Generic Indifferentiability Proofs of Hash Designs

Marion Daubignard, Pierre-Alain Fouque, Yassine Lakhnech

▶ **To cite this version:**

Marion Daubignard, Pierre-Alain Fouque, Yassine Lakhnech. Generic Indifferentiability Proofs of Hash Designs. 25th IEEE Computer Security Foundations Symposium, CSF 2012, Jun 2012, Cambridge, United States. pp.14, 10.1109/CSF.2012.13 . hal-01094323

HAL Id: hal-01094323

https://inria.hal.science/hal-01094323

Submitted on 12 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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, 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 exenders 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 bitstring. To construct a hash function, cryptographers first build a fixed input-length function $f : \{0,1\}^m \to \{0,1\}^n$, 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 indifferentiability results, cryptographers need to show that two idealized systems are indistin-

guishable. 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, indifferentiability 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 indifferentiability 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 indifferentiability 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+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 indifferentiability 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+08].

Outline. The next section briefly provides background on CIL, while the generic definition to capture hash designs and a formal definition of their indifferentiability 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, $\mathsf{dom}(L)$ 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 $s \leq |w|$, $\mathsf{Last}_s(w)$ and $\mathsf{First}_s(w)$ 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 λ.

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 $i = 1, \cdots n$, M_i be a countable set of memories (states) and $\mathsf{M}_{\mathbb{O}} = M_1 \times \cdots \times M_n$. Furthermore, let $o_1, \ldots, o_n$ be disjoint names of oracles that operate on $\mathsf{M}_{\mathbb{O}}$. An *implementation* of o_i is a probabilistic algorithm $\mathsf{Imp}(o_i)^{o_{j_1},\cdots,o_{j_i}}$ which:

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

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

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

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

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 query and answer domains. Compatible systems can however differ in the oracle implementations and memories. We therefore use the notation $\mathsf{Imp}_{\mathbb{O}}(o_i)$ (resp. $\mathsf{Imp}_{\mathbb{O}'}(o_i)$) 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 $o \in \mathsf{N}_{\mathbb{O}}$, $q \in \mathsf{In}(o)$ and $a \in \mathsf{Out}(o)$. Let Xch denote the set of exchanges. *Final exchanges* are of the form $(o_{\mathrm{F}}, -, -)$, i.e. queries to the finalization oracle. The set of final exchanges is denoted by $\mathsf{Xch}_{\mathrm{F}}$. The sets Que of queries and Ans of answers are, respectively, defined by $\mathsf{Que} = \{(o, q) \mid (o, q, a) \in \mathsf{Xch}\}$ and $\mathsf{Ans} = \{(o, a) \mid (o, q, a) \in \mathsf{Xch}\}$, while for a particular oracle o_i, $\mathsf{Que}(o_i)$ (resp. $\mathsf{Ans}(o_i), \mathsf{Xch}(o_i)$) only contains elements of Que (resp. $\mathsf{Ans}, \mathsf{Xch}$) starting with o_i.

Oracle $\mathcal{F}$
$\mathsf{Imp}_{\mathbb{O}}(\mathcal{F})(q, L_{\mathcal{F}}) =$
if $q \in \mathsf{dom}(L_{\mathcal{F}})$ then
 return $L_{\mathcal{F}}(q)$
else
 let $a \leftarrow \mathcal{U}(\{0,1\}^n)$ in
 $L_{\mathcal{F}} := L_{\mathcal{F}}.(q, a)$;
 return a
endif

Oracle $\mathcal{K}$
$\mathsf{Imp}_{\mathbb{O}}(\mathcal{K})^{\mathcal{F}}(q, L_{\mathcal{K}}) =$
if $q \in \mathsf{dom}(L_{\mathcal{K}})$ then
 return $L_{\mathcal{K}}(q)$
else
 let $a \leftarrow \mathcal{F}(q)$ in
 let $b \leftarrow \mathcal{F}(a)$ in
 $L_{\mathcal{K}} := L_{\mathcal{K}}.(q, b)$;
 return b
endif

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

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

$$\begin{array}{rcl} \mathsf{A} & : & \mathsf{M}_{\mathbb{A}} \longrightarrow \mathcal{D}(\mathsf{Que} \times \mathsf{M}_{\mathbb{A}}) \\ \mathsf{A}_{\downarrow} & : & \mathsf{Xch} \times \mathsf{M}_{\mathbb{A}} \to \mathcal{D}(\mathsf{M}_{\mathbb{A}}) \end{array}$$

Informally, the interaction between an oracle system and an adversary starts from the initial memory $(\bar{m}_{\mathbb{A}}, \bar{m}_{\mathbb{O}})$. Using 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. 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* Σ *of actions, with a distinguished subset* Σ_{F} *of finalization actions;*
- *a (partial probabilistic) transition function* $\mathsf{st} : \mathsf{M} \longrightarrow \mathcal{D}(\Sigma \times \mathsf{M})$.

