i$. This is impossible by *radical-decodability* and badness condition (ii).

(2) $S' = S_i$. In this case, write $(M', Z') \leftarrow \text{extract}(S_i)$. We can distinguish between the following cases.

   (a) $(M', Z') \neq (M_i, Z_i)$. This means that the message and tree template have not been properly extracted from S_i (impossible due to *message-decodability*).

   (b) $(M', Z') = (M_i, Z_i)$. We can distinguish between whether or not the value of $Ł(x_i)$ is already defined at the point of querying x_i.

   - $Ł(x_i)$ was not defined yet. By Algorithm 2, at the point of querying x_i, the simulator obtains the correct (M_i, Z_i) and queries its $\mathcal{RO}$ in a correct way so that (5) is satisfied.
   - $Ł(x_i)$ was already defined. If the definition of $Ł(x_i)$ was done by consultation of $\mathcal{RO}$, by Algorithm 2, $Ł(x_i)$ was defined by proper consultation of $\mathcal{RO}$, as it extracted (M_i, Z_i) correctly. (This case happens, for example, if $\mathcal{D}$ already made all primitive queries corresponding to a construction query, and $\mathcal{D}'$ then duplicates these queries.) On the other hand, if the definition of $Ł(x_i)$ was done without consultation of $\mathcal{RO}$, this means that at the point of defining the response to x_i, $\text{radicalExtend}[Ł_{i-1}](x_i)$ did not return a complete tree. This necessarily means that a radical CV must have been hit at some point (impossible by badness condition (iii)) or that there are two leaved subtrees with colliding root (impossible by badness condition (ii)). □

We obtain for the simulated world $\mathcal{O}_2 = \big(\mathcal{RO}, \mathcal{S}[\mathcal{RO}]\big)$ that

$$\Pr\left[D_{\mathcal{O}_2} = \nu\right] = \Pr\left[\mathcal{S}[\mathcal{RO}] \vdash Ł \wedge \mathcal{RO} \vdash \mathcal{M}\right] \overset{(i)}{=} \Pr\left[\mathcal{S}[\mathcal{RO}] \vdash Ł\right] = \prod_{i=1}^{q} \frac{1}{2^{\ell_i}}\,,$$

where the randomness is taken over the coins of $\mathcal{S}[\mathcal{RO}]$ (and thus, implicitly, in part $\mathcal{RO}$), we use that primitive tuples never get overwritten, and where $\overset{(i)}{=}$ holds due to Lemma 2. We conclude that

$$\frac{\Pr\left[D_{\mathcal{O}_1} = \nu\right]}{\Pr\left[D_{\mathcal{O}_2} = \nu\right]} = 1\,,$$

and we can set ε to 0.

6 Proof of Theorem 2: Mode of a Truncated Permutation

The simulator for the proof of Theorem 2 is given in Section 6.1, a re-modeling of the distinguisher in Section 6.2, a summary of views in Section 6.3, analysis of bad views in Section 6.4, and of good views in Section 6.5.

6.1 Simulator

$\mathcal{F}$ is defined in terms of a random permutation $\mathcal{P} \xleftarrow{\$} \mathsf{perm}(b)$ as

$$\mathcal{F}(a) = \lfloor \mathcal{P}(a) \rfloor_n\,,$$

and the simulator $\mathcal{S}$ is expected to have the same interface as $\mathcal{P}$. This forces us to deal with two additional issues: (i) the adversary can make inverse queries to $\mathcal{S}$ and

Algorithm 3 Simulator for proof of Theorem 2

Interface: $\mathcal{S} : \mathbf{Z}_2^b \to \mathbf{Z}_2^b\ ,\ x \mapsto y$
  if $x \notin \text{domŁ}$ **then**
    $S \leftarrow \text{radicalExtend}[\mathcal{L}^{\text{fwd}}](x)$ ▷ radical-extend tree from single node
    if $S \in \mathcal{S}_{\mathcal{T}}$ **then** ▷ query completes tree
      $(M, Z) \leftarrow \text{extract}(S)$ ▷ extract message and tree template
      $z \xleftarrow{\$} \mathbf{Z}_2^{b-n}$
      $y \leftarrow \mathcal{RO}(M, Z)\|z$
    else ▷ query does not complete tree
      $y \xleftarrow{\$} \mathbf{Z}_2^b$
    end if
    $Ł(x) \leftarrow y$
  end if
  return $Ł(x)$

Interface: $\mathcal{S}^{-1} : \mathbf{Z}_2^b \to \mathbf{Z}_2^b\ ,\ y \mapsto x$
  if $y \notin \text{rngŁ}$ **then**
    $x \xleftarrow{\$} \mathbf{Z}_2^b$
    $Ł^{-1}(y) \leftarrow x$
  end if
  return $Ł^{-1}(y)$

construct collisions and preimages for $\mathcal{F}$, and (ii) the adversary can check for permutation inconsistency. We deal with the latter by simply generating random responses and labeling (the rare) accidental collisions as bad events. For the former issue, the key idea of $\mathcal{S}$ is that inverse queries rarely contribute to a tree in $\mathcal{S}_{\mathcal{T}}$ and that the simulator ignores them in the radical-extension process radicalExtend. Indeed, an inverse query would only be valuable to the distinguisher if it hits the IV or if it would extend a leaf subtree.

