

# Generic Indifferentiability Proofs of Hash Designs

Marion Daubignard  
University of Grenoble (UJF),  
CNRS, Verimag, FRANCE  
DGA-MI, Rennes, FRANCE  
msdaubignard@gmail.com

Pierre-Alain Fouque  
École normale supérieure,  
CNRS, INRIA Rennes, FRANCE  
pierre-alain.fouque@ens.fr

Yassine Lakhnech  
University of Grenoble (UJF),  
CNRS, Verimag, FRANCE  
yassine.lakhnech@imag.fr

**Abstract**—In this paper, we propose a formal analysis of domain extenders for hash functions in the indifferentiability framework. We define a general model for domain extenders and provide a unified proof of their security in the form of a generic reduction theorem. Our general model for domain extenders captures many iterated constructions such as domain extenders, modes of operation of symmetric cryptography such as CBC-MAC or blockciphers based on Feistel networks. Its proof has been carried out using the Computational Indistinguishability Logic of Barthe *et al.*. The theorem can help designers of hash functions justifying the security of their constructions: they only need to bound the probability of well-defined events. Our model allows to consider many SHA-3 finalists and is instantiated on two well-known constructions, namely Chop-MD and Sponge. Finally, the indifferentiability bounds which we prove are convincing since they match previous proofs.

## I. INTRODUCTION

**Motivation.** Hash functions are the swiss army knife of cryptographers. They are used to generate unique identifiers in hash-and-sign signatures, as one-way functions for one-time-password, to break the structure of the input in key derivation functions and also for authentications...

Recently, cryptographers have looked carefully at the security of these functions after the breakthrough discovery of differential attacks by Wang *et al.* on the MD4 family. Moreover, weaknesses in the Merkle-Damgard (MD) mode of operation [Mer89], [Dam89], standardly used to design hash functions, have been exposed (see [Jou04], [KS05], [KK06]). In response to all these attacks, the NIST decided to launch a competition in order to select the new standard SHA-3.

Hash functions are public functions that map arbitrarily long bitstrings to fixed-length bitstrings. To construct a hash function, cryptographers first build a fixed input-length function  $f : \{0, 1\}^m \rightarrow \{0, 1\}^n$ , where  $n < m$ , called a compression function. They then increase the input domain to  $\{0, 1\}^*$  using a

domain extender, a.k.a. a mode of operation. This latter defines the way the compression function  $f$  is iterated to obtain a hash construction  $H^f$ .

Some security goals have been defined for  $H^f$  such as collision, second-preimage or preimage resistance, but they are not sufficient to cover the needs of all applications of hash functions. Indeed, in security proofs of some cryptographic constructions using hash functions, cryptographers would like to model these latter as public random functions: this is the random oracle model (ROM) of [BR93]. Controversial [CGH04], the ROM is now superseded by the *standard* model. However, many practical schemes are only proved in the ROM. To increase confidence in these proofs, Coron *et al.* [CDMP05] propose to take into account the structure of the domain extender by showing a novel security criterion for them: indifferentiability from a random oracle. This notion is strongly inspired from the *indifferentiability* framework proposed by Maurer *et al.* [Mau02], [MRH04].

Indifferentiability captures the absence of generic adversaries against the domain extender. Informally, it means that there exists a convincing way to mimic the pair hash construction and compression function using a random oracle and a simulator. A nice composition property ensues: if the compression function behaves as a small random oracle, then the hash construction as a whole is safely modeled by a monolithic random oracle within a bigger cryptographic construction. Therefore, the domain extender does not introduce any structural weakness in the whole cryptographic construction. Consequently, and though it needs to be used with caution [RSS11], indifferentiability from a random oracle proves to be a relevant security criterion. It is worth noticing that it does capture the aforementioned weaknesses in the MD design, which is shown differentiable from a random oracle. Investigating the indifferentiability of new modes of operations proposed during the SHA-3 competition is an important issue.

To prove indistinguishability results, cryptographers need to show that two idealized systems are indistinguishable. The first system captures the real setting: it is built from a random function in place of the compression function and a domain-extender which can query the former to compute its result. The second system is an idealized version of the first one, where the construction is replaced by a big random oracle  $RO$  and the compression function by a simulator  $S$ . To simulate the random function it replaces,  $S$  has access to  $RO$ . The main difficulty of the proof is that  $S$  should answer queries *consistently* while it cannot see the queries/answers of the adversary to  $RO$ .

**Formal Model.** As security proofs in general, indistinguishability proofs present a lot of technicalities and are hard to verify. Moreover, the quantifier order in the definition of security yields subtleties [RSS11] that recall those raised in universal composability frameworks as [Can01], [BPW03]. Yet, as cornerstones of many other cryptographic designs, hash constructions need strong and trusted security foundations. To achieve this, we believe that confidence in indistinguishability proofs can strongly benefit from a formal treatment.

**Our Contribution.** We propose to profit from the computational indistinguishability logic CIL proposed in [BDKL10], whose rules are formalized in Coq. We prove a reduction theorem dedicated to indistinguishability of hash constructions in this framework. Our formalization of proofs thus contributes to bridging the gap towards their automated verification. We extend the semantics of [BDKL10] with the notion of overlayers, that captures many iterated constructions: domain extenders, modes of operation of symmetric cryptography such as CBC-MAC or blockciphers based on Feistel networks. Our definition allows us to take into account many domain extenders used in the SHA-3 competition such as JH [Wu11], Keccak [BDPA11], Skein [FLS<sup>+</sup>10] and BLAKE [AHMP10], the EMD transform [BR06], HMAC and NMAC modes [BCK96]. It generalizes the generic domain extenders proposed by Bhattacharyya et al. in [BMN09] since it allows post-processing and multiple inner-primitives. Then, we describe a generic simulator and prove directly in CIL a meta-theorem allowing to bound the indistinguishability of constructions from random oracles. To instantiate this theorem, users have to bound some inconsistency events that can happen in the simulator.

We show on two examples that the bounds provided by our theorem are convincing: for the

ChopMD solution, we achieve the same result as Maurer and Tessaro in [MT07] in case of prefix-free padding and a better bound than that of Chang and Nandi in [CN08] in the general case. Finally, the application of our result on the sponge construction (underlying the Keccak design) highlights the lack of an additional term in the bound provided by Bertoni *et al.* in [BDPA08], as was anticipated but not justified by Bresson *et al.* in [BCCM<sup>+</sup>08].

**Outline.** The next section briefly provides background on CIL, while the generic definition to capture hash designs and a formal definition of their indistinguishability from a random oracle appear in section III. In section IV, we detail the construction of a generic simulator involved in our result. In V, we define characteristic graphs to capture events of bad simulation and state the reduction theorem. Finally, examples of application appear in section VI.

**Notations.** LISTS. Given a set  $A$ , we denote by  $A^*$  (resp.  $A^+$ ) the set of finite lists with elements in  $A$  (resp. non-empty finite lists). The empty list is denoted by  $[\ ]$ .  $[L]_{i=m}^n$  denotes the sublist of  $L$  containing elements of  $L$  from the  $m$ -th to the  $n$ -th position. The append to the right of an element  $a \in A$  to a list  $L \in A^*$  is denoted by  $L : a$ . The selective append of  $a$  to  $L$ , denoted by  $L.a$ , is defined by  $L.a = L$ , if  $a \in L$  and  $L.a = L : a$ , otherwise. Given an indexed set  $A = (a_i)_{i \in \mathbb{N}}$ , and an index set  $I$ ,  $[a_i]_{i \in I}$  denotes the list of elements  $a_i$  for  $i \in I$ . Moreover,  $\text{dom}(L)$  denotes the set of first components of elements of  $L$ , while  $L(a)$  is the set of elements of  $L$  with first component  $a$ .

STRINGS. Given a bitstring  $w$ ,  $|w|$  denotes the length of  $w$ . For  $s \leq |w|$ ,  $\text{Last}_s(w)$  and  $\text{First}_s(w)$  denote the suffix of  $w$ , respectively its prefix, of length  $s$ . For  $1 \leq m \leq n \leq |w|$ ,  $w[m, n]$  denotes the substring of  $w$  starting with its  $m$ -th bit and ending its  $n$ -th bit. The concatenation of two bitstrings  $x$  and  $y$  is denoted by  $x||y$ . A string of length 0 is denoted by  $\lambda$ .

MISCELLANEOUS.  $\lceil x \rceil$  denotes the ceiling of  $x$ . We use  $\mathbf{1}$  to denote the unit type. Given a finite set  $A$ , distributions on  $A$  are denoted  $\mathcal{D}(A)$ ; given  $a \in A$  and  $d \in \mathcal{D}(A)$ , the probability of sampling  $a$  according to  $d$  is denoted by  $\Pr[d = a]$ . The uniform distribution over  $A$  is denoted by  $\mathcal{U}(A)$ . We write  $\_$  to stand for elements we do not need to name.

## II. ORACLE SYSTEMS AND ADVERSARIES

### A. Formalization of Oracles and Adversaries

Following [BDKL10], we use oracle systems to describe cryptographic schemes. Informally, an oracle system is composed of a finite list of stateful oracles that can be queried by an adversary. Each oracle has an implementation which is a probabilistic algorithm that may have access to other oracles. A query call to an oracle yields an output, called *answer*, and may modify the memory as a side effect.

Let  $n$  be a positive natural number, for each  $i = 1, \dots, n$ ,  $M_i$  be a countable set of memories (states) and  $M_{\mathbb{O}} = M_1 \times \dots \times M_n$ . Furthermore, let  $o_1, \dots, o_n$  be disjoint names of oracles that operate on  $M_{\mathbb{O}}$ . An *implementation* of  $o_i$  is a probabilistic algorithm  $\text{Imp}(o_i)^{o_{j_1}, \dots, o_{j_i}}$  which:

- has oracle access to the implementations of  $o_{j_1}, \dots, o_{j_i}$ ;
- takes as input a query in a finite *query domain*  $\text{In}(o_i)$  and a tuple of memories in  $\prod_{k \in [1..n] - \{j_1, \dots, j_i\}} M_k$ ;
- yields an answer in a finite *answer domain*  $\text{Out}(o_i)$  and updates the memories in  $M_k$ , for  $k \in [1..n] - \{j_1, \dots, j_i\}$ . We stress that oracle  $o_i$  thus shares memories with oracle  $o_k$  for such indices  $k$ .

An oracle implementation  $\text{Imp}(o_i)^{o_{j_1}, \dots, o_{j_i}}$  is called  $(\alpha, t)$ -*bounded*, where  $\alpha : \{o_{j_1}, \dots, o_{j_i}\} \rightarrow \mathbb{N}$  and  $t \in \mathbb{N}$ , if every execution of  $\text{Imp}(o_i)^{o_{j_1}, \dots, o_{j_i}}$  makes at most  $\alpha(o_{j_m})$  calls to  $o_{j_m}$ , for  $j_m \in \{j_1, \dots, j_i\}$ , and takes at most time  $t$ .