A partial execution sequence of $\mathcal{S}$ is a sequence η of the form $m_0 \xrightarrow{act_1} m_1 \xrightarrow{act_2} \ldots \xrightarrow{act_k} m_k$ such that $m_0 = \bar{m}, act_i \in \Sigma$, $m_{i-1}, m_i \in \mathsf{M}$, and $\Pr[\mathsf{st}(m_{i-1}) = (act_i, m_i)] > 0$, for $i = 1, \ldots, k$. If $k = 1$, then η is a step. If $act_k \in \Sigma_{\mathrm{F}}$ or $m_k \notin \mathsf{dom}(\mathsf{st})$, then η 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 η is $\prod_{i=1}^{k} Pr[\mathsf{st}(m_{i-1}) = (act_i, m_i)]$.

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 $\mathsf{M} = \mathsf{M}_{\mathbb{A}} \times \mathsf{M}_{\mathbb{O}}$, the initial memory is $(\bar{m}_{\mathbb{A}}, \bar{m}_{\mathbb{O}})$, the set of actions is $\Sigma = \mathsf{Xch}$, and $\Sigma_{\mathrm{F}} = \mathsf{Xch}_{\mathrm{F}}$, and*

$$\begin{array}{rl} \mathit{st}_{\mathbb{A}|\mathbb{O}}(m_{\mathbb{A}}, m_{\mathbb{O}}) \stackrel{def}{=} & \mathsf{let}\ ((o,q), m'_{\mathbb{A}}) \leftarrow \mathsf{A}(m_{\mathbb{A}})\ \mathsf{in} \\ & \mathsf{let}\ (a, m'_{\mathbb{O}}) \leftarrow \mathsf{Imp}_{\mathbb{O}}(o)(q, m_{\mathbb{O}})\ \mathsf{in} \\ & \mathsf{let}\ m''_{\mathbb{A}} \leftarrow \mathsf{A}_{\downarrow}((o,q,a), m'_{\mathbb{A}})\ \mathsf{in} \\ & \mathsf{return}\ ((o,q,a), (m''_{\mathbb{A}}, m'_{\mathbb{O}})) \end{array}$$

Let $k : \mathsf{N}_{\mathbb{O}} \to \mathbb{N}$. An adversary is called *k-bounded*, if for every $o \in \mathsf{N}_{\mathbb{O}}$, 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 $(k, t) \in (\mathsf{N}_{\mathbb{O}} \to \mathbb{N}) \times \mathbb{N}$ and talk about (k, t)-bounded adversaries, whose set we denote $\mathsf{Adv}(k, t)$.

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

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

- *A* partial trace *is a sequence τ of the form $m_0 \xrightarrow{act_1} m_1 \xrightarrow{act_2} \ldots \xrightarrow{act_k} m_k$, where $m_0, \ldots, m_k \in \mathsf{M}_{\mathbb{O}}$ and $act_1, \ldots, act_k \in \mathsf{Xch}$ such that for $i = 1, \ldots, k$ and $act_i = (o_i, q_i, a_i)$, $\Pr[\mathsf{Imp}_{\mathbb{O}}(o_i)(q_i, m_{i-1}) = (a_i, m_i)] > 0$. A* trace *is a partial trace τ such that $m_0 = \bar{m}_{\mathbb{O}}$ and $act_k = (o_{\mathrm{F}}, \_, \_)$.*
- *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 η induces a trace $\mathcal{T}(\eta)$ simply by erasing the adversary memory at each step. Consequently, for each trace τ, we define the set $\mathcal{T}^{-1}(\tau)$ of execution sequences that are erased to τ, and for every event E the probability $Pr(\mathbb{A}|\mathbb{O} : E) = Pr(\mathbb{A}|\mathbb{O} : \mathcal{T}^{-1}(E))$.