The simulator now maintains two lists $\mathcal{L}^{\text{fwd}}$ and $\mathcal{L}^{\text{inv}}$, storing queries (x, y) made in forward or inverse direction, respectively. We define $Ł = \mathcal{L}^{\text{fwd}} \cup \mathcal{L}^{\text{inv}}$, and write domŁ and rngŁ as before. If the simulator gets as input a query ($x \mapsto y$ in forward or $y \mapsto x$ in inverse direction) that is multiply defined in Ł, it just responds with one of those (in this case, the simulator has failed; it will be covered by a bad event). Our simulator for the proof of Theorem 2 is given in Algorithm 3. Its changes from Algorithm 2 are in the fact that radicalExtend is evaluated on $\mathcal{L}^{\text{fwd}}$ only, in the straightforward implementation of an inverse interface and in the fact that we no longer have to deal with variable length responses.

6.2 Distinguisher

Consider any distinguisher that has total complexity q. Note that any construction query by $\mathcal{D}$ entails a certain amount of primitive queries. Our goal is to bound $\mathsf{Adv}^{\text{diff}}_{\mathcal{T}[\mathcal{P}],\mathcal{S}}(\mathcal{D})$ (see Definition 8).

We build a distinguisher $\mathcal{D}'$ along the same lines as Section 5.2. After $\mathcal{D}$ is finished, $\mathcal{D}'$ makes forward permutation queries corresponding to all construction queries made by $\mathcal{D}$. As before, it has equal advantage and equal complexity.

6.3 Views

We denote by $\mathcal{M} = \{(M_1, Z_1, h_1), \ldots, (M_r, Z_r, h_r)\}$ the view seen by $\mathcal{D}'$ on interaction with the construction oracle, and by $Ł = \{(x_1, y_1), \ldots, (x_q, y_q)\}$ the view seen by $\mathcal{D}'$ on

interaction with the primitive oracle. Recall the split of Ł into $\mathcal{L}^{\text{fwd}}$ and $\mathcal{L}^{\text{inv}}$. We further split $\mathcal{L}^{\text{fwd}}$ into

$$\begin{aligned}\mathcal{L}^{\text{rad}} &= \{(x_i, y_i) \in Ł \mid S = \text{radicalExtend}[Ł_{i-1}](x_i) \in \mathcal{S}_{\mathcal{T}}^{\text{rad}}\},\\ \mathcal{L}^{\text{other}} &= \mathcal{L}^{\text{fwd}} \setminus \mathcal{L}^{\text{rad}}.\end{aligned}$$

Denote $\nu = (\mathcal{M}, \mathcal{L}^{\text{rad}}, \mathcal{L}^{\text{other}}, \mathcal{L}^{\text{inv}})$. The set $\mathcal{V}$ denotes any attainable view that can be observed by $\mathcal{D}'$.

An attainable view ν is called *bad* if:

(i) There exist distinct $(x_i, y_i), (x_j, y_j) \in \mathcal{L}^{\text{rad}}$ with $\lfloor y_i \rfloor_n = \lfloor y_j \rfloor_n$;

(ii) There exist distinct $(x_i, y_i), (x_j, y_j) \in \mathcal{L}^{\text{other}}$ with $\lfloor y_i \rfloor_n = \lfloor y_j \rfloor_n$;

(iii) There exist $(x_i, y_i) \in \mathcal{L}^{\text{rad}}$ and $(x_j, y_j) \in \mathcal{L}^{\text{other}}$ with $i < j$ such that $\lfloor y_j \rfloor_n = \text{radicalValue}[Ł_{i-1}](x_i)$;

(iv) There exist $(x_i, y_i) \in \mathcal{L}^{\text{inv}}$ such that $\lfloor x_i \rfloor_n = \text{IV}$;

(v) There exist $(x_i, y_i) \in \mathcal{L}^{\text{inv}}$ and $(x_j, y_j) \in \mathcal{L}^{\text{fwd}}$ such that $\lfloor x_i \rfloor_n = \lfloor y_j \rfloor_n$;

(vi) There are distinct $(x_i, y_i), (x_j, y_j) \in (\mathcal{L}^{\text{fwd}} \cup \mathcal{L}^{\text{inv}})$ with $x_i = x_j$ or $y_i = y_j$.

The first three conditions are identical to those in Section 5. Conditions (iv) and (v) cover issues that may arise from omitting inverse queries in radical-extension. Badness condition (vi), finally, is triggered if the primitive responses do not appear like a permutation.

We will analyze $\Pr[D_{\mathcal{O}_2} \in \mathcal{V}_{\text{bad}}]$ in Section 6.4 and $\frac{\Pr[D_{\mathcal{O}_1} = \nu]}{\Pr[D_{\mathcal{O}_2} = \nu]}$ for good views $\nu \in \mathcal{V}_{\text{good}}$ in Section 6.5, and the bound of Theorem 2 immediately follows.