**Definition II.1.** An oracle system  $\mathbb{O}$  is given by:

- a finite set  $N_{\mathbb{O}} = \{o_1, \dots, o_n\}$  of oracles that operate on  $M_{\mathbb{O}}$  equipped with their implementations  $\text{Imp}(o_i)^{o_{j_1}, \dots, o_{j_i}}$ ;
- distinguished oracles  $o_{\text{I}} \in N_{\mathbb{O}}$ , the initialization oracle, and  $o_{\text{F}} \in N_{\mathbb{O}}$ , the finalization oracle, with  $\text{In}(o_{\text{F}}) = \{\text{true}, \text{false}\}$  and  $\text{Out}(o_{\text{F}}) = \mathbf{1}$ ;
- an initial memory  $\bar{m}_{\mathbb{O}} \in M_{\mathbb{O}}$ .

**EXAMPLE II.1.** As an example of oracle system, we provide the system  $\mathbb{O}$ , consisting of two oracles named  $\mathcal{F}$  and  $\mathcal{K}$ . Informally,  $\mathcal{F}$  is a random function on bitstrings of length  $n$  and oracle  $\mathcal{K}$  is  $\mathcal{F} \circ \mathcal{F}$ . A rigorous description of the implementations of  $\mathcal{F}$  and  $\mathcal{K}$  is given in Figure 1, where  $L_{\mathcal{F}}$  and  $L_{\mathcal{K}}$  are finite mappings that build the memory. The initial state is composed of the empty mappings. Initialization oracle and finalization oracle are idle.

Oracle systems  $\mathbb{O}$  and  $\mathbb{O}'$  are called *compatible*, if they have the same set of oracle names, with same

<p>Oracle <math>\mathcal{F}</math></p> <pre> Imp<math>_{\mathbb{O}}</math>(<math>\mathcal{F}</math>)(<math>q, L_{\mathcal{F}}</math>) = if <math>q \in \text{dom}(L_{\mathcal{F}})</math> then   return <math>L_{\mathcal{F}}(q)</math> else   let <math>a \leftarrow \mathcal{U}(\{0, 1\}^n)</math> in   <math>L_{\mathcal{F}} := L_{\mathcal{F}}.(q, a)</math>;   return <math>a</math> endif </pre>	<p>Oracle <math>\mathcal{K}</math></p> <pre> Imp<math>_{\mathbb{O}}</math>(<math>\mathcal{K}</math>)<math>^{\mathcal{F}}</math>(<math>q, L_{\mathcal{K}}</math>) = if <math>q \in \text{dom}(L_{\mathcal{K}})</math> then   return <math>L_{\mathcal{K}}(q)</math> else   let <math>a \leftarrow \mathcal{F}(q)</math> in   let <math>b \leftarrow \mathcal{F}(a)</math> in   <math>L_{\mathcal{K}} := L_{\mathcal{K}}.(q, b)</math>;   return <math>b</math> endif </pre>
--	--

Figure 1. Implementations of oracles  $\mathcal{F}$  and  $\mathcal{K}$

query and answer domains. Compatible systems can however differ in the oracle implementations and memories. We therefore use the notation  $\text{Imp}_{\mathbb{O}}(o_i)$  (resp.  $\text{Imp}_{\mathbb{O}'}(o_i)$ ) to refer to the implementation of  $o_i$  in  $\mathbb{O}$  (resp.  $\mathbb{O}'$ ).

With an oracle system  $\mathbb{O}$ , adversaries interact by making queries and receiving answers. An interaction step produces an *exchange* (for an oracle system  $\mathbb{O}$ ), which is a triple  $(o, q, a)$  where  $o \in N_{\mathbb{O}}$ ,  $q \in \text{In}(o)$  and  $a \in \text{Out}(o)$ . Let  $\text{Xch}$  denote the set of exchanges. *Final exchanges* are of the form  $(o_{\text{F}}, -, -)$ , i.e. queries to the finalization oracle. The set of final exchanges is denoted by  $\text{Xch}_{\text{F}}$ . The sets  $\text{Que}$  of queries and  $\text{Ans}$  of answers are, respectively, defined by  $\text{Que} = \{(o, q) \mid (o, q, a) \in \text{Xch}\}$  and  $\text{Ans} = \{(o, a) \mid (o, q, a) \in \text{Xch}\}$ , while for a particular oracle  $o_i$ ,  $\text{Que}(o_i)$  (resp.  $\text{Ans}(o_i)$ ,  $\text{Xch}(o_i)$ ) only contains elements of  $\text{Que}$  (resp.  $\text{Ans}$ ,  $\text{Xch}$ ) starting with  $o_i$ .

**Definition II.2.** An  $\mathbb{O}$ -adversary  $\mathbb{A} = (\mathbb{A}, \mathbb{A}_{\downarrow})$  is given by a countable set  $M_{\mathbb{A}}$  of adversary memories, an initial memory  $\bar{m}_{\mathbb{A}} \in M_{\mathbb{A}}$  and functions for querying and updating:

$$\begin{aligned} \mathbb{A} & : M_{\mathbb{A}} \longrightarrow \mathcal{D}(\text{Que} \times M_{\mathbb{A}}) \\ \mathbb{A}_{\downarrow} & : \text{Xch} \times M_{\mathbb{A}} \rightarrow \mathcal{D}(M_{\mathbb{A}}) \end{aligned}$$

Informally, the interaction between an oracle system and an adversary starts from the initial memory  $(\bar{m}_{\mathbb{A}}, \bar{m}_{\mathbb{O}})$ . Using  $\mathbb{A}$ ,  $\mathbb{A}$  computes a query to  $\mathbb{O}$  and updates its memory. Upon receiving a query,  $\mathbb{O}$  updates its memory and replies to  $\mathbb{A}$ , which in turn updates its memory using  $\mathbb{A}_{\downarrow}$ . This goes on until  $\mathbb{A}$  calls the finalization oracle. We formalize this interaction as the execution of a transition system, defined below.

**Definition II.3.** A transition system  $\mathcal{S}$  consists of:

- a (countable non-empty) set  $M$  of memories (states), with a distinguished initial memory  $\bar{m}$ ;

- a set  $\Sigma$  of actions, with a distinguished subset  $\Sigma_F$  of finalization actions;
- a (partial probabilistic) transition function  $\text{st} : \mathbb{M} \rightarrow \mathcal{D}(\Sigma \times \mathbb{M})$ .

A partial execution sequence of  $\mathcal{S}$  is a sequence  $\eta$  of the form  $m_0 \xrightarrow{\text{act}_1} m_1 \xrightarrow{\text{act}_2} \dots \xrightarrow{\text{act}_k} m_k$  such that  $m_0 = \bar{m}$ ,  $\text{act}_i \in \Sigma$ ,  $m_{i-1}, m_i \in \mathbb{M}$ , and  $\Pr[\text{st}(m_{i-1}) = (\text{act}_i, m_i)] > 0$ , for  $i = 1, \dots, k$ . If  $k = 1$ , then  $\eta$  is a step. If  $\text{act}_k \in \Sigma_F$  or  $m_k \notin \text{dom}(\text{st})$ , then  $\eta$  is an execution sequence of length  $k$ . A probabilistic transition system  $\mathcal{S}$  induces a sub-distribution on executions, denoted  $\mathcal{S}$ , such that the probability  $\Pr[\mathcal{S} = \eta]$  of a finite execution sequence  $\eta$  is  $\prod_{i=1}^k \Pr[\text{st}(m_{i-1}) = (\text{act}_i, m_i)]$ .

A transition system is of height  $k \in \mathbb{N}$  if all its executions have length at most  $k$ ; in this case,  $\mathcal{S}$  is a distribution.

**Definition II.4.** Let  $\mathbb{O}$  be an oracle system and  $\mathbb{A}$  be an  $\mathbb{O}$ -adversary. The composition  $\mathbb{A} \mid \mathbb{O}$  is a transition system such that  $\mathbb{M} = \mathbb{M}_{\mathbb{A}} \times \mathbb{M}_{\mathbb{O}}$ , the initial memory is  $(\bar{m}_{\mathbb{A}}, \bar{m}_{\mathbb{O}})$ , the set of actions is  $\Sigma = \text{Xch}$ , and  $\Sigma_F = \text{Xch}_F$ , and

$$\text{st}_{\mathbb{A} \mid \mathbb{O}}(m_{\mathbb{A}}, m_{\mathbb{O}}) \stackrel{\text{def}}{=} \begin{array}{l} \text{let } ((o, q), m'_{\mathbb{A}}) \leftarrow \text{A}(m_{\mathbb{A}}) \text{ in} \\ \text{let } (a, m'_{\mathbb{O}}) \leftarrow \text{Imp}_{\mathbb{O}}(o)(q, m_{\mathbb{O}}) \text{ in} \\ \text{let } m''_{\mathbb{A}} \leftarrow \text{A}_{\downarrow}((o, q, a), m'_{\mathbb{A}}) \text{ in} \\ \text{return } ((o, q, a), (m''_{\mathbb{A}}, m'_{\mathbb{O}})) \end{array}$$

Let  $k : \mathbb{N}_{\mathbb{O}} \rightarrow \mathbb{N}$ . An adversary is called  $k$ -bounded, if for every  $o \in \mathbb{N}_{\mathbb{O}}$ , the number of queries to  $o$  in every execution of  $\mathbb{A} \mid \mathbb{O}$  is not greater than  $k(o)$ . An adversary is called bounded, if it is  $k$ -bounded for some  $k$ . Thus,  $k$  bounds the number of oracle calls that can be performed by an adversary. To meaningfully state security properties of oracle systems, we also bound the adversary's global running time. Therefore, we consider bounds of the form  $(k, t) \in (\mathbb{N}_{\mathbb{O}} \rightarrow \mathbb{N}) \times \mathbb{N}$  and talk about  $(k, t)$ -bounded adversaries, whose set we denote  $\text{Adv}(k, t)$ .

Security properties abstract away from the state of adversaries, and are modeled using traces. Informally, a trace  $\tau$  is an execution sequence  $\eta$  from which the adversary memories have been erased.

**Definition II.5.** Let  $\mathbb{O}$  be an oracle system.

- A partial trace is a sequence  $\tau$  of the form  $m_0 \xrightarrow{\text{act}_1} m_1 \xrightarrow{\text{act}_2} \dots \xrightarrow{\text{act}_k} m_k$ , where  $m_0, \dots, m_k \in \mathbb{M}_{\mathbb{O}}$  and  $\text{act}_1, \dots, \text{act}_k \in \text{Xch}$  such that for  $i = 1, \dots, k$  and  $\text{act}_i = (o_i, q_i, a_i)$ ,  $\Pr[\text{Imp}_{\mathbb{O}}(o_i)(q_i, m_{i-1}) = (a_i, m_i)] > 0$ . A trace is a partial trace  $\tau$  such that  $m_0 = \bar{m}_{\mathbb{O}}$  and  $\text{act}_k = (o_F, -, -)$ .