B. The Logic

The logic CIL 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. 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 practise, unconditional bisimulation may be insufficient. Therefor, 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 mul-

tiple 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 $l = m + k$ iterations, where m is the number 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 Pad_{sp} that yields a bitstring $x_1||\ldots||x_m$ 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 more times to get a collection of answers $(a^{m+1}, \ldots, a^l)$. The final output is then obtained by concatenation of the first r bits of each a^j: $\mathsf{First}_r(a^{m+1})||\ldots||\mathsf{First}_r(a^l)$. The implementation is provided in Figure 2.*

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+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 Θ that describes how an input x is padded and cut into $\mathsf{init}(x)$ blocks $\theta_1(x), \ldots, \theta_{\mathsf{init}(x)}(x)$, with $\mathsf{init}(x) \in [0, \mathfrak{L}]$;
2) a finite sequence $[o^1, \ldots, o^{\mathfrak{L}}]$ 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 $[o^1, \ldots, o^{\mathfrak{L}}]$. All known hash designs are based on a fixed sequence $[o^1, \ldots, o^{\mathfrak{L}}]$;
3) a function piv with $\mathsf{piv}(x) \in [1, \mathsf{init}(x)]$ and such that the output of the post-processing does not depend on calls performed to the inner-primitives before $\mathsf{piv}(x)$. We call $\mathsf{piv}(x)$ the *pivot index*.
4) functions $\mathsf{H}_1, \ldots, \mathsf{H}_{\mathfrak{L}}$ that compute the input queries to the inner-primitives;
5) a post-processing function H_{post}.

$\mathsf{In}(Sponge) = \{0,1\}^{\leq 2^{64}}$, $\mathsf{Out}(Sponge) = \{0,1\}^K$
$\mathsf{Imp}(Sponge)^{\mathcal{F}}(x, L_{sp}) =$
if $x \in \mathsf{dom}(L_{sp})$ then
  return $L_{sp}(x)$
else
  $l := \mathsf{init}_{sp}(x)$;
  $w := \mathsf{Pad}_{sp}(x)$;
  $m := \frac{|w|}{r}$;
  $(x_1, \ldots, x_p) := (w[1, r], \ldots, ;$
  $\qquad\qquad w[r(m-1)+1, rm])$;
  $a^0 := 0^{r+c}$;
  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
  $a^f := \mathsf{First}_r(a^{m+1})||\ldots||\mathsf{First}_r(a^l)$;
  $L_{sp} := L_{sp}.(x, a^f, Q)$;
  return a^f
endif

where $\mathsf{init}_{sp}(x) = m + k$, with $m = |w|/r$ and $m * r$ the length of the output of the sponge construction.

Figure 2. Sponge Implementation

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 p 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 p calls to the inner-primitive $\mathcal{F}$, while there are k calls performed during the squeezing case. For this construction, $[o^1, \ldots, o^{\mathfrak{L}}]$ is then a list where $\mathcal{F}$ is repeated $\mathfrak{L}$ times. Any index in the absorbing phase and the first index of the squeezing 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 indifferentiability bound. For sponge, this is the first index of the squeezing phase.*

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

oracle system $\mathbb{O}$ with oracles in $\mathsf{N}_{\mathbb{O}}$.

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

- *$[o^1,\ldots,o^{\mathfrak{L}}]$ is a sequence of oracles in $\mathsf{N}_{\mathbb{O}}$.*
- *$\Theta : \mathsf{In}_{\mathcal{H}} \to (\{0,1\}^{\leq r})^+$ with $\Theta(x) = (\theta_1(x),\ldots,\theta_{\mathsf{init}(x)}(x))$ the input transformation and $r > 0$ the block length.*
- *$\mathsf{piv} : \mathsf{In}_{\mathcal{H}} \to [1,\mathfrak{L}]$ outputs a pivot index such that $\mathsf{piv}(x) \leq \mathsf{init}(x)$. For simplicity, we require that there is an oracle o_{piv}, that we call* the pivot oracle*, such that for any x, $o^{\mathsf{piv}(x)} = o_{\mathsf{piv}}$.*
- *functions $\mathsf{H}_1 : \{0,1\}^{\leq r} \to \mathsf{In}(o^1)$ and $\mathsf{H}_j : \{0,1\}^{\leq r} \times \mathsf{Xch} \to \mathsf{In}(o^j)$ for $j = 2,\ldots,\mathfrak{L}$.*
- *$\mathsf{H}_{post} : \mathsf{In}_{\mathcal{H}} \times \mathsf{Out}_{o_{\mathsf{piv}}} \times \mathsf{Xch}^* \to \mathsf{Out}_{\mathcal{H}}$ is the post-processing function.*