6.4 Analysis of Bad Views

The analysis of badness conditions (i)-(iii) is identical to that of Section 5.4, and the probability that any of them is violated is at most $\binom{|\mathcal{L}^{\text{fwd}}|}{2}/2^n$. Badness condition (iv) is satisfied with probability $|\mathcal{L}^{\text{inv}}|/2^n$, badness condition (v) with probability $|\mathcal{L}^{\text{inv}}| \cdot |\mathcal{L}^{\text{fwd}}|/2^n$, and badness condition (vi) with probability at most $\binom{|Ł|}{2}/2^b$, where we recall that $b \geq n$ is the width of the permutation. As $|\mathcal{L}^{\text{rad}}| + |\mathcal{L}^{\text{other}}| = |\mathcal{L}^{\text{fwd}}|$ and $|\mathcal{L}^{\text{fwd}}| + |\mathcal{L}^{\text{inv}}| = q$, the probability of a bad view in the simulated world is hence easily computed as

$$\begin{aligned}\Pr[D_{\mathcal{O}_2} \in \mathcal{V}_{\text{bad}}] &\leq \frac{\binom{|\mathcal{L}^{\text{fwd}}|}{2} + |\mathcal{L}^{\text{inv}}| + |\mathcal{L}^{\text{inv}}| \cdot |\mathcal{L}^{\text{fwd}}|}{2^n} + \frac{\binom{|Ł|}{2}}{2^b}\\ &= \frac{\binom{|Ł|}{2} - \binom{|\mathcal{L}^{\text{inv}}|}{2} + |\mathcal{L}^{\text{inv}}|}{2^n} + \frac{\binom{|Ł|}{2}}{2^b}\\ &= \frac{\binom{q}{2}}{2^n} + \frac{-\binom{|\mathcal{L}^{\text{inv}}|}{2} + |\mathcal{L}^{\text{inv}}|}{2^n} + \frac{\binom{q}{2}}{2^b}.\end{aligned}$$

One can observe that $-\binom{|\mathcal{L}^{\text{inv}}|}{2} + |\mathcal{L}^{\text{inv}}| \leq 1$ for any $|\mathcal{L}^{\text{inv}}|$, and we obtain

$$\Pr[D_{\mathcal{O}_2} \in \mathcal{V}_{\text{bad}}] \leq \frac{\binom{q}{2} + 1}{2^n} + \frac{\binom{q}{2}}{2^b}.$$

6.5 Analysis of Good Views

The derivation is very similar to that of Section 5.5, with the difference that the primitive is now invertible. Consider any good view $\nu = (\mathcal{M}, \mathcal{L}^{\text{rad}}, \mathcal{L}^{\text{other}}, \mathcal{L}^{\text{inv}})$.

As before, the real world $\mathcal{O}_1 = \big(\mathcal{Y}[\mathcal{P}], \mathcal{P}\big)$ satisfies that $\mathcal{M}$ does not contain additional information relative to Ł. We obtain that

$$\Pr\left[D_{\mathcal{O}_1} = \nu\right] = \Pr\left[\mathcal{P} \vdash Ł \wedge \mathcal{Y}[\mathcal{P}] \vdash \mathcal{M}\right] \overset{(i)}{=} \Pr\left[\mathcal{P} \vdash Ł\right] = \frac{(2^b - q)!}{2^b!}\,,$$

where the randomness is taken over the drawing of $\mathcal{P} \xleftarrow{\$} \mathsf{perm}(b)$, and where $\overset{(i)}{=}$ holds as $\mathcal{M}$ is properly represented by Ł.

Also in the simulated world $\mathcal{O}_2 = \big(\mathcal{RO}, \mathcal{S}[\mathcal{RO}]\big)$ the list $\mathcal{M}$ does not contain extra information relative to Ł.

Lemma 3. *For every tuple $(M_i, Z_i, h_i) \in \mathcal{M}$ with S_i being the tree coming from the evaluation of $\mathcal{Y}[\mathcal{S}[\mathcal{RO}]](M_i, Z_i)$,*

$$\lfloor Ł(\text{final}(S_i)) \rfloor_n = h_i\,. \tag{6}$$

Proof. The proof is identical to that of Lemma 2, in addition using that by *leaf-anchoring* and badness condition (iv) and (v) inverse queries do not play a role in the construction of trees. Badness condition (vi), finally, assures permutation consistency and that there are no colliding (sub-)trees. □

We obtain for the simulated world $\mathcal{O}_2 = \big(\mathcal{RO}, \mathcal{S}[\mathcal{RO}]\big)$ that

$$\Pr\left[D_{\mathcal{O}_2} = \nu\right] = \Pr\left[\mathcal{S}[\mathcal{RO}] \vdash Ł \wedge \mathcal{RO} \vdash \mathcal{M}\right] \overset{(i)}{=} \Pr\left[\mathcal{S}[\mathcal{RO}] \vdash Ł\right] = \frac{1}{2^{bq}}\,,$$

where the randomness is taken over the coins of $\mathcal{S}[\mathcal{RO}]$ (and thus, implicitly, in part $\mathcal{RO}$), and where $\overset{(i)}{=}$ holds due to Lemma 3. We conclude that

$$\frac{\Pr\left[D_{\mathcal{O}_1} = \nu\right]}{\Pr\left[D_{\mathcal{O}_2} = \nu\right]} = \frac{(2^b - q)!2^{bq}}{2^b!} \geq 1\,,$$

and we can set ε to 0.