- An  $\mathbb{O}$ -event  $E$  is a predicate over  $\mathbb{O}$ -traces.

The probability of an event is derived directly from the definition of  $\mathbb{A} \mid \mathbb{O}$ . Indeed, each execution sequence  $\eta$  induces a trace  $\mathcal{T}(\eta)$  simply by erasing the adversary memory at each step. Consequently, for each trace  $\tau$ , we define the set  $\mathcal{T}^{-1}(\tau)$  of execution sequences that are erased to  $\tau$ , and for every event  $E$  the probability  $\Pr(\mathbb{A} \mid \mathbb{O} : E) = \Pr(\mathbb{A} \mid \mathbb{O} : \mathcal{T}^{-1}(E))$ .

## B. The Logic

The logic CIL [BDKL10], [?], [?] features around twenty sound rules to reason on oracle systems. It is built to establish two kinds of judgments classically used to express concrete security notions and carry out their proofs. Namely, judgments capture that a function of the adversarial resources bounds the indistinguishability between two oracle systems or the probability that an event happens in an oracle system.

Rules of the logic formalize reasoning patterns that generally appear in cryptographic proofs. Rather than simply mimic frequent steps of proofs carried out in the game-based methodology [Sho04], CIL rules stem from classic programming language and concurrent systems proof techniques, such as bisimulation relations, embedding in a context or determinization. Indeed, the formalism is far more resembling descriptions of security experiments and code-based transformations proposed in [?] than game-based proofs. The fact that it allows to reason on oracle systems and abstracts away adversaries is in particular worth noticing, though it is totally possible to code games as systems. The main focus of the proof system is to build transitions between systems relying on provably correct equalities (or inequalities) between probabilities. Compared to a game-based pencil-and-paper proof, it often looks like taking smaller steps and the proof outline might get modified to get a full proof within the proof system. Indeed, some relations between probabilities seem quite obvious but are not so easily proven. The whole purpose of the proof system is to only allow correct transitions. Hence, such situations or sufficient conditions such as bisimulation between programs have been identified and formalized as rules of the proof system. Furthermore, the proof system comprises interface rules allowing for input of results obtained by external reasoning. We elaborate a little on bisimulations and determinization concepts, since they are used in the proof of the result presented in this paper.

Bisimulation relations appear as a key notion to formally link two oracle systems. They are indeed

equivalence relations on the memories of the systems and provide an explicit relation between probabilities of classes of partial executions in both systems. Therefore, events compatible with the partition yielded by the equivalence relation happen with equal probability in two bisimilar systems. In practice, unconditional bisimulation may be insufficient. Therefore, CIL uses the notion of bisimulation up to a condition.

Oracle system determinization allows to group states and thus partial executions that correspond to the same exchange sequence. As a result, where bisimulation relations fail to formalize anticipation or delay of sampling of some values from one oracle call to another one, determinization successfully capture that such modifications yield behaviors indistinguishable by an adversary.

### III. OVERLAYERS AND SECURITY CRITERION

#### A. Overlayers

Many cryptographic functions are built by iterating a set of *inner primitives*: hash functions, CBC-MAC, blockciphers... Inner-primitives take as input bitstrings of fixed length. As hash functions take as input longer bitstrings, they are based on so-called *domain extenders* [Mer89], [Dam90]. These specify how the input message is split into blocks that can be treated by the inner-primitives.

In [BMN09], a formal definition for domain extenders is presented. Though applicable to several known constructions, this definition does not capture constructions that include a post-processing function. Post-processing is used to compute the global hash result out of the multiple inner-primitive outputs. We emphasize that post-processing does not perform calls to inner-primitives and that these take place *before* post-processing. For example, the ChopMD [CDMP05] and the sponge construction [BDPA07] include a post-processing function.

**EXAMPLE III.1.** *The sponge construction relies on an inner primitive  $\mathcal{F}$ , which is a random function from  $\{0, 1\}^{r+c}$  into  $\{0, 1\}^{r+c}$ , where  $r$  is the length of blocks parsed during preprocessing. For an input  $x$ , the sponge construction needs  $l = m + k - 1$  iterations, where  $m$  is the number of  $r$ -blocks of the input and  $k$  is the number of  $r$ -blocks desired in the output. The inner primitive  $\mathcal{F}$  is called at each iteration. The output size is parameterized by  $K \in \mathbb{N}$ . While the general design deals with any possible  $K$ , in the sequel we assume for sake of simplicity that  $K = kr$ , and refer the readers to [BDPA07] for*

*more details. The construction comprises two phases. During the absorbing phase, the input is padded using an injective, easily computable and invertible function  $\text{Pad}_{sp}$  that yields a bitstring  $x_1 || \dots || x_m$  of length  $m * r$ . Then, a bitwise xor operation is applied to  $x_j$  and the first  $r$ -block of the previous answer of  $\mathcal{F}$  to compute the next query to  $\mathcal{F}$ . During the squeezing phase,  $\mathcal{F}$  is queried  $k - 1$  more times to get a collection of answers  $(a^{m+1}, \dots, a^l)$ . The final output is then obtained by concatenation of the first  $r$  bits of each  $a^j$ :  $\text{First}_r(a^m) || \dots || \text{First}_r(a^l)$ . The implementation is provided in Figure 2.*

```

In(Sponge) =  $\{0, 1\}^{\leq 2^{64}}$ , Out(Sponge) =  $\{0, 1\}^K$ 
Imp(Sponge) $^{\mathcal{F}}$ ( $x, L_{sp}$ ) =
if  $x \in \text{dom}(L_{sp})$  then
  return  $L_{sp}(x)$ 
else
   $l := \text{init}_{sp}(x)$ ;
   $w := \text{Pad}_{sp}(x)$ ;
   $m := \lfloor \frac{|w|}{r} \rfloor$ ;
   $(x_1, \dots, x_m) := (w[1, r], \dots, w[r(m-1) + 1, rm])$ ;
   $a^0 := 0^{r+c}$ ;
  for  $j = 1$  to  $m$  do
     $q^j := (x_j || 0^c) \oplus a^{j-1}$ ;
    let  $a^j \leftarrow \mathcal{F}(q^j)$  in
     $Q := Q : (q^j, a^j)$ ;
  endfor
  for  $j = m + 1$  to  $l$  do
     $q^j := a^{j-1}$ ;
    let  $a^j \leftarrow \mathcal{F}(q^j)$  in
     $Q := Q : (q^j, a^j)$ ;
  endfor
   $a^f := \text{First}_r(a^m) || \dots || \text{First}_r(a^l)$ ;
   $L_{sp} := L_{sp} \cdot (x, a^f, Q)$ ;
  return  $a^f$ 
endif
where  $\text{init}_{sp}(x) = m + k - 1$ , with  $m = |w|/r$  and
 $k * r$  the length of the output of the sponge
construction.

```

Figure 2. Sponge Implementation

Besides the limitation mentioned previously, we notice that the case of multiple inner-primitives is not dealt with in [BMN09]. For instance, the Grøstl [GKM<sup>+</sup>11] construction is out of scope of this definition. Hence, we introduce a new definition based on the notion of *overlayer* that allows to capture all hash functions based on domain extenders we are aware of. A hash design can then be described as an overlayer applied to an oracle system, where this latter defines the inner-primitives. Informally, an overlayer consists in:

- 1) a function  $\Theta$  that describes how an input

- $x$  is padded and cut into  $\text{init}(x)$  blocks  $\theta_1(x), \dots, \theta_{\text{init}(x)}(x)$ , with  $\text{init}(x) \in [0, \mathfrak{L}]$ ;
- 2) a finite sequence  $[o^1, \dots, o^{\mathfrak{L}}]$  of inner-primitives such that the sequence of calls to inner-primitives generated during the computation of the hash of any input  $x$  is a prefix of a  $[o^1, \dots, o^{\mathfrak{L}}]$ . All known hash designs are based on a fixed sequence  $[o^1, \dots, o^{\mathfrak{L}}]$ ;
  - 3) a function  $\text{piv}$  with  $\text{piv}(x) \in [1, \text{init}(x)]$  and such that the output of the post-processing does not depend on calls performed to the inner-primitives before  $\text{piv}(x)$ . We call  $\text{piv}(x)$  the *pivot index*.
  - 4) functions  $H_1, \dots, H_{\mathfrak{L}}$  that compute the input queries to the inner-primitives;
  - 5) a post-processing function  $H_{\text{post}}$ .

**EXAMPLE III.2.** *Let us consider the sponge construction to comment some of the choices in the notion of overlayer. Assume that the padding yields a message of  $m$  blocks of length  $r$  and that the hash is of length  $kr$ . The sponge has a two-phase structure, and the post-processing function depends on all calls in the squeezing phase. The absorbing phase consists of  $m$  calls to the inner-primitive  $\mathcal{F}$ , while there are  $k-1$  calls performed during the squeezing case. For this construction,  $[o^1, \dots, o^{\mathfrak{L}}]$  is then a list where  $\mathcal{F}$  is repeated  $\mathfrak{L}$  times. Any index in the absorbing phase is a valid pivot. In practice, it is better to choose the latest possible index for the pivot since it yields a better simulator, and hence, a tighter indistinguishability bound. For sponge, this is the last index of the absorbing phase.*

Henceforth, we consider hash constructions that take as input bitstrings in a finite set  $\text{In}_{\mathcal{H}}$  and produce hash values in a finite set  $\text{Out}_{\mathcal{H}}$ , where  $\text{In}_{\mathcal{H}}$  and  $\text{Out}_{\mathcal{H}}$  are arbitrary finite sets. We also consider fixed an arbitrary oracle system  $\mathbb{O}$  with oracles in  $\mathbb{N}_{\mathbb{O}}$ .

**Definition III.1.** *An  $\mathbb{O}$ -overlayer  $h$  is a tuple  $([o^1, \dots, o^{\mathfrak{L}}], \Theta, (H_j)_{j \in \{1.. \mathfrak{L}\}}, \text{piv}, H_{\text{post}})$ , where:*

- $[o^1, \dots, o^{\mathfrak{L}}]$  is a sequence of oracles in  $\mathbb{N}_{\mathbb{O}}$ .
- $\Theta : \text{In}_{\mathcal{H}} \rightarrow (\{0, 1\}^{\leq r})^+$  with  $\Theta(x) = (\theta_1(x), \dots, \theta_{\text{init}(x)}(x))$  the input transformation and  $r > 0$  the block length.
- $\text{piv} : \text{In}_{\mathcal{H}} \rightarrow [1, \mathfrak{L}]$  outputs a pivot index such that  $\text{piv}(x) \leq \text{init}(x)$ . For simplicity, we require that there is an oracle  $o_{\text{piv}}$ , that we call the pivot oracle, such that for any  $x$ ,  $o^{\text{piv}(x)} = o_{\text{piv}}$ .
- functions  $H_1 : \{0, 1\}^{\leq r} \rightarrow \text{In}(o^1)$  and  $H_j : \{0, 1\}^{\leq r} \times \text{Xch} \rightarrow \text{In}(o^j)$  for  $j = 2, \dots, \mathfrak{L}$ .
- $H_{\text{post}} : \text{In}_{\mathcal{H}} \times \text{Out}_{o_{\text{piv}}} \times \text{Xch}^* \rightarrow \text{Out}_{\mathcal{H}}$  is the

*post-processing function.*

Thus,  $H_{\text{post}}(x, y, Q)$  is the hash of  $x$ , when  $Q = (o^k, q^k, a^k)_{k \in [1, \text{init}(x)]}$  is the list of exchanges generated by the  $H_j$  functions for  $x$  and  $y = a^{\text{piv}(x)}$ .