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

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

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

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

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 $\mathsf{In}_{\mathcal{H}}$ to $\mathsf{Out}_{\mathcal{H}} \times \mathsf{Xch}^*$; 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

$\mathsf{Imp}(\mathcal{H})^{o^1,\ldots,o^{\mathfrak{L}}}(x,L_{\mathcal{H}}) =$ if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  return $L_{\mathcal{H}}(x)$
else
  $l := \mathsf{init}(x)$;
  $(x_1,\ldots,x_l) := \Theta(x)$;
  $q^1 := \mathsf{H}_1(x_1)$;
  let $a^1 \leftarrow o^1(q^1)$ in
  $Q := [(o^1,q^1,a^1)]$;
  for $j = 2$ to l do
    $q^j := \mathsf{H}_j(x_j,(o^{j-1},q^{j-1},a^{j-1}))$;
    let $a^j \leftarrow \mathsf{Imp}_{\mathbb{O}}(o^j)(q^j)$ in
    $Q := Q : (o^j,q^j,a^j)$;
  endfor
  $a^f := \mathsf{H}_{post}(x,a^{\mathsf{piv}(x)},[Q]_{k>\mathsf{piv}(x)})$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x,a^f,Q)$;
  return a^f
endif

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

that hash constructions are indistinguishable from a random function (see [CDMP05]), which is commonly named indifferentiability 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 indifferentiability.

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

if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then return $L_{\mathcal{H}}(x)$
else let $y \leftarrow \mathcal{U}(\mathsf{Out}_{\mathcal{H}})$ 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 ϵ-*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 ϵ-indistinguishable, for any adversary $\mathbb{A} \in \mathsf{Adv}(k,t)$.*

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

The oracle set $\mathbb{S}$ in this definition is usually refered 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*[1]. 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 $L_{\mathcal{S}}$ denotes a list variable containing all exchanges performed with the simulator so far.

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

1) *for any answer* y *to* q*, if there exists* $x \in \mathsf{In}_{\mathcal{H}}$ *such that* $\mathcal{H}(x)$ *yields a list of queries* Q *satisfying* $[Q]_{i<\mathsf{piv}(x)} \subseteq L_{\mathcal{S}}$ *and* $[Q]_{i=\mathsf{piv}(x)} = (o_{\mathsf{piv}}, q, y)$*, then* $\mathsf{PathFinder}$ *outputs* $(\mathsf{true}, \_, \_)$
2) *if* $\mathsf{PathFinder}(q, L_{\mathcal{S}}) = (\mathsf{true}, x, [(o^1, q_1, y_1), \ldots, (o^{p-1}, q_{p-1}, y_{p-1})])$ *then* $[(o^1, q_1, y_1), \ldots, (o^{p-1}, q_{p-1}, y_{p-1})] \subseteq L_{\mathcal{S}}$*, and this list corresponds to the beginning of a hash execution on* x*, namely:*
$$\begin{cases} \mathsf{piv}(x) = p, \quad q_1 = \mathsf{H}_1(\theta_1(x)), \\ \forall j \in [2..p-1], \\ \qquad q^j = \mathsf{H}_j(\theta_j(x), (o^{j-1}, q_{j-1}, y_{j-1})) \\ q = \mathsf{H}_p(\theta_{p-1}(x), (o^{p-1}, q_{p-1}, a_{p-1})) \end{cases}$$

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

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

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

Example IV.1. *To get some intuition about this definition let us consider a degenerated case, where* $\mathsf{Out}_{\mathcal{H}} = \mathcal{U}(\mathsf{Out}_{o_{\mathsf{piv}}})$*, the pivot is the last step of iteration and* H_{post} *forwards the answer of the last query performed by the hash algorithm. Thus, intuitively, we expect that*

[1]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.