7 Proof of Theorem 3: Mode of a Block Cipher

The proof is identical to that of Theorem 2, barring some additional bookkeeping (the simulator maintains a family of 2^κ lists $Ł_k$, one for every key input k to $\mathcal{E}$), and the analysis of good views. The reason is that in the proof of Theorem 2 we considered a simplified simulator that responds uniformly at random (without maintaining permutation consistency) on every query. We leave notational changes implicit, and fast-forward to the analysis of good views. For $k \in \mathbf{Z}_2^\kappa$, define by q_k the number of tuples in Ł having key input k. We have for $\mathcal{O}_1 = \big(\mathcal{Y}[\mathcal{E}], \mathcal{E}\big)$:

$$\Pr\left[D_{\mathcal{O}_1} = \nu\right] = \Pr\left[\mathcal{E} \vdash Ł \wedge \mathcal{Y}[\mathcal{E}] \vdash \mathcal{M}\right] \overset{(i)}{=} \Pr\left[\mathcal{E} \vdash Ł\right] = \prod_{k \in \mathbf{Z}_2^\kappa} \frac{(2^b - q_k)!}{2^b!}\,,$$

where the randomness is taken over the drawing of $\mathcal{E} \xleftarrow{\$} \mathsf{bc}(\kappa, b)$, and where $\overset{(i)}{=}$ holds as $\mathcal{M}$ is properly represented by Ł. The simulated world $\mathcal{O}_2 = \big(\mathcal{RO}, \mathcal{S}[\mathcal{RO}]\big)$ satisfies

$$\Pr\left[D_{\mathcal{O}_2} = \nu\right] = \frac{1}{2^{bq}}$$

as before. As $\sum_{k \in \mathbf{Z}_2^\kappa} q_k = q$, we conclude that

$$\frac{\Pr\left[D_{\mathcal{O}_1} = \nu\right]}{\Pr\left[D_{\mathcal{O}_2} = \nu\right]} = \prod_{k \in \mathbf{Z}_2^\kappa} \frac{(2^b - q_k)! 2^{b q_k}}{2^b!} \geq 1\,,$$

and we can set ε to 0.

8 Applications

We start by presenting minimal examples of sound modes in Section 8.1. We elaborate on the role of the IV in practice in Section 8.2. We subsequently apply our conditions and results to two existing modes from literature, namely suffix-free Merkle-Damgård in Section 8.3 and Enveloped Merkle-Damgård in Section 8.4. We finally discuss some tree hashing modes that are used in practice in Section 8.5, give an overview of the Sakura encoding [BDPV14a] in the context of our modes in Section 8.6, and apply our analysis to message authentication in Section 8.7.

8.1 Minimal Sequential and Tree Hashing Modes

The simplest sequential hashing mode that meets subtree-freeness, radical-decodability, and message-decodability is the following. Consider a compression function $\mathcal{F} : \mathbf{Z}_2^b \to \mathbf{Z}_2^n$. Each node ends with 2 frame bits: the first one is 0 in the leaf nodes and 1 in all other nodes and the second one is 1 in the final node and 0 in all other nodes. The leaf node consists of $b - 2$ message pointer bits; the subsequent nodes (except for the final node) consist of n chaining pointer bits and $b - n - 2$ message pointer bits, and the final node consists of n chaining pointer bits, and the remaining message pointer bits (at most $b - n - 3$) followed by a frame bit 1 and up to $b - n - 3$ frame bits with value 0. The tree template is illustrated in Figure 6a.

It is relatively easy to verify that the three sufficient conditions are satisfied:

- Subtree-freeness can easily be explained by contradiction. Assume S' a subtree of S and both are in $\mathcal{S}_\mathcal{T}$. As only the root in S' ends in 11 and only the root in S ends in 11, they must be equal. Then, the child node of this root in both trees must be the same and this argument can be extended recursively until the leaf node, that must be reached simultaneously in both S' and S due to its ending in 00 rather than 10. So S' and S must be equal.
- For radical-decodability we can take $\mathcal{S}_\mathcal{T}^{\text{rad}} = \mathcal{S}_\mathcal{T}^{\text{sub}} \setminus \mathcal{S}_\mathcal{T}^{\text{leaf}}$. Any tree in this set has a node without a child that ends in 10 and the radical is its first n bits.
- For message-decodability, reconstructing the template from the tree is trivial and the message consists of the concatenation of the message blocks in all nodes.

If the underlying primitive is a permutation or a block cipher (as in Sections 4.3 or 4.4), we in addition need leaf-anchoring. This can be achieved by putting an n-bit IV at the start of the leaf node. The adjusted template is depicted in Figure 6b. The introduction of the IV does not impact the other three conditions. It is interesting to note that the function does not need a feed-forward to overcome the invertibility property of the underlying primitive.

As for tree hashing, the template of Figure 1a can be considered as a minimal example: although it is described for a message of four blocks, the example straightforwardly stretches in the vein of the minimal sequential mode and demonstrating that the three properties are satisfied can be done with similar arguments. If the compression function is a truncated permutation or block cipher, one additionally needs to include an IV at the start of every leaf.