**EXAMPLE III.3.** *Consider again the sponge construction. Then, the pivot is  $\text{piv}_{sp}(x) = m$ ,  $H_j(x_j, (\mathcal{F}, q^{j-1}, a^{j-1})) = (x_j || 0^c) \oplus a^{j-1}$  for  $j = 1..m$ , and  $H_{\text{post}}(x, a^{\text{piv}_{sp}(x)}, [(\mathcal{F}, q^j, a^j)]_{j > \text{piv}_{sp}(x)}) = \text{First}_r(a^{\text{piv}_{sp}(x)}) || \dots || \text{First}_r(a^{\text{init}_{sp}(x)})$ . Moreover,  $\theta_j(x) = x_j$  for  $j \in [1..m]$ , and  $\theta_j(x) = 0^r$  for  $j \in [m+1, \text{init}(x)_{sp}]$ .*

We require overlayers to satisfy the following conditions that are met by all hash designs we know about:

- 1) the function  $\Theta$  is injective;
- 2)  $H_{\text{post}}$  only depends on pivot and post-pivot queries. Formally, for all lists  $Q$  and  $Q'$  and all  $x, y$ , if  $[Q]_{k > \text{piv}(x)} = [Q']_{k > \text{piv}(x)}$  then  $H_{\text{post}}(x, y, Q) = H_{\text{post}}(x, y, Q')$ . Therefore, we sometimes write  $H_{\text{post}}(x, a^{\text{piv}(x)}, [Q]_{k > \text{piv}(x)})$  instead of  $H_{\text{post}}(x, a^{\text{piv}(x)}, Q)$ .

**Definition III.2.** *The composition of an  $\mathbb{O}$ -overlayer  $h$  with  $\mathbb{O}$  defines an oracle system which contains the oracles of  $\mathbb{O}$  augmented with the overlayer oracle  $\mathcal{H}$  given by:*

- the memory  $L_{\mathcal{H}}$  of oracle  $\mathcal{H}$  is a mapping from  $\text{In}_{\mathcal{H}}$  to  $\text{Out}_{\mathcal{H}} \times \text{Xch}^*$ ; its initial value is the empty mapping.
- The implementation of oracle  $\mathcal{H}$  is given in figure 3.

## B. Security Definition of Hash Constructions

A widely accepted approach for proving properties of hash constructions consists in assuming idealized inner-primitives (e.g. random functions) and proving that hash constructions are indistinguishable from a random function (see [CDMP05]), which is commonly named indistinguishability from a random oracle. In our framework, this notion is formalized as a comparison between two oracle systems: one where the real hash mode is used as an overlayer of an idealized (inner) oracle system, and another where the hash construction is idealized as a random oracle and a simulator makes up for the inner system. We thus introduce a notation for idealization by uniform functions before writing our formalization for indistinguishability.

```

Imp( $\mathcal{H}$ ) $^{o^1, \dots, o^l}$ ( $x, L_{\mathcal{H}}$ ) = if  $x \in \text{dom}(L_{\mathcal{H}})$  then
  return  $L_{\mathcal{H}}(x)$ 
else
   $l := \text{init}(x)$ ;
   $(x_1, \dots, x_l) := \Theta(x)$ ;
   $q^1 := H_1(x_1)$ ;
  let  $a^1 \leftarrow o^1(q^1)$  in
   $Q := [(o^1, q^1, a^1)]$ ;
  for  $j = 2$  to  $l$  do
     $q^j := H_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$ ;
    let  $a^j \leftarrow \text{Imp}_{\mathbb{O}}(o^j)(q^j)$  in
     $Q := Q : (o^j, q^j, a^j)$ ;
  endfor
   $a^f := H_{\text{post}}(x, a^{\text{piv}(x)}, [Q]_{k > \text{piv}(x)})$ ;
   $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$ ;
  return  $a^f$ 
endif

```

Figure 3. Implementation of the  $\mathcal{H}$  Oracle

We denote by  $\mathcal{RO}(\mathcal{H})$  the oracle implemented as a random function on  $\text{Out}_{\mathcal{H}}$  using a list (mapping)  $L_{\mathcal{H}}$  as follows:

```

if  $x \in \text{dom}(L_{\mathcal{H}})$  then return  $L_{\mathcal{H}}(x)$ 
else let  $y \leftarrow \mathcal{U}(\text{Out}_{\mathcal{H}})$  in  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, y)$ ;
  return  $y$  endif

```

We naturally lift this definition to an oracle system, by writing  $\mathcal{RO}(\mathbb{O})$ .

**Definition III.3.** Consider an oracle  $\mathbb{O}$  and an  $\mathbb{O}$ -overlayer  $h$ . The system  $(\mathcal{H}^{\mathbb{O}}, \mathbb{O})$  defined by the composition of  $h$  with  $\mathbb{O}$  is said to be  $\epsilon$ -indifferentiable from its idealization  $\mathcal{RO}(\mathcal{H})$  with a  $(k_s, t_s)$ -simulator, if there is an oracle set  $\mathbb{S}^{\mathcal{RO}(\mathcal{H})}$  that is  $(k_s, t_s)$ -bounded and such that the oracle systems  $(\mathcal{H}^{\mathbb{O}}, \mathbb{O})$  and  $(\mathcal{RO}(\mathcal{H}), \mathbb{S}^{\mathcal{RO}(\mathcal{H})})$  are compatible and  $\epsilon$ -indistinguishable, for any adversary  $\mathbb{A} \in \text{Adv}(k, t)$ .

In this case,  $\text{Indiff}(\mathcal{H}, \mathcal{RO}(\mathbb{O}), \mathcal{RO}(\mathcal{H}), \mathbb{S}) \leq \epsilon(k, t)$ , where the left term stands for  $|\text{Pr}[\mathbb{A} | (\mathcal{H}^{\mathcal{RO}(\mathbb{O})}, \mathcal{RO}(\mathbb{O})) : \text{true}] - \text{Pr}[\mathbb{A} | (\mathcal{RO}(\mathcal{H}), \mathbb{S}^{\mathcal{RO}(\mathcal{H})}) : \text{true}]|$

The oracle set  $\mathbb{S}$  in this definition is usually referred to as the simulator. It is not a stand-alone oracle system, since it requires access to  $\mathcal{RO}(\mathcal{H})$  to compute its outputs.

#### IV. GENERIC SIMULATOR

Indifferentiability proofs are difficult because one has to come up with a simulator which mimics the inner-primitives consistently despite the fact that it

cannot access the list of the adversary calls to the hash oracle.

Consider for example the oracle system in Example II.1. To replace  $\mathcal{K}$  by a random oracle and simulate  $\mathcal{F}$ , we have to take into account that  $\mathcal{F}(\mathcal{F}(x)) = \mathcal{K}(x)$  is true for any  $x \in \{0, 1\}^n$  in the real setting. Hence, if an adversary queries  $\mathcal{F}(x)$ , gets  $y$  and then queries  $\mathcal{F}(y)$ , a simulator of  $\mathcal{F}$  has to output an answer matching  $\mathcal{K}(x)$ . Otherwise, a distinguisher can perform this query and in case  $\mathcal{K}(x) \neq \mathcal{F}(y)$ , claim to interact with the simulated world. Thus, the simulator should query  $\mathcal{K}$  on  $x$ , and forward what it gets as a reply to the query  $\mathcal{F}(y)$ .

We can now extrapolate a simulation strategy. The key idea is to detect when an adversary has enough information to compute a hash value for some input  $x$ . In other words, when queried on a value  $q$ , the simulator has to determine whether and to which  $x$  the adversary can associate a hash value, given an answer for  $q$ . In case such an  $x$  is determined, the simulator queries the hash of  $x$  to get a result  $t$ . Then, it uses  $t$  to enforce that the remaining calls to the inner-primitives needed for computing the hash of  $x$  are consistent with  $t$ . Thus, the idea is to identify chains of queries that can correspond to a hash execution. Our simulator uses an algorithm which, given a list of oracle queries and a query to the pivot oracle, decides whether and for which hash input the latter is a pivot query. In case it is, this algorithm outputs a matching hash input and a list of pre-pivot queries, among the given list of queries. Such an algorithm is called a *path-finder*<sup>1</sup>. Intuitively, it should have a non-trivial output as soon as there exists a satisfactory one, and any non-trivial output should correspond to a satisfying answer. This is captured by the following definition, where  $L_S$  denotes a list variable containing all exchanges performed with the simulator so far.

**Definition IV.1.** A path-finder algorithm `PathFinder` takes as input a query  $q \in \text{In}_{o_{\text{piv}}}$  and list  $L_S$  of queries and answers performed to oracles in  $\mathbb{O}$ . Its output is either the triple  $(\text{false}, \lambda, [ ])$ , or a triple of the form  $(\text{true}, x, \text{List})$ , with  $(x, \text{List}) \in \text{In}_{\mathcal{H}} \times \text{Xch}^*$  such that:

- 1) for any answer  $y$  to  $q$ , if there exists  $x \in \text{In}_{\mathcal{H}}$  such that  $\mathcal{H}(x)$  yields a list of queries  $Q$  satisfying  $[Q]_{i < \text{piv}(x)} \subseteq L_S$  and  $[Q]_{i = \text{piv}(x)} =$

<sup>1</sup>The algorithm is named after one of its plausible implementations, which involves building a particular graph. As we introduce a different graph construction for our theorem, we choose to introduce only this latter, which is the most relevant to the presentation of our result.

$(o_{\text{piv}}, q, y)$ ,  
then PathFinder outputs  $(\text{true}, \_, \_)$   
2) if PathFinder( $q, L_S$ ) =  
 $(\text{true}, x, [(o^1, q_1, y_1), \dots, (o^{p-1}, q_{p-1}, y_{p-1})])$   
then  $[(o^1, q_1, y_1), \dots, (o^{p-1}, q_{p-1}, y_{p-1})] \subseteq$   
 $L_S$ , and this list corresponds to the  
beginning of a hash execution on  $x$ , namely:  

$$\left\{ \begin{array}{l} \text{piv}(x) = p, \quad q_1 = H_1(\theta_1(x)), \\ \forall j \in [2..p-1], \\ \quad q^j = H_j(\theta_j(x), (o^{j-1}, q_{j-1}, y_{j-1})) \\ q = H_p(\theta_p(x), (o^{p-1}, q_{p-1}, a_{p-1})) \end{array} \right.$$

We assume that the execution time of the path-finder algorithm is bounded by a function  $t_{\text{PathFinder}}(\text{Card}(L_S))$ .