for each pair (x_0, t_0), $\mathsf{PreIm}(t_0)$ *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:*
$Pr[\mathcal{U}(\mathsf{Out}_{o_{\mathsf{piv}}}) = t_0 \wedge \bigwedge_{j=\mathsf{piv}(x)+1}^{\mathsf{init}(x)} \mathcal{U}(\mathsf{Out}_{o^j}) = y_j \wedge [\,] = [v_j]_{j=\mathsf{piv}(x)+1}^{\mathsf{init}(x)} \wedge (t_0, [\,]) \in \mathsf{PreIm}(t_0)] = Pr[\mathcal{U}(\mathsf{Out}_{o_{\mathsf{piv}}}) = t_0 \wedge (t_0, [\,]) \in \mathsf{PreIm}(t_0)] = Pr[\mathcal{U}(\mathsf{Out}_{o_{\mathsf{piv}}}) = t_0] = Pr[\mathcal{U}(\mathsf{Out}_{\mathcal{H}}) = t_0]$.
Therefore, $Pr[\mathsf{FwdSplr}(x_0, t_0) = (t_0, [\,])] = 1$.

We can now provide implementations of oracles in the generic simulator. We recall that $L_{\mathcal{S}}(o_i)$ denotes the list of all tuples starting with o_i appearing in $L_{\mathcal{S}}$ and that if (o_i, q, y) is one of these tuples, $L_{\mathcal{S}}(o_i, q)$ 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* $L_{\mathcal{S}} \in \mathsf{Xch}^*$*, and the initial memory* $\bar{m}$ *of a system containing the simulator is chosen so that* $\bar{m}.L_{\mathcal{S}} = [\,]$*. Moreover,* o_{piv} *is implemented as follows:*

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

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

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

If we let c *be a constant bounding the time necessary to search in* $L_{\mathcal{S}}$*, the implementation of* o_{piv} *is* (k_s, t_s)*-bounded, where* $k_s(\mathcal{H}) = 0$*, and* $t_{\mathsf{FwdSplr}} + t_{\mathsf{PathFinder}}(\mathsf{Card}(L_{\mathcal{S}})) + c$*. 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 indifferentiability 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 are 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},

$\mathsf{Imp}_{\mathbb{O}_{ant}}(\mathcal{H})(x, L_{\mathcal{H}}, L_{\mathcal{S}}) =$
if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  $(a^f, Q) := L_{\mathcal{H}}(x)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
else
  $l := \mathsf{init}(x)$;
  $p := \mathsf{piv}(x)$;
  $(x_1, \ldots, x_l) := \Theta(x)$;
  $(o^1, q^1) := \mathsf{H}_1(x_1)$;
  if $q^1 \in \mathsf{dom}(L_{\mathcal{S}}(o^1))$ then
    $(a^1, lbl) := L_{\mathcal{S}}(o^1, q^1)$;
    $Q := [(o^1, q^1, a^1, lbl)]$;
  else let $a^1 \leftarrow \mathcal{U}(\mathsf{Out}_{o^1})$ in
    $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^1, q^1, a^1, Inv)$;
    $Q := [(o^1, q^1, a^1, Inv)]$;
  endif
  for $j = 2$ to $p - 1$ do
    $(o^j, q^j) := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $Q := Q : (o^j, q^j, a^j, lbl)$;
    else let $a^j \leftarrow \mathcal{U}(\mathsf{Out}_{o^j})$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, Inv)$;
      $Q := Q : (o^j, q^j, a^j, Inv)$;
    endif
  endfor
  for $j = p$ to l do
    $(o^j, q^j) := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, max(PVis, lbl))$;
      $Q := Q : (o^j, q^j, a^j, max(PVis, lbl))$;
    else let $a^j \leftarrow \mathcal{U}(\mathsf{Out}_{o^j})$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, PVis)$;
      $Q := Q : (o^j, q^j, a^j, PVis)$;
    endif
  endfor
  $a^f := \mathsf{H}_{post}(x, a^p, [Q]_{j>p})$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
endif

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

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

For tuple $(o_i, q, y, lbl) \in L_{\mathcal{S}}$, $L_{\mathcal{S}}(o_i, q)$ 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 $\mathbb{O}_{ant}$