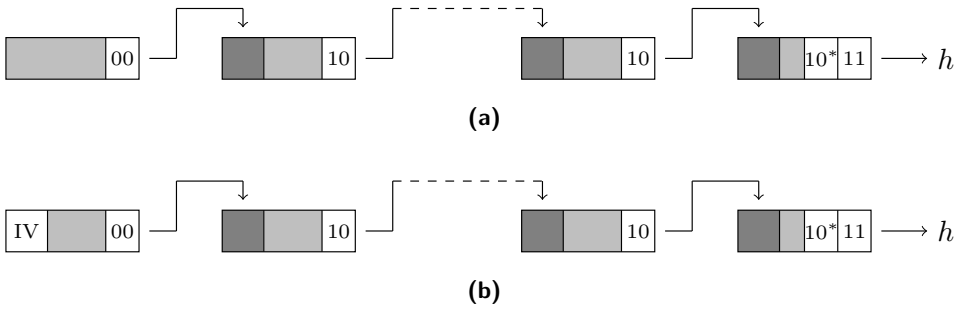

Figure 6: Tree template of a minimal sequential hashing mode of (a) an arbitrary function and (b) a truncated permutation or block cipher. White blocks contain frame bits, light gray blocks contain message pointer bits, and dark gray chaining blocks contain chaining pointer bits. See Section 8.2 on the possibility of omitting the first frame bit in case (b).

8.2 Role of the IV

When leaf-anchoring is applied, one can identify nodes as non-leaf with certainty by the absence of the IV and leaf nodes with high probability by the presence of the IV. This allows for relaxing subtree-freeness slightly, and a subtree $S \in \mathcal{S}_{\mathcal{T}}^{\text{sub}}$ that has an IV as radical CV can also be in $\mathcal{S}_{\mathcal{T}}$. When receiving the final node of such a tree, radical-extension that is aware of the IV will consider the CV with value IV as a leaf node and will not look further. This may lead to an incorrect simulator query to the random oracle, but that is not a problem as a CV that happens to be an IV is a rare event that can be taken into account by treating views that contain forward queries with $\lfloor y \rfloor_n = \text{IV}$ as bad views. This increases the probability of a bad view by a term at most $q/2^n$, basically turning $\binom{q}{2}/2^n$ to $\binom{q+1}{2}/2^n$. Concluding, in the presence of leaf-anchoring, one can slightly relax the condition of subtree-freeness by excluding from $\mathcal{S}_{\mathcal{T}}$ nodes that have a CV with the IV-value. This typically allows saving a frame bit: instead of having a frame bit for distinguishing leaf nodes from non-leaf node, the presence or absence of the IV is sufficient. For the concrete example of Figure 6b, the first frame bit (0 for the leaves and 1 for the non-leaves) can be omitted.

If the compression function is a truncated permutation or block cipher, applying leaf anchoring is quite natural and this optimization makes sense. However, sequential modes applied in former industry standard MD5 and true standards SHA-1 and SHA-2 all use leaf-anchoring, and several flavors of Merkle-Damgård (MD) also use leaf-anchoring, most notably suffix-free MD, prefix-free MD, and Enveloped MD. In MD5, SHA-1, SHA-2, and suffix-free MD the leaf-anchoring does not help: as already demonstrated by Coron et al. [CDMP05], any tree S can be extended by adding a node at the end, a weakness widely known as length extension (see also Section 8.3). In prefix-free MD and Enveloped MD, leaf-anchoring does help as the final node can be identified unambiguously from the leaf node. This confirms earlier indifferentiability proofs on prefix-free Merkle-Damgård by Coron et al. [CDMP05], Chang et al. [CLNY06], Bhattacharyya et al. [BMN09], and Bellare and Ristenpart [BR06]. We detail the case of Enveloped MD in Section 8.4.

8.3 Suffix-Free Merkle-Damgård

Suffix-free Merkle-Damgård gets as input an arbitrarily sized message $M \in \mathbf{Z}_2^*$ and outputs a digest $h \in \mathbf{Z}_2^n$ using a compression function $\mathcal{F} : \mathbf{Z}_2^b \to \mathbf{Z}_2^n$ for $b > n$. First, the message M is padded into $(b-n)$-bit message blocks $M_1 \| \dots \| M_\ell = \text{pad-sf}(M)$ using a suffix-free padding function pad-sf: it satisfies the property that there do not exist M, M', X such that $X \| \text{pad-sf}(M) = \text{pad-sf}(M')$. It can be achieved by ending with a fixed-length encoding of

the message length. Next, the function $\mathcal{F}$ is iteratively evaluated as

$$\mathrm{CV}_i = \mathcal{F}(\mathrm{CV}_{i-1} \| M_i)$$

for $i = 1, \ldots, \ell$, where $\mathrm{CV}_0 := \mathrm{IV}$ is an initial value that is customary included, and the output of the hash function is defined as $h = \mathrm{CV}_\ell$.

Suffix-free Merkle-Damgård is obviously not subtree-free (Definition 4) due to the possibility of a simple length extension attack: it is possible to derive three non-empty messages M, M', X such that $\text{pad-sf}(M) \| X = \text{pad-sf}(M')$. Our indifferentiability results do therefore not apply to the mode. Non-surprisingly, differentiability of suffix-free Merkle-Damgård was already demonstrated by Coron et al. [CDMP05], using the length extension attack.