When the simulator detects a pivot query that allows to say  $\mathcal{H}(x) = t$ , it has to impose answers to pivot and post-pivot queries consistent with  $t$ . More precisely, consistency is achieved when applying  $H_{\text{post}}$  to pivot answer and post-pivot exchanges yields  $t$ . To perform this task, we introduce another algorithm: the *forward sampler*.

Given  $x$  and  $t$ , there is a set of lists of exchanges  $[v_j]_{j=\text{piv}(x)+1}^{\text{init}(x)}$  and values for  $y$  such that  $H_{\text{post}}(x, y, [v_j]_{j=\text{piv}(x)+1}^{\text{init}(x)}) = t$ . We denote this set by  $\text{Prelm}(t)$ . Informally, a *forward sampler* is an algorithm that samples an element in  $\text{Prelm}(t)$  while preserving the original distribution of  $((y, [v_j]_{j=\text{piv}(x)+1}^{\text{init}(x)}), t)$ . Obviously, a necessary condition for the existence of a forward sampler is that  $\text{Prelm}(t)$  is not empty.

**Definition IV.2.** A forward sampler  $\text{FwdSpr}$  is an algorithm that takes as input a pair  $(x_0, t_0)$  and outputs a distribution such that:  
 $Pr[\mathcal{U}(\text{Out}_{\mathcal{H}}) = t_0] Pr[\text{FwdSpr}(x_0, t_0) = (y_0, L_0)] =$   
 $Pr \left[ \begin{array}{l} \mathcal{U}(\text{Out}_{o_{\text{piv}}}) = y_0 \wedge \bigwedge_{j=\text{piv}(x)+1}^{\text{init}(x)} \mathcal{U}(\text{Out}_{o^j}) = y_j \wedge \\ L_0 = [v_j]_{j=\text{piv}(x)+1}^{\text{init}(x)} \wedge (y_0, L_0) \in \text{Prelm}(t_0) \end{array} \right]$   
where  $v_j = (o^j, H_j(\theta_j(x), v_{j-1}), y_j)$ , for  $j = \text{piv}(x) + 1, \dots, \text{init}(x)$ . We require that the execution time of  $\text{FwdSpr}$  is bounded by constant  $t_{\text{FwdSpr}}$ .

**EXAMPLE IV.1.** To get some intuition about this definition let us consider a degenerated case, where  $\text{Out}_{\mathcal{H}} = \text{Out}_{o_{\text{piv}}}$ , the pivot is the last step of iteration and  $H_{\text{post}}$  forwards the answer of the last query performed by the hash algorithm. Thus, intuitively, we expect that for each pair  $(x_0, t_0)$ ,  $\text{Prelm}(t_0)$  is the singleton  $(t_0, [])$  and a forward sampler should output this pair with probability 1. We check that this is what our definition imposes:

$$Pr[\mathcal{U}(\text{Out}_{o_{\text{piv}}}) = t_0 \wedge \bigwedge_{j=\text{piv}(x)+1}^{\text{init}(x)} \mathcal{U}(\text{Out}_{o^j}) = y_j \wedge$$

$$\begin{aligned} [] &= [v_j]_{j=\text{piv}(x)+1}^{\text{init}(x)} \wedge (t_0, []) \in \text{Prelm}(t_0) = \\ Pr[\mathcal{U}(\text{Out}_{o_{\text{piv}}}) = t_0 \wedge (t_0, []) \in \text{Prelm}(t_0)] &= \\ Pr[\mathcal{U}(\text{Out}_{o_{\text{piv}}}) = t_0] = Pr[\mathcal{U}(\text{Out}_{\mathcal{H}}) = t_0]. & \\ \text{Therefore, } Pr[\text{FwdSpr}(x_0, t_0) = (t_0, [])] &= 1. \end{aligned}$$

We can now provide implementations of oracles in the generic simulator. We recall that  $L_S(o_i)$  denotes the list of all tuples starting with  $o_i$  appearing in  $L_S$  and that if  $(o_i, q, y)$  is one of these tuples,  $L_S(o_i, q)$  denotes the value  $y$ , which is unique by construction.

**Definition IV.3.** The generic simulator  $\mathbb{S}$  is the following set of oracles compatible with  $\mathbb{O}$ . They have a shared memory of the form  $L_S \in \text{Xch}^*$ , and the initial memory  $\bar{m}$  of a system containing the simulator is chosen so that  $\bar{m}.L_S = []$ . Moreover,  $o_{\text{piv}}$  is implemented as follows:

```

Imp $\mathbb{S}(o_{\text{piv}})^{\mathcal{H}}(q, L_S) = \text{if } q \in \text{dom}(L_S(o_{\text{piv}})) \text{ then}$ 
  return  $L_S(o_{\text{piv}}, q)$ 
elseif PathFinder( $q, L_S$ ) =  $(\text{true}, x, \text{List})$  then
  let  $t \leftarrow \mathcal{H}(x)$  in
  ( $y, L$ ) := FwdSpr( $x, t$ );
   $L_S := L_S.((o_{\text{piv}}, q, y) : L)$ ;
else let  $y \leftarrow \mathcal{U}(\text{Out}_{o_{\text{piv}}})$  in
   $L_S := L_S.(o_{\text{piv}}, q, y)$ ;
endif
return  $y$ 

```

For any  $o \neq o_{\text{piv}}$  in  $\mathbb{N}_{\mathbb{O}}$ :

```

Imp $\mathbb{S}(o)^{\mathcal{H}}(q, L_S) = \text{if } q \in \text{dom}(L_S(o)) \text{ then}$ 
  return  $L_S(o, q)$ 
else let  $y \leftarrow \mathcal{U}(\text{Out}_o)$  in
   $L_S := L_S.(o, q, y)$ ;
return  $y$ 

```

If we let  $c$  be a constant bounding the time necessary to search in  $L_S$ , the implementation of  $o_{\text{piv}}$  is  $(k_s, t_s)$ -bounded, where  $k_s(\mathcal{H}) = 0$ , and  $t_{\text{FwdSpr}} + t_{\text{PathFinder}}(\text{Card}(L_S)) + c$ . The implementation of the other oracles is  $(0, c)$ -bounded.

This simulator works completely independently of the fact that multiple outputs may exist from which the path-finder has to choose. However, we notice that if it is possible that the path-finder can answer two distinct hash inputs  $x, x'$  for a pivot query, the simulator can only anticipate the adversary queries for one of these inputs to  $\mathcal{H}$ . If the adversary can easily uncover such values, our simulation strategy is flawed and should yield a large indistinguishability bound, which can reflect a misconception in the hash construction or a bad choice of the pivot index.



## V. THE THEOREM

Even though path-finder and forward sampler algorithms may prevent some obvious inconsistencies introduced in the idealized system, there are still cases in which they are not sufficient. Namely, when a pivot query is made to the simulator, consistency can only be enforced if, on the one hand, the path-finder can *detect that it is a pivot query*, and on the other hand, the pivot and post-pivot queries are still fresh, i.e., answers to these queries *have not been yet generated*.

### A. Capturing Dependencies: Anticipating System and Characteristic Graph

We want to capture dependencies enforced in the real setting by intermediate queries (performed by  $\mathcal{H}$  to oracles in  $\mathbb{O}$ ) in addition to direct and anticipated queries. To this end, we start by defining an intermediate system, the *anticipating system*  $\mathbb{O}_{ant}$ , which is the real system augmented with the anticipation of the post-pivot queries by oracle  $o_{piv}$ , and visibility labels that we introduce later on. The implementations of oracles in this system can be found in figure 4. This hybrid system, by enforcing the computation of all exchanges ever playing a role in the answer to the adversary, highlights problematic configurations.

Then, we introduce the *characteristic graph*, a data structure dedicated to the representation of dependencies between exchanges. Vertices of the graph are exchanges  $(o, q, a)$  computed either via a direct query or and indirect one. If an edge links two vertices, it means that they can be successive exchanges in a hash computation. To formalize that an adversary does not acquire the same knowledge of direct queries and intermediate queries necessary to the computation of a hash value, the visibility map associates vertices to visibility labels in  $\{Inv, PVis, Vis\}$  (standing for invisible, partially visible and visible and ordered this way). Intuitively, for an interaction with the anticipating system, pre-pivot intermediate exchanges are labelled invisible, while pivot and post-pivot are deemed partially visible. Moreover, direct exchanges are considered visible, as are exchanges anticipated by the simulator.

Formally, characteristic graphs are defined as follows.

**Definition V.1.** A characteristic graph  $CG$  is defined by a tuple  $(v_{root}, V, E, \mathcal{V})$  where:

- a root  $v_{root}$ ,
- a finite set of vertices  $V \subseteq Xch$ ,

- a set  $E \subseteq (V \cup v_{root}) \times \{0, 1\}^{\leq r} \times V$  of labeled edges such that:

- 1)  $(o^1, q, a) \in V, (v_{root}, x_1, (o^1, q, a)) \in E$  implies  $q = H_1(x_1)$ ;
- 2) for  $j \geq 2, ((o^{j-1}, q, a), x_j, (o^j, q', a')) \in E$  implies  $q' = H_j(x_j, (o^{j-1}, q, a))$ .

- $\mathcal{V}$  is a visibility map, which associates to every vertex in  $V$  a value in  $\{Inv, PVis, Vis\}$ ,
- $E$  contains all possible edges linking visible vertices:

- 1) for all visible vertex  $(o^1, q, a), q = H_1(x_1)$  implies  $(v_{root}, x_1, (o^1, q, a)) \in E$ ;
- 2) for  $j \geq 2$ , for all visible vertices  $((o^{j-1}, q, a)$  and  $(o^j, q', a')$ ), if  $q' = H_j(x_j, (o^{j-1}, q, a))$  then  $((o^{j-1}, q, a), x_j, (o^j, q', a')) \in E$ .

The set of characteristic graphs is denoted by  $\mathcal{CG}$ . We distinguish a particular graph  $CG_{init} = (v_{root}, [], \emptyset, \mathcal{V}_{init})$  with  $\text{dom}(\mathcal{V}_{init}) = \emptyset$  which we call the initial characteristic graph. We use the term *non-visible* to refer to vertices which are either partially visible or invisible. Moreover, we talk about visibility of *queries*: the visibility of a query  $(o, q)$  is the same as that of the (unique) vertex  $v$  in a characteristic graph such that  $v = (o, q, \_)$ . Intuitively, we are interested in chains of exchanges exhibited by the characteristic graph. We thus introduce the following terminology.