- *a finite set of vertices* $V \subseteq \mathsf{Xch}$,
- *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* $q = \mathsf{H}_1(x_1)$*;*
  2) *for* $j \geq 2$, $((o^{j-1}, q, a), x_j, (o^j, q', a')) \in E$ *implies* $q' = \mathsf{H}_j(x_j, (o^{j-1}, q, a))$.
- $\mathcal{V}$ *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), $q = \mathsf{H}_1(x_1)$ *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* $q' = \mathsf{H}_j(x_j, (o^{j-1}, q, a))$ *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 $\mathsf{dom}(\mathcal{V}_{init}) = \emptyset$ 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* $v_0 \xrightarrow{l_1} v_1 \xrightarrow{l_2} \cdots \xrightarrow{l_n} v_n$ *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*

$[x_1, \ldots, x_L]$ *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* $L = \mathsf{init}(x)$. *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 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 Γ 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 Γ to each intermediate state created during the iteration.

Definition V.3. *Given a characteristic graph* CG, v *is a* collision vertex, *denoted* $v \in \mathsf{CollVertex}(CG)$, *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 belongs 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.* $\mathsf{CollVertex}(CG') - \mathsf{CollVertex}(CG) \neq \emptyset$.
- 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,* $\mathcal{V}'(v) = Vis$ *and* $\mathcal{V}'(v') \neq Vis$, *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 indifferentiability 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* $\mathbb{A} \in \mathsf{Adv}(k, t)$,

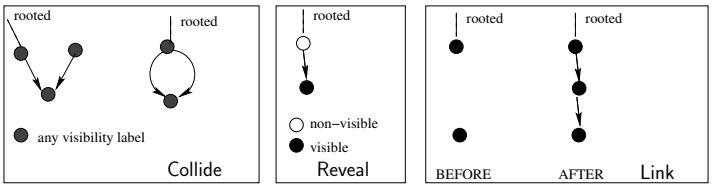
- $\mathbb{S}$ *is* (k_s, t_s)*-bounded with* $k_s(\mathcal{H}) = 1$, *and* $t_s = t_{\mathsf{FwdSplr}} + t_{\mathsf{PathFinder}}(k' + 1) + c$, *where* $k' = \sum_{o \in \mathbb{O}} k(o)$.
- *the indifferentiability advantage is bounded by the probability of inconsistency events:*

$$\mathsf{Indiff}(\mathcal{H}, \mathcal{RO}(\mathbb{O}), \mathbb{S}, \leq) \\ Pr[\mathbb{A}|\mathbb{O}_{ant} : \mathsf{F}_{\mathsf{Collide} \vee \mathsf{Reveal} \vee \mathsf{Link}}]$$

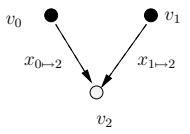
where t_{FwdSplr} *and* $t_{\mathsf{PathFinder}}$ *respectively bound the execution time of forward sampler* FwdSplr *and path-finder* PathFinder *used in the simulator.*

Sketch of Proof. We present a proof sketch of this theorem in CIL developed in [Dau11], [BDK+10]. An extended version of the proof can be found in appendix. To relate real and simulated oracle systems, we use $\mathbb{O}_{ant}$ and another intermediate system $\mathbb{F}wd\mathbb{S}pl$, 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.



a - Inconsistency Events



b-Vertices Involved in $\mathsf{F}_{\mathsf{Collide}}$

Figure 5.

$$\frac{\text{determinization}}{(\mathcal{H}^{\mathbb{O}},\mathbb{O}) \sim_0 \mathbb{O}_{ant}} \qquad \frac{\begin{cases} \text{Bisimulations up to condition } \phi \text{ relate } \mathbb{O}_{ant} \text{ to } \mathbb{F}wd\mathbb{S}pl \\ \Pr[\phi] \leq \Pr[\mathsf{F}_{\mathsf{Collide}\vee\mathsf{Reveal}\vee\mathsf{Link}}] = \epsilon \end{cases}}{\mathbb{O}_{ant} \sim_\epsilon \mathbb{F}wd\mathbb{S}pl} \qquad \frac{\text{determinization}}{\mathbb{F}wd\mathbb{S}pl \sim_0 (\mathcal{RO}(\mathcal{H}),\mathbb{S}^{\mathcal{RO}(\mathcal{H})})}$$

$$(\mathcal{H}^{\mathbb{O}},\mathbb{O}) \sim_\epsilon (\mathcal{RO}(\mathcal{H}),\mathbb{S}^{\mathcal{RO}(\mathcal{H})})$$

Figure 6. Sketch Of Proof

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 $\mathbb{F}wd\mathbb{S}pl$, 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 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 $\mathbb{F}wd\mathbb{S}pl$ 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 $\mathbb{F}wd\mathbb{S}pl$ with a better advantage that the probability that Reveal, Collide or Link occurs.