8.4 Enveloped Merkle-Damgård

Enveloped Merkle-Damgård [BR06] gets as input an arbitrarily sized message $M \in \mathbf{Z}_2^*$ and outputs a digest $h \in \mathbf{Z}_2^n$ using a compression function $\mathcal{F} : \mathbf{Z}_2^b \to \mathbf{Z}_2^n$ for $b \geq 2n + 64$. First, the message M is injectively padded into $(b-n)$-bit message blocks $M_1 \| \ldots \| M_\ell = \mathrm{pad}(M)$. Next, the function $\mathcal{F}$ is iteratively evaluated as

$$\mathrm{CV}_i = \mathcal{F}(\mathrm{CV}_{i-1} \| M_i)$$

for $i = 1, \ldots, \ell$, where $\mathrm{CV}_0 := \mathrm{IV}_1$ is an initial value that is customary included. The output of the hash function is defined as

$$h = \mathcal{F}(\mathrm{IV}_2 \| \mathrm{CV}_\ell \| \langle |M| \rangle_{64}) \,,$$

where $\langle |M| \rangle_{64}$ is the 64-bit encoding of the message length (this also causes the presence of "64" in the condition $b \geq 2n + 64$).

We will verify whether the mode satisfies the three sufficient conditions. Clearly, any tree instance $S \in \mathcal{S}_\mathcal{T}$ is of the form

$$(\mathrm{IV}_1 \| M_1) \longrightarrow (\mathrm{CV}_1 \| M_2) \longrightarrow \cdots \longrightarrow (\mathrm{CV}_{\ell-1} \| M_\ell) \longrightarrow (\mathrm{IV}_2 \| \mathrm{CV}_\ell \| \langle |M| \rangle_{64}) \,,$$

where $M_1 \| \ldots \| M_\ell = \mathrm{pad}(M)$.

- Subtree-freeness is satisfied provided that none of the chaining values CV_i collides with IV_1 or IV_2. In other words, the mode is *not unconditionally* subtree-free, but the condition is satisfied with high probability (see Section 8.2).
- For radical-decodability, we have $\mathcal{S}_\mathcal{T}^{\mathrm{rad}} = \mathcal{S}_\mathcal{T}^{\mathrm{sub}} \setminus \mathcal{S}_\mathcal{T}^{\mathrm{leaf}}$. Consider any $S \in \mathcal{S}_\mathcal{T}^{\mathrm{rad}}$. A single-node final subtree has a node without a child of the form $(\mathrm{IV}_2 \| \mathrm{CV}_\ell \| \langle |M| \rangle_{64})$ and any subtree that is not final or has more than one node has a node without a child of the form $(\mathrm{CV} \| M_i)$. In the single-node final subtree case the presence of IV_2 points to the radical CV_ℓ and in the other case, the fact that the first n bits of the node are not IV_1 or IV_2 allows to conclude that they form a radical. Clearly, also radical-decodability is conditional on chaining values not colliding with IV_1 and IV_2.
- For message-decodable, the function extract() takes $M_1 \| \ldots \| M_\ell$ and strips off the injective padding pad(). The message length encoding $\langle |M| \rangle_{64}$ is redundant.

8.5 Tree Hashing Modes in the Wild

Bitcoin [Nak08, GKL15] is a peer-to-peer electronic cash system that make use of tree hashing based on Merkle trees [Mer79]. Remarkably, the employed tree hashing mode satisfies none of the three main conditions, and it is easy to generate collisions or perform

length extension attacks. However, abusing these properties seems to be made infeasible by the higher-level layers of the protocol.

The Tree Hash Exchange (THEX) format is proposed for assisting in checking the integrity of exchanged files, allowing arbitrary subranges of bytes to be verified before the entire file has been received [CM03]. It is not subtree-free, hence it is vulnerable to length extension attacks. This may not be a problem for the typical use cases.

The generalization and simplification over the five required properties of Dodis et al. [DRRS09] suggests that the mode of MD6 [ABC+09] can be made more efficient and simpler without sacrificing security. In particular, MD6 satisfies its sufficiency conditions at the cost of 73 frame bits per node call: a 64-bit word for the location of the node in the tree, a single bit for indicating whether the node is final, and an 8-bit encoding of the maximum tree height. MD6 in addition allocates 960 bits per compression function call to a constant Q, in order to prove indifferentiability of this compression function and to make generic composition go through [ABC+09, DRRS09]. As we showed in Sections 8.1 and 8.2, a single frame bit per node and an additional n-bit IV per leaf node (with n twice the targeted security strength, so typically $n = 256$) would have been sufficient.

8.6 Sakura

Sakura [BDPV14a] is not a tree hashing mode itself but rather an encoding that allows a wide range of tree hashing modes of arbitrary functions. It specifies how to encode message fragments and chaining values in nodes, and this encoding ensures the conditions subtree-freeness, radical-decodability, and message-decodability. The encoding is performed by having a frame bit at the end of each node indicating whether it is a final node or not, and by making each node decodable into a message chunk and chaining values.