**Definition V.2.** • Given a graph, a path is a chain  $v_0 \xrightarrow{l_1} v_1 \xrightarrow{l_2} \dots \xrightarrow{l_n} v_n$  of vertices  $v_i$  such that for all  $i$ , edge  $(v_i, l_i, v_{i+1})$  belongs to the graph.

- A rooted path is a path starting with vertex  $v_{root}$ . A vertex is rooted whenever it belongs to a rooted path.
- A meaningful path is a rooted path such that if  $[x_1, \dots, x_L]$  is the list of labels on the sequence of edges, then there exists  $x$  such that  $\forall j = 1..L, \theta_j(x) = x_j$ .
- A meaningful path is said to be complete when  $L = \text{init}(x)$ . In such cases, bitstring  $x$  is then said to label the meaningful path to which it corresponds.

### B. Inconsistency Events

As foreseen previously, we face two main causes of inconsistencies. On the one hand, we can only expect that the path-finder detects pivot queries in case all pre-pivot queries have been performed before by the adversary, i.e., in case they form up a rooted path of visible queries in the characteristic graph. On the other hand, a pivot or post-pivot query corresponding

```

Imp $\mathbb{O}_{ant}$ ( $\mathcal{H}$ )( $x, L_{\mathcal{H}}, L_S$ ) =
if  $x \in \text{dom}(L_{\mathcal{H}})$  then
  ( $a^f, Q$ ) :=  $L_{\mathcal{H}}(x)$ ;
   $L_{\mathcal{H}} := L_{\mathcal{H}} \cdot (x, a^f, Q)$ ;
  return  $a^f$ 
else
   $l := \text{init}(x)$ ;
   $p := \text{piv}(x)$ ;
  ( $x_1, \dots, x_l$ ) :=  $\Theta(x)$ ;
  ( $o^1, q^1$ ) :=  $H_1(x_1)$ ;
  if  $q^1 \in \text{dom}(L_S(o^1))$  then
    ( $a^1, lbl$ ) :=  $L_S(o^1, q^1)$ ;
     $Q := [(o^1, q^1, a^1, lbl)]$ ;
  else let  $a^1 \leftarrow \mathcal{U}(\text{Out}_{o^1})$  in
     $L_S := L_S \cdot (o^1, q^1, a^1, Inv)$ ;
     $Q := [(o^1, q^1, a^1, Inv)]$ ;
  endif
  for  $j = 2$  to  $p - 1$  do
    ( $o^j, q^j$ ) :=  $H_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$ ;
    if  $q^j \in \text{dom}(L_S(o^j))$  then
      ( $a^j, lbl$ ) :=  $L_S(o^j, q^j)$ ;
       $Q := Q : (o^j, q^j, a^j, lbl)$ ;
    else let  $a^j \leftarrow \mathcal{U}(\text{Out}_{o^j})$  in
       $L_S := L_S \cdot (o^j, q^j, a^j, Inv)$ ;
       $Q := Q : (o^j, q^j, a^j, Inv)$ ;
    endif
  endfor
  for  $j = p$  to  $l$  do
    ( $o^j, q^j$ ) :=  $H_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$ ;
    if  $q^j \in \text{dom}(L_S(o^j))$  then
      ( $a^j, lbl$ ) :=  $L_S(o^j, q^j)$ ;
       $L_S := L_S \cdot (o^j, q^j, a^j, \max(PVis, lbl))$ ;
       $Q := Q : (o^j, q^j, a^j, \max(PVis, lbl))$ ;
    else let  $a^j \leftarrow \mathcal{U}(\text{Out}_{o^j})$  in
       $L_S := L_S \cdot (o^j, q^j, a^j, Inv)$ ;
       $Q := Q : (o^j, q^j, a^j, Inv)$ ;
    endif
  endfor

else let  $a^j \leftarrow \mathcal{U}(\text{Out}_{o^j})$  in
   $L_S := L_S \cdot (o^j, q^j, a^j, PVis)$ ;
   $Q := Q : (o^j, q^j, a^j, PVis)$ ;
endif
endfor
 $a^f := H_{post}(x, a^p, [Q]_{j>p})$ ;
 $L_{\mathcal{H}} := L_{\mathcal{H}} \cdot (x, a^f, Q)$ ;
return  $a^f$ 
endif

If  $o_i \neq o_{piv}$ :
Imp $\mathbb{O}_{ant}$ ( $o_i$ )( $q, L_S$ ) =
if  $q \in \text{dom}(L_S(o_i))$  then
  ( $y, \_$ ) :=  $L_S(o_i, q)$ ;
else let  $y \leftarrow \mathcal{U}(\text{Out}_{o_i})$  in
   $L_S := L_S \cdot (o_i, q, y, Vis)$ ;
return  $y$ 

Imp $\mathbb{O}_{ant}$ ( $o_{piv}$ )( $q, L_S, L_{\mathcal{H}}$ ) =
if  $q \in \text{dom}(L_S(o_{piv})|Vis)$  then
  ( $y, Vis$ ) :=  $L_S(o_{piv}, q)$ ;
elseif PathFinder( $q, L_S$ ) = ( $\text{true}, x, List$ ) then
  let  $t \leftarrow \mathcal{H}(x)$  in
    ( $o_{piv}, q, y$ ) :  $L := \Pi_3(L_{\mathcal{H}}(x))_{j \geq \text{piv}(x)}$ ;
     $L_S := L_S \cdot ((o_{piv}, q, y, Vis) : (L, Vis))$ ;
  elseif  $q \in \text{dom}(L_S(o_{piv})|PVis, Inv)$  then
    ( $y, \_$ ) :=  $L_S(o_{piv}, q)$ ;
     $L_S := L_S \cdot (o_{piv}, q, y, Vis)$ ;
  else let  $y \leftarrow \mathcal{U}(\text{Out}_{o_{piv}})$  in
     $L_S := L_S \cdot (o_{piv}, q, y, Vis)$ ;
  endif
return  $y$ 

```

For tuple  $(o_i, q, y, lbl) \in L_S$ ,  $L_S(o_i, q)$  denotes the pair  $(y, lbl)$  (which is unique by construction). We denote  $(L, lbl)$  a list of exchanges consisting in  $L$  except that all visibility labels are replaced by  $lbl$ , and  $(L|lbl)$  is the restriction of list  $L$  to elements of label  $lbl$ .

Figure 4. Implementations of the oracles in the anticipating system  $\mathbb{O}_{ant}$

to a hash input  $x$  can be already bound to an answer at the moment when the simulator detects that it is asked a pivot query. In such a case, this latter carries on running the forward sampler, but when updating the list of queries, it stumbles upon a preexisting vertex.

As far as the characteristic graph is concerned, it means that we have to ensure three invariants:

- 1) no two meaningful rooted paths have a common vertex,
- 2) along meaningfully rooted paths, visibility labels increase,
- 3) no vertex creation results in linking a meaningfully rooted path with a preexisting vertex.

Each time one of these invariants is broken, it corresponds to an *inconsistency event*, i.e. a case of bad simulation, acknowledged on the characteristic graph. However, probabilities are only defined over traces

previously. As a result, we need to define a way to map traces to sequences of graphs. Intuitively, this can be done putting to use a function  $\Gamma$  mapping a state to a characteristic graph. Then, calls to oracles  $o_i$  yield a sequence of length 1 and calls to the hash oracle are mapped to the sequence of graphs yield by applying function  $\Gamma$  to each intermediate state created during the iteration.

**Definition V.3.** *Given a characteristic graph  $CG$ ,  $v$  is a collision vertex, denoted  $v \in \text{CollVertex}(CG)$ , if there exist at least two distinct edges having  $v$  as a target (i.e. there exists  $(v', l') \neq (v'', l'')$  such that edges  $(v', l', v)$  and  $(v'', l'', v)$  appear in the graph), and both of them belong to a meaningfully rooted path going through  $v$ .*

We now introduce *inconsistency predicates* (cf figure 5 - a), mapping a graph stage to a boolean

value, and formalize inconsistency events using the temporal operator “eventually”.

**Definition V.4.** Let  $CG \xrightarrow{(o,q,y)} CG'$  be a graph stage. The inconsistency predicates are:

- *Collide* evaluates to true iff a collision vertex is created at this step, i.e.  $\text{CollVertex}(CG') - \text{CollVertex}(CG) \neq \emptyset$ .
- *Reveal* evaluates to true if  $CG'$  contains a vertex which is the first visible vertex of a meaningful non-visibly rooted path: there exists  $v, v' \in V'$  s.t. firstly,  $\mathcal{V}'(v) = \text{Vis}$  and  $\mathcal{V}'(v') \neq \text{Vis}$ , secondly, a meaningful path goes through  $v'$  in  $CG'$  and lastly edge  $(v', \_ , v)$  is in  $E'$ .
- *Link* evaluates to true if  $(o, q, y)$  is a visible vertex of  $CG'$  not belonging to  $CG$  and there exists a visible vertex  $(o', q', y')$  in  $V$  such that a visibly meaningfully rooted path of  $CG'$  goes through  $(o, q, y)$  and  $(o', q', y')$ .

### C. The Theorem

We can now state our theorem, according to which the indistinguishability between real and simulated systems is bounded by the probability of inconsistency events.

**THEOREM V.1.** Let  $h$  be an  $\mathbb{O}$ -overlayer. The composition of  $h$  and  $\mathcal{RO}(\mathbb{O})$  yields an oracle system  $(\mathcal{H}^{\mathcal{RO}(\mathbb{O})}, \mathcal{RO}(\mathbb{O}))$ . We denote by  $\mathbb{S}$  the generic simulator and  $\mathbb{O}_{ant}$  the anticipating system, which we have defined above. Then, for all adversaries  $\mathbb{A} \in \text{Adv}(k, t)$ ,

- $\mathbb{S}$  is  $(k_s, t_s)$ -bounded with  $k_s(\mathcal{H}) = 1$ , and  $t_s = t_{\text{FwdSplr}} + t_{\text{PathFinder}}(k' + 1) + c$ , where  $k' = \sum_{o \in \mathbb{O}} k(o)$ .
- the indistinguishability advantage is bounded by the probability of inconsistency events:  
 $\text{Indiff}(\mathcal{H}, \mathcal{RO}(\mathbb{O}), \mathbb{S}, \leq)$

$$\text{Pr}[\mathbb{A} | \mathbb{O}_{ant} : \text{F}_{\text{Collide} \vee \text{Reveal} \vee \text{Link}}]$$

where  $t_{\text{FwdSplr}}$  and  $t_{\text{PathFinder}}$  respectively bound the execution time of forward sampler  $\text{FwdSplr}$  and path-finder  $\text{PathFinder}$  used in the simulator, and  $c$  is a constant bounding the time necessary to search in  $L_{\mathcal{S}}$ .