Finally, we justify the perfect indistinguishability between $\mathbb{F}wd\mathbb{S}pl$ and $(\mathcal{RO}(\mathcal{H}),\mathbb{S}^{\mathcal{RO}(\mathcal{H})})$ by a determinization argument: indeed, as the hash oracle in $\mathbb{F}wd\mathbb{S}pl$ 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_{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 indifferentiability 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 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 $\mathsf{F}_{\mathsf{Collide}}$, $\mathsf{F}_{\mathsf{Reveal}}$ and $\mathsf{F}_{\mathsf{Link}}$ 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 $\mathsf{F}_{\mathsf{Collide}}$ 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 RootCollide capture the event that $v_0 = v_{root}$ and v_1 is created such that the collision happens. Then necessarily, there exist $j, x_{0\mapsto 2}, x_{1\mapsto 2}$ such that $\mathsf{H}_1(x_{0\mapsto 2}) = \mathsf{H}_j(x_{1\mapsto 2}, (\_, q_1, a_1))$ since they both equal the query part of v_2. Furthermore, WkCollide is verified when $v_0 \neq v_{root}$ is $(\_, q_0, a_0)$, and v_1 is created; i.e. when there exist $(j, x_{1\mapsto 2})$ and $(j', x_{0\mapsto 2})$ such that $\mathsf{H}_j(x_{1\mapsto 2}, (_, q_1, a_1)) = \mathsf{H}_{j'}(x_{0\mapsto 2}, (_, q_0, a_0))$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 $\mathsf{F}_{\mathsf{Collide}}$ implies $\mathsf{F}_{\mathsf{RootCollide}\vee\mathsf{WkCollide}}$. We define

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 $\mathsf{F}_{\mathsf{Collide}\vee\mathsf{Reveal}\vee\mathsf{Link}}$ is implied by $\mathsf{F}_{\mathsf{WkLink}} \vee \mathsf{F}_{\mathsf{RootCollide}\vee\mathsf{WkCollide}} \vee \mathsf{F}_{\mathsf{Reveal}\wedge\neg\mathsf{WkLink}\wedge\neg\mathsf{Collide}}$.

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. Its precise implementation is provided in appendix A.

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

When $\mathsf{F}_{\mathsf{RootCollide}}$ occurs, there exists v_1 such that $\mathsf{H}_1(x_{0\mapsto 2}) = \mathsf{H}_j(x_{1\mapsto 2}, (\mathcal{F}, q_1, a_1))$ for some labels $x_{0\mapsto 2}$ and $x_{1\mapsto 2}$ 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 $\mathsf{F}_{\mathsf{WkLink}}$ happens at the ℓ-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 $\mathsf{H}_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$ is at most $\frac{\ell-1}{2^c}$. Summing on ℓ, it results in a bound of $\frac{k_{tot}(k_{tot}-1)}{2^{c+1}}$.

Finally, we bound the probability of $\mathsf{F}_{\mathsf{Reveal}\wedge\neg\mathsf{WkLink}\wedge\neg\mathsf{Collide}}$ 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 $\neg\mathsf{WkLink}$, necessarily, v_1 is created before v_2. Realizing Reveal means that $\mathsf{H}_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$. Since we assume $\neg\mathsf{Collide}$, 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* $\mathbb{A} \in \mathsf{Adv}(k, t)$,

$$\mathsf{Indiff}(Sponge, \mathcal{RO}(\mathcal{F}), \mathcal{RO}(Sponge), \mathbb{S}) \leq \frac{k_{tot}^2}{2^c} + \frac{k(\mathcal{F})}{2^c}$$

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+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+04]. For sake of completeness, we describe it in appendix A. 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 $\mathsf{FwdSplr}$ to sample uniformly the s missing bits to compute the result of the pivot query $y^{\mathsf{piv}(x)}$, and outputs the concatenation of t with these bits. Probability computations are very similar to the sponge case, see appendix A for details.