One may see Sakura as a layer within the template generation processing: The mode-specific processing would map the message length and parameters into a *pre-template* that specifies the tree topology and where the message chunks go and that is on a higher abstraction level as a template. The encoding of this pre-template into a template is then done according to Sakura. A hash function that makes use of Sakura encoding is KangarooTwelve [BDP+18]. This hash function uses a tree structure to exploit parallelism, and its underlying arbitrary function is a variant of Keccak.

8.7 Application to Message Authentication

Another interesting application is the definition of a MAC function construction that would form an alternative for SHA-256 HMAC [KBC97, Bel06]. This function computes a MAC on an n-byte message that requires $\lceil \frac{n+9}{64} \rceil + 3$ calls of the SHA-256 compression function. Using the insights of this paper, we can build a MAC function that offers the same security strength, but only requires $\lceil \frac{n+33}{64} \rceil$ calls to the SHA-256 compression function (without the Davies-Meyer feedforward). This mode would be purely sequential and simply take, in the compression function input, a bit indicating whether the node is final or not, a 255-bit CV or IV (for the leaf node), and bits from the concatenation of the key and message after padding with a single 1 and 0s. For messages shorter than 32 bytes this MAC takes 1 compression function call instead of 4.

Note that SHA-256 HMAC achieves security based on the assumption that the SHA-256 block cipher is ideal in a three-step reduction: HMAC assumes an underlying hash function that is secure (tolerating length-extension), SHA-256 is collision resistant if the underlying compression function is collision resistant, and the SHA-256 compression function is collision resistant if the underlying block cipher is ideal. In our approach the security reduction only has a single level: the hash function is secure if the underlying block cipher is ideal.

Acknowledgments

Joan Daemen is supported by the European Research Council under the ERC advanced grant agreement under grant ERC-2017-ADG Nr. 788980 ESCADA. Bart Mennink is supported by a postdoctoral fellowship from the Netherlands Organisation for Scientific Research (NWO) under Veni grant 016.Veni.173.017.

References

[ABC+09] Benjamin Agre, Daniel V. Bailey, Christopher Crutchfield, Yevgeniy Dodis, Kermin Elliott Fleming, Asif Khan, Jayant Krishnamurthy, Yuncheng Lin, Leo Reyzin, Emily Shen, Jim Sukha, Drew Sutherland, Eran Tromer, and Yiqun Lisa Yin. The MD6 hash function – A proposal to NIST for SHA-3, 2009.

[ALM12] Elena Andreeva, Atul Luykx, and Bart Mennink. Provable security of BLAKE with non-ideal compression function. In Lars R. Knudsen and Huapeng Wu, editors, *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, volume 7707 of *Lecture Notes in Computer Science*, pages 321–338. Springer, 2012.

[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.

[AMPŠ12] Elena Andreeva, Bart Mennink, Bart Preneel, and Marjan Škrobot. Security analysis and comparison of the SHA-3 finalists BLAKE, Grøstl, JH, Keccak, and Skein. In Aikaterini Mitrokotsa and Serge Vaudenay, editors, *Progress in Cryptology - AFRICACRYPT 2012 - 5th International Conference on Cryptology in Africa, Ifrance, Morocco, July 10-12, 2012. Proceedings*, volume 7374 of *Lecture Notes in Computer Science*, pages 287–305. Springer, 2012.

[BCS05] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 526–541. Springer, 2005.

[BD07] Eli Biham and Orr Dunkelman. A framework for iterative hash functions – HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007.

[BDP+18] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. KangarooTwelve: Fast hashing based on Keccak-p. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 400–418. Springer, 2018.

[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.

[BDPV14a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sakura: A flexible coding for tree hashing. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings*, volume 8479 of *Lecture Notes in Computer Science*, pages 217–234. Springer, 2014.

[BDPV14b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sufficient conditions for sound tree and sequential hashing modes. *Int. J. Inf. Sec.*, 13(4):335–353, 2014.

[Bel06] Mihir Bellare. New proofs for NMAC and HMAC: security without collision-resistance. In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 602–619. Springer, 2006.

[BKL+09] Mihir Bellare, Tadayoshi Kohno, Stefan Lucks, Niels Ferguson, Bruce Schneier, Doug Whiting, Jon Callas, and Jesse Walker. Provable security support for the Skein hash family, 2009.

[BMN09] Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Indifferentiability characterization of hash functions and optimal bounds of popular domain extensions. In Bimal K. Roy and Nicolas Sendrier, editors, *Progress in Cryptology - INDOCRYPT 2009, 10th International Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings*, volume 5922 of *Lecture Notes in Computer Science*, pages 199–218. Springer, 2009.

[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.

[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.

[BR97] Mihir Bellare and Phillip Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. 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 470–484. Springer, 1997.

[BR06] Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the EMD transform. In Lai and Chen [LC06], pages 299–314.

[Bra90] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2002.

[BRSS10] John Black, Phillip Rogaway, Thomas Shrimpton, and Martijn Stam. An analysis of the blockcipher-based hash functions from PGV. *J. Cryptology*, 23(4):519–545, 2010.

[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.

[CFN+12] Anne Canteaut, Thomas Fuhr, Marìa Naya-Plasencia, Pascal Paillier, Jean-René Reinhard, and Marion Videau. A unified indifferentiability proof for permutation- or block cipher-based hash functions. Cryptology ePrint Archive, Report 2012/363, 2012.