**Sketch of Proof.** We present a proof sketch of this theorem in CIL developed in [Dau11], [BDK<sup>+</sup>10]. To relate real and simulated oracle systems, we use  $\mathbb{O}_{ant}$  and another intermediate system  $\text{FwdSplr}$ , show indistinguishability relations between them and then use transitivity. Proof steps are outlined in figure 6.

We have built the anticipating system  $\mathbb{O}_{ant}$  out of the real system by changing the implementation of

the pivot oracle so that it pre-computes the exchanges that the forward sampler is meant to anticipate inside the simulator. Therefore, perfect indistinguishability of the real system  $(\mathcal{H}^{\mathbb{O}}, \mathbb{O})$  and the anticipated system  $\mathbb{O}_{ant}$  follows from an argument of determinization.

The motivation behind the next step in the proof is to break the dependency existing between a hash oracle input and the matching output. We show that  $\mathbb{O}_{ant}$  is indistinguishable of another system, denoted  $\text{FwdSplr}$ , which is defined as follows. On a fresh input, the hash oracle resamples all pre-pivot exchanges (no matter their freshness), which yields a pivot query. Then, the oracle draws a hash output uniformly at random and applies  $\text{FwdSplr}$  to compute a pivot answer and post-pivot exchanges coherent with the hash output. As for the anticipating system, pre-pivot exchanges are deemed invisible and pivot and post-pivot ones partially visible.

Linking systems  $\mathbb{O}_{ant}$  and  $\text{FwdSplr}$  involves bisimulation-up-to arguments. Informally, we prove that invisible exchanges can be resampled up to some criterion, before getting rid of the test of this criterion and replacing original code with executions of forward sampler. Conditions on the bisimulation relations can be shown to hold as soon as no inconsistency predicate is broken. As a result, we can conclude that an adversary cannot distinguish between interacting with  $\mathbb{O}_{ant}$  and  $\text{FwdSplr}$  with a better advantage that the probability that *Reveal*, *Collide* or *Link* occurs.

Finally, we justify the perfect indistinguishability between  $\text{FwdSplr}$  and  $(\mathcal{RO}(\mathcal{H}), \mathbb{S}^{\mathcal{RO}(\mathcal{H})})$  by a determinization argument: indeed, as the hash oracle in  $\text{FwdSplr}$  computes a sequence of exchanges independent from its answer during its execution, we consider that this constitutes an anticipation of computations that system  $(\mathcal{RO}(\mathcal{H}), \mathbb{S}^{\mathcal{RO}(\mathcal{H})})$  only performs when necessary, i.e. when  $o_{\text{piv}}$  is called on a pivot query.

**Generalization.** The theorem and proof provided here are expressed for inner-primitives with random functions. Dealing with random permutations yields minor changes in the detailed proof and adds a term in the bound of the indistinguishability advantage capturing failure of injectivity as a possible bad simulation event.

## VI. APPLICATIONS

Our generic theorem reminds the importance of preventing length-extension attacks. Namely, applying the theorem on the MD mode yields a bound

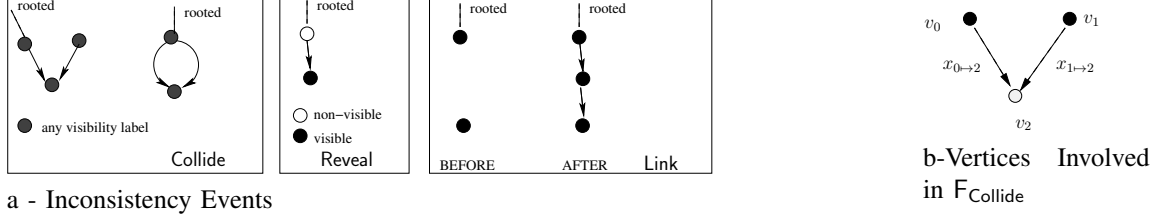


Figure 5.

determinization	$\left\{ \begin{array}{l} \text{Bisimulations up to condition } \phi \text{ relate } \mathbb{O}_{ant} \text{ to } \mathbb{FwdSpl} \\ \Pr[\phi] \leq \Pr[\mathbb{F}_{Collide} \vee \mathbb{F}_{Reveal} \vee \mathbb{F}_{Link}] = \epsilon \end{array} \right.$	determinization
$(\mathcal{H}^0, \mathbb{O}) \sim_0 \mathbb{O}_{ant}$	$\mathbb{O}_{ant} \sim_\epsilon \mathbb{FwdSpl}$	$\mathbb{FwdSpl} \sim_0 (\mathcal{RO}(\mathcal{H}), \mathbb{S}^{\mathcal{RO}(\mathcal{H})})$
$(\mathcal{H}^0, \mathbb{O}) \sim_\epsilon (\mathcal{RO}(\mathcal{H}), \mathbb{S}^{\mathcal{RO}(\mathcal{H})})$		

Figure 6. Sketch Of Proof

worth 1, as length extension always allows to realize event Reveal.

In our examples of application, we do not provide implementation for a path-finder algorithm (though to obtain an instantiated bound on the execution time we should), but only specify forward sampler algorithms.

As the events  $\mathbb{F}_{Collide}$ ,  $\mathbb{F}_{Reveal}$  and  $\mathbb{F}_{Link}$  can intersect, we take care to evaluate slightly weaker events but which partly avoid that some overlapping increases artificially the bound. The following decomposition proves useful in both examples we develop.

As there is only one possible label for an edge between two vertices, when event  $\mathbb{F}_{Collide}$  happens and results in the creation of a collision vertex  $v_2$ , then it necessarily involves vertices  $v_0, v_1$  linked to  $v_2$  such that  $v_0 \neq v_1$ . Without loss of generality, we suppose that  $v_0$  is created before  $v_1$  (see figure 5). We denote  $v_1 = (-, q_1, a_1)$  and let  $\text{RootCollide}$  capture the event that  $v_0 = v_{root}$  and  $v_1$  is created such that the collision happens. Then necessarily, there exist  $j, x_{0 \rightarrow 2}, x_{1 \rightarrow 2}$  such that  $H_1(x_{0 \rightarrow 2}) = H_j(x_{1 \rightarrow 2}, (-, q_1, a_1))$  since they both equal the query part of  $v_2$ . Furthermore,  $\text{WkCollide}$  is verified when  $v_0 \neq v_{root}$  is  $(-, q_0, a_0)$ , and  $v_1$  is created; i.e. when there exist  $(j, x_{1 \rightarrow 2})$  and  $(j', x_{0 \rightarrow 2})$  such that  $H_j(x_{1 \rightarrow 2}, (-, q_1, a_1)) = H_{j'}(x_{0 \rightarrow 2}, (-, q_0, a_0))$ . Note that we reason *on the appearance* of  $v_1$  in the graph, whether it is before or after creation of  $v_2$ . It allows us to state that  $\mathbb{F}_{Collide}$  implies  $\mathbb{F}_{\text{RootCollide} \vee \text{WkCollide}}$ . We define  $\text{WkLink}$  as the event in which a vertex  $v_1$  is created and gets linked to a preexisting vertex  $v_2$ , without imposing any visibility constraint on the vertices  $v_1$  and  $v_2$ . We can see that  $\mathbb{F}_{Collide \vee \text{Reveal} \vee \text{Link}}$  is implied by  $\mathbb{F}_{\text{WkLink} \vee$

```

FwdSplr(x, t) =
(t_0, ..., t_{k-1}) := (t[1..r], ..., t[(k-1)r + 1, r]);
let t' ← U(c) in
y^0 := t_0 || t';
q^1 := y^0;
for j = 1 to k - 1 do
let t'_j ← U({0, 1}^c) in
y^j := t_j || t'_j;
v^j := (F, q^j, y^j);
q^{j+1} := y^j;
endfor
return (y^0, [v^j]_{j=(piv(x)+1)..(piv(x)+k-1)})

```

Figure 7. Forward Sampler Algorithm for the Sponge Example

$$\mathbb{F}_{\text{RootCollide} \vee \text{WkCollide}} \vee \mathbb{F}_{\text{Reveal} \wedge \neg \text{WkLink} \wedge \neg \text{Collide}}$$

### A. The Sponge Construction

We define the forward sampler so that it parses a hash output  $t$  into  $k$  blocks of  $r$  bits and draws iteratively the  $c$  missing bits of the answers to pivot and post-pivot queries. The implementation of the forward sampler used by the simulator for the sponge construction is in figure ??.

If  $\mathbb{F}_{\text{WkCollide}}$  happens at the  $\ell$ -th fresh query then there exist  $(j, x_{1 \rightarrow 2})$  and  $(j', x_{0 \rightarrow 2})$  such that  $H_j(x_{1 \rightarrow 2}, (\mathcal{F}, q_1, a_1)) = H_{j'}(x_{0 \rightarrow 2}, (\mathcal{F}, q_0, a_0))$ , which imposes  $\text{Last}_c(a_0) = \text{Last}_c(a_1)$ . Since  $a_1$  is drawn uniformly at random, this happens with probability at most  $\frac{\ell-1}{2^c}$ . Summing on  $\ell$  yields a bound of  $\frac{k_{tot}(k_{tot}-1)}{2^{c+1}}$ .

When  $\mathbb{F}_{\text{RootCollide}}$  occurs, there exists  $v_1$  such that  $H_1(x_{0 \rightarrow 2}) = H_j(x_{1 \rightarrow 2}, (\mathcal{F}, q_1, a_1))$  for some labels  $x_{0 \rightarrow 2}$  and  $x_{1 \rightarrow 2}$  and index  $j$ . Hence, the last  $c$  bits of  $a_1$  to be worth  $0^c$ . The probability that this happens

is bounded by  $\frac{k_{tot}}{2^c}$ .

If  $F_{\text{WkLink}}$  happens at the  $\ell$ -th direct query creating  $v_1$ , then there exists  $v_2$  to which  $v_1$  gets linked by an edge  $(j, x_j)$ . With  $a_1$  drawn uniformly at random, the probability that there is a  $v_2$  such that  $H_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$  is at most  $\frac{\ell-1}{2^c}$ . Summing on  $\ell$ , it results in a bound of  $\frac{k_{tot}(k_{tot}-1)}{2^{c+1}}$ .

Finally, we bound the probability of  $F_{\text{Reveal} \wedge \neg \text{WkLink} \wedge \neg \text{Collide}}$  by  $\frac{k(\mathcal{F})}{2^c}$ . Indeed, if this occurs, a vertex  $v_1 = (\mathcal{F}, q_1, a_1)$ , non-visible, gets linked to  $v_2 = (\mathcal{F}, q_2, a_2)$ , visible, by an edge labeled by  $(j, x_j)$ . Since we assume  $\neg \text{WkLink}$ , necessarily,  $v_1$  is created before  $v_2$ . Realizing *Reveal* means that  $H_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$ . Since we assume  $\neg \text{Collide}$ , there is only one vertex  $v_1$  which can satisfy this equation. With  $v_1$  at most partially visible, the probability that a good query  $q_2$  is issued is bounded by  $\frac{1}{2^c}$ . To conclude, we sum over the total number of direct queries.

**THEOREM VI.1.** *We consider the sponge construction. For an adversary  $\mathbb{A} \in \text{Adv}(k, t)$ ,*

$\text{Indiff}(\text{Sponge}, \mathcal{RO}(\mathcal{F}), \mathcal{RO}(\text{Sponge}), \mathbb{S}) \leq \frac{k_{tot}^2}{2^c} + \frac{k(\mathcal{F})}{2^c}$  probability less than  $\frac{\ell-1}{2^n}$ . Summing on  $\ell$ , it results in a bound of  $\frac{k_{tot}(k_{tot}-1)}{2^{n+1}}$ .

where  $k_{tot} = k(\mathcal{F}) + \mathfrak{L}_{sp} * k(\mathcal{H})$ .

In [BDPA08], Bertoni et. al. present a clever proof of the the sponge indifferentiability concluding to a bound of  $\frac{k_{tot}(k_{tot}+1)}{2^{c+1}}$ . We obtain a greater bound, containing terms which are omitted in their final bound computation, as was first suggested in [BCCM<sup>+</sup>08]. The missing term corresponds to the probability that length-extension attacks can be carried out, which, even though the authors propose a simulator different from ours, should not be overlooked in their computation.

### B. The ChopMD Construction

We consider the hash function ChopMD introduced in [CDMP05] and inspired of [DGH<sup>+</sup>04]. The implementation of the Chop construction can be found in figure 7. It is obtained from the Merkle-Damgård construction by chopping off the last  $s$  bits of the output in order to prevent extension attacks.

On input  $(x, t)$ , we define *FwdSplr* to sample uniformly the  $s$  missing bits to compute the result of the pivot query  $y^{\text{piv}(x)}$ , and outputs the concatenation of  $t$  with these bits.

Probability computations are very similar to the previous case. If  $\text{WkCollide}$  happens at the  $\ell$ -th fresh query, then the equation between  $v_0$  and  $v_1$  imposes  $a_0 = a_1$ . For a random  $a_1$ , this happens with

Oracle *ChopMD<sub>s</sub>*

```

In(ChopMDs) = {0, 1}≤264,
Out(ChopMDs) = {0, 1}n-s
Imp(ChopMDs)ℱ(x, Lchop) =
if x ∈ dom(Lchop) then
  return Lchop(x)
else
  l := ⌈|x|/r⌉;
  w := x || 10l*s-r-|x|-1;
  (w1, ..., wl) := (w[1, r], ..., w[r(l - 1) + 1, rl]);
  a0 := 0n;
  for j = 1 to l do
    qj := aj-1 || wj;
    let aj ← ℱ(qj) in
  endfor
  af := Firstn-s(al);
  Lchop := Lchop.(x, af);
  return af
endif

```

Figure 8. ChopMD Implementation

In turn, if  $F_{\text{RootCollide}}$  holds, then there exist  $j, x_{0 \rightarrow 2}, x_{1 \rightarrow 2}$  such that  $H_1(x_{0 \rightarrow 2}) = H_j(x_{1 \rightarrow 2}, (\mathcal{F}, q_1, a_1))$ . Necessarily,  $j > 1$ , otherwise  $v_1 = v_2$ . It yields  $a_1 = 0^n$ , occurring with probability less than  $\frac{k_{tot}}{2^n}$ .

Then we bound of the probability that  $F_{\text{WkLink}}$  occurs by  $\frac{k_{tot}(k_{tot}-1)}{2^{n+1}}$ . Indeed, if it happens at the  $\ell$ -th fresh query to  $\mathcal{F}$ , then there exists a vertex  $v_2$  to which  $v_1$  gets linked by an edge  $(j, x_j)$ . Since answer  $a_1$  is random, the probability that there is a  $v_2$  such that  $H_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$  is bounded by  $\frac{\ell-1}{2^n}$ . We conclude by summing on  $\ell$ .

Finally, when  $F_{\text{Reveal} \wedge \neg \text{WkLink} \wedge \neg \text{Collide}}$  occurs, there exists a vertex  $v_1 = (\mathcal{F}, q_1, a_1)$ , non-visible, which gets linked to  $v_2 = (\mathcal{F}, q_2, a_2)$ , visible, by an edge labeled by  $(j, x_j)$ . Again,  $\neg \text{WkLink}$  implies  $v_1$  is created before  $v_2$ . Since *Reveal* happens,  $H_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$ . Because of  $\neg \text{Collide}$ , there is only one satisfactory vertex  $v_1$ . With  $v_1$  at most partially visible, the probability that a satisfactory query  $q_2$  is performed is bounded by  $\frac{1}{2^s}$ . We then have to sum over the total number of direct queries issued, which provides a bound of  $\frac{k(\mathcal{F})}{2^s}$ .

These three bounds result in a global bound of  $\frac{k_{tot}^2}{2^n} + \frac{k(\mathcal{F})}{2^s}$ .

**THEOREM VI.2.** *We consider the ChopMD<sub>s</sub> construction. For an adversary  $\mathbb{A} \in \text{Adv}(k, t)$ ,*

$$\text{Indiff}(\text{ChopMD}_s, \mathcal{F}, \mathcal{RO}(\text{ChopMD}_s), \mathbb{S}) \leq \frac{k_{tot}^2}{2^n} + \frac{k(\mathcal{F})}{2^s} \text{AHMP10}$$

where  $k_{tot} = k(\mathcal{F}) + \mathcal{L} * k(\mathcal{H})$ .

Indifferentiability results for the Chop construction already appear in various works. In [CDMP05], Coron et. al. determine a bound for this construction considering a random permutation in place of  $\mathcal{F}$ . We only have a result for random functions, yet we notice that their proof results in a bound of  $\mathcal{O}(\frac{(\mathcal{L} * k_{tot})^2}{2^s})$ , which is the same magnitude.

Later, Maurer and Tessaro show in [MT07] that using a prefix-free padding function yields a bound of  $\mathcal{O}(\frac{(\mathcal{L} * k_{tot})^2}{2^n})$ . We obtain the same bound: no *meaningful* path can be obtained as an extension of a meaningful path, so that *Reveal* can only happen when  $y' = \text{First}_n(q)$  belongs to an *invisible* vertex. As a consequence, the adversary has to guess all  $n$  bits of  $y'$  and our second term becomes  $\frac{k(\mathcal{F})}{2^n}$ .

Finally, our result slightly improves the  $\mathcal{O}(\frac{3(n-s)(k(\mathcal{F})+k(\mathcal{H}))}{2^s})$  of Chang and Nandi in [CN08].

## VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented a generic reduction theorem to prove indifferentiability from a random oracle for hash constructions when their inner-primitives are modeled by random functions. In an attempt to develop a formal approach to security proofs, we have extended the framework of the logic CIL of [BDK<sup>+</sup>10] with a formalization of modes of operations - *overlayers* - and proven our theorem in this logic.

## REFERENCES

- Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Sha-3 proposal blake. Submission to NIST (Round 3), 2010.
- [BCCM<sup>+</sup>08] Emmanuel Bresson, Anne Canteaut, Benoit Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-Francois Misarsky, M. Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, and Marion Videau. Shabal, a submission to nists cryptographic hash algorithm competition. Submission to NIST, 2008.
- [BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proceedings of the 37th Symposium on Foundations of Computer Science, IEEE*, pages 514–523. IEEE, 1996.
- [BDK<sup>+</sup>10] Gilles Barthe, Marion Daubignard, Bruce Kapron, Yassine Lakhnech, and Vincent Laporte. Deciding equality of probabilistic terms. In *Proceedings of LPAR'10, Lecture Notes in Computer Science*. Springer, 2010. To appear.
- [BDKL10] Gilles Barthe, Marion Daubignard, Bruce Kapron, and Yassine Lakhnech. Computational indistinguishability logic. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, (CCS'10)*, Chicago, USA, oct 2010. ACM.
- [BDPA07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. *Ecrypt Hash Workshop*, may 2007.
- [BDPA08] Guido Bertoni, Joan Daemen, Michael 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.
- [BDPA11] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The keccak sha-3 submission. Submission to NIST (Round 3), 2011.
- [BMN09] Rishiraj Bhattacharyya, Avradip Mandal, and Mridul Nandi. Indifferentiability characterization of hash functions and optimal bounds of popular domain extensions. In Bimal Roy and Nicolas Sendrier, editors, *Progress*

- in *Cryptology - INDOCRYPT 2009*, volume 5922 of *Lecture Notes in Computer Science*, pages 199–218. Springer Berlin / Heidelberg, 2009.
- [BPW03] Michael Backes, Birgit Pfizmann, and Michael Waidner. A composable cryptographic library with nested operations. In *ACM Conference on Computer and Communications Security*, pages 220–230, 2003.
- [BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
- [BR06] Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the emd transform. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology - ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer Berlin / Heidelberg, 2006.
- [Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. *Foundations of Computer Science, Annual IEEE Symposium on*, pages 136–145, 2001.
- [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.
- [CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
- [CN08] Donghoon Chang and Mridul Nandi. Improved indistinguishability 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.
- [Dam89] Ivan Damgård. A design principle for hash functions. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, pages 416–427, 1989.
- [Dam90] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer Berlin / Heidelberg, 1990.
- [Dau11] Marion Daubignard. *Formal Methods for Concrete Security Proofs*. PhD thesis, University of Grenoble, Verimag, France, 2011.
- [DGH<sup>+</sup>04] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the cbc, cascade and hmac modes. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2004.
- [FLS<sup>+</sup>10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The skein hash function family. Submission to NIST (Round 3), 2010.
- [GKM<sup>+</sup>11] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a sha-3 candidate. Submission to NIST (Round 3), 2011.
- [Jou04] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, pages 306–316, 2004.
- [KK06] John Kelsey and Tadayoshi Kohno. Herding hash functions and the nostradamus attack. In *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, pages 183–200, 2006.
- [KS05] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than  $2^n$  work. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 474–490, 2005.

- [Mau02] Ueli M. Maurer. Indistinguishability of random systems. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*, pages 110–132. Springer, 2002.
- [Mer89] Ralph C. Merkle. One way hash functions and des. In 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*, pages 428–446. Springer, 1989.
- [MRH04] Ueli 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*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer Berlin / Heidelberg, 2004.
- [MT07] Ueli M. Maurer and Stefano Tessaro. Domain extension of public random functions: Beyond the birthday barrier. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 187–204. Springer, 2007.
- [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.
- [Sho04] Victor Shoup. Sequences of games: a tool for taming complexity in security proofs. Cryptology ePrint Archive, Report 2004/332, 2004.
- [Wu11] Hongjun Wu. The hash function jh. Submission to NIST (round 3), 2011.