[CLNY06] Donghoon Chang, Sangjin Lee, Mridul Nandi, and Moti Yung. Indifferentiable security analysis of popular hash functions with prefix-free padding. In Lai and Chen [LC06], pages 283–298.

[CM03] Justin Chapweske and Gordon Mohr. Tree Hash EXchange format (THEX), 2003.

[CN08] Donghoon Chang and Mridul Nandi. Improved indifferentiability security analysis of chopMD hash function. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 429–443. Springer, 2008.

[CNY11] Donghoon Chang, Mridul Nandi, and Moti Yung. Indifferentiability of the hash algorithm BLAKE. Cryptology ePrint Archive, Report 2011/623, 2011.

[CS14] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350. Springer, 2014.

[Dam89] Ivan Damgård. A design principle for hash functions. In Brassard [Bra90], pages 416–427.

[DDV11] Joan Daemen, Tony Dusenge, and Gilles Van Assche. Sufficient conditions for sound hashing using a truncated permutation. Cryptology ePrint Archive, Report 2011/459, 2011.

[DRRS09] Yevgeniy Dodis, Leonid Reyzin, Ronald L. Rivest, and Emily Shen. Indifferentiability of permutation-based compression functions and tree-based modes of operation, with applications to MD6. In Dunkelman [Dun09], pages 104–121.

[Dun09] 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*. Springer, 2009.

[FIP15] FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2015.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[HPY07] Shoichi Hirose, Je Hong Park, and Aaram Yun. A simple variant of the Merkle-Damgård scheme with a permutation. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 113–129. Springer, 2007.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: keyed-hashing for message authentication. *RFC*, 2104:1–11, 1997.

[KCP16] John Kelsey, Shu-jen Chang, and Ray Perlner. NIST SP 800-185: SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash, December 2016.

[KM07] Hidenori Kuwakado and Masakatu Morii. Indifferentiability of single-block-length and rate-1 compression functions. *IEICE Transactions*, 90-A(10):2301–2308, 2007.

[LC06] Xuejia Lai and Kefei Chen, editors. *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, volume 4284 of *Lecture Notes in Computer Science*. Springer, 2006.

[LCL+03] Wonil Lee, Donghoon Chang, Sangjin Lee, Soo Hak Sung, and Mridul Nandi. New parallel domain extenders for UOWHF. In Chi-Sung Laih, editor, *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, volume 2894 of *Lecture Notes in Computer Science*, pages 208–227. Springer, 2003.

[LCL+05] Wonil Lee, Donghoon Chang, Sangjin Lee, Soo Hak Sung, and Mridul Nandi. Construction of UOWHF: two new parallel methods. *IEICE Transactions*, 88-A(1):49–58, 2005.

[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.

[Luc05] Stefan Lucks. A failure-friendly design principle for hash functions. In Bimal K. Roy, editor, *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005.

[Mer79] Ralph Charles Merkle. *Secrecy, authentication and public key systems.* PhD thesis, UMI Research Press, 1979.

[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987.

[Mer89] Ralph C. Merkle. One way hash functions and DES. In Brassard [Bra90], pages 428–446.

[Mer90] Ralph C. Merkle. A fast software one-way hash function. *J. Cryptology*, 3(1):43–58, 1990.

[Mer92] Ralph C. Merkle. *Method of Providing Digital Signatures.* US Patent Number 4,309,569, January 5, 1992.

[MP15] Bart Mennink and Bart Preneel. On the XOR of multiple random permutations. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 619–634. Springer, 2015.

[MPN10] Avradip Mandal, Jacques Patarin, and Valérie Nachef. Indifferentiability beyond the birthday bound for the xor of two public random permutations. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010 - 11th International Conference on Cryptology in India, Hyderabad, India, December 12-15, 2010. Proceedings*, volume 6498 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2010.

[MPS16] Dustin Moody, Souradyuti Paul, and Daniel Smith-Tone. Improved indifferentiability security bound for the JH mode. *Des. Codes Cryptography*, 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.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[Pat08] Jacques Patarin. The "Coefficients H" technique. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 2008.

[PGV93] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer, 1993.

[Rab78] Michael O. Rabin. Digitalized signatures. In *Foundations of Secure Computation*, pages 155–166, New York, 1978. Academic Press.

[Riv92] Ronald Rivest. The MD5 message-digest algorithm. Request for Comments (RFC) 1321, April 1992. http://tools.ietf.org/html/rfc1321.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2004.

[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.

[Sar07] Palash Sarkar. Construction of universal one-way hash functions: Tree hashing revisited. *Discrete Applied Mathematics*, 155(16):2174–2180, 2007.

[SHA08] National Institute of Standards and Technology. Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-3, October 2008.

[SS01] Palash Sarkar and Paul J. Schellenberg. A parallel algorithm for extending cryptographic hash functions. In C. Pandu Rangan and Cunsheng Ding, editors, *Progress in Cryptology - INDOCRYPT 2001, Second International Conference on Cryptology in India, Chennai, India, December 16-20, 2001, Proceedings*, volume 2247 of *Lecture Notes in Computer Science*, pages 40–49. Springer, 2001.

[Sta09] Martijn Stam. Blockcipher-based hashing revisited. In Dunkelman [Dun09], pages 67–83.