THEOREM VI.2. *We consider the* $ChopMD_s$ *construction. For an adversary* $\mathbb{A} \in \mathsf{Adv}(k, t)$,

$$\mathsf{Indiff}(ChopMD_s, \mathcal{F}, \mathcal{RO}(ChopMD_s), \mathbb{S}) \leq \frac{k_{tot}^2}{2^n} + \frac{k(\mathcal{F})}{2^s}$$

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

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 $\mathcal{O}(\frac{(\mathfrak{L}.k_{tot})^2}{2^s})$, which is the same magnitude.

Later, Maurer and Tessaro show in [MT07] that using a prefix-free padding function yields a bound of $\mathcal{O}(\frac{(\mathfrak{L}*k_{tot})^2}{2^n})$. 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 $y' = \mathsf{First}_n(q)$ 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+10] with a formalization of modes of operations - *overlayers* - and proven our theorem in this logic.

REFERENCES

[AHMP10] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. Sha-3 proposal blake. Submission to NIST (Round 3), 2010.

[BCCM+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+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 Pfitzmann, 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 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.

[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+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 CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2004.

[FLS+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+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 CryptologyConference, 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.

Oracle $ChopMD_s$
$\mathsf{In}(ChopMD_s) = \{0,1\}^{\leq 2^{64}}$,
$\mathsf{Out}(ChopMD_s) = \{0,1\}^{n-s}$
$\mathsf{Imp}(ChopMD_s)^{\mathcal{F}}(x, L_{chop}) =$
if $x \in \mathsf{dom}(L_{chop})$ then
return $L_{chop}(x)$
else
$l := \lceil |x|/r \rceil$;
$w := x||10^{l*r-|x|-1}$;
$(w_1, \ldots, w_l) := (w[1,r], \ldots, w[r(l-1)+1, rl])$;
$a^0 := 0^n$;
for $j = 1$ to l do
$q^j := a^{j-1}||w_j$;
let $a^j \leftarrow \mathcal{F}(q^j)$ in
endfor
$a^f := \mathsf{First}_{n-s}(a^l)$;
$L_{chop} := L_{chop}.(x, a^f)$;
return a^f
endif

Figure 7. ChopMD Implementation

APPENDIX

A. Additional Details Of Proofs Of Applications

1) The Sponge Construction: The implementation of the forward sampler used by the simulator for the sponge construction is as follows:

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

2) The ChopMD Construction: The implementation of the Chop construction can be found in figure 7. This section details the computations leading to the bound in the theorem instantiated with the ChopMD construction.

If WkCollide happens at the ℓ-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 probability less than $\frac{\ell-1}{2^n}$. Summing on ℓ, it results in a bound of $\frac{k_{tot}(k_{tot}-1)}{2^{n+1}}$.

In turn, if $\mathsf{F}_{\mathsf{RootCollide}}$ holds, then there exist $j, x_{0\mapsto 2}, x_{1\mapsto 2}$ such that $\mathsf{H}_1(x_{0\mapsto 2}) = \mathsf{H}_j(x_{1\mapsto 2}, (\mathcal{F}, q_1, a_1))$. Necessarily, $j > 1$, otherwise $v_1 = v_2$. It yields $a_1 = 0^n$, occuring with probability less than $\frac{k_{tot}}{2^n}$.

Then we bound of the probability that $\mathsf{F}_{\mathsf{WkLink}}$ occurs by $\frac{k_{tot}(k_{tot}-1)}{2^{n+1}}$. Indeed, if it happens at the ℓ-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 $\mathsf{H}_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$ is bounded by $\frac{\ell-1}{2^n}$. We conlude by summing on ℓ.

Finally, when $\mathsf{F}_{\mathsf{Reveal}\wedge\neg\mathsf{WkLink}\wedge\neg\mathsf{Collide}}$ 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, $\neg$WkLink implies v_1 is created before v_2. Since Reveal happens, $\mathsf{H}_j(x_j, (\mathcal{F}, q_1, a_1)) = q_2$. Because of $\neg$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 $\frac{k_{tot}^2}{2^n} + \frac{k(\mathcal{F})}{2^s}$.

B. CIL Rules Used in The Proof

1) Determinization:

Definition A.1. *Let $\mathbb{O}$ and $\mathbb{O}'$ be compatible oracle systems.* $\mathbb{O}$ determinizes $\mathbb{O}'$ *by distribution* $\gamma : \mathsf{M}_{\mathbb{O}} \to \mathcal{D}(\mathsf{M}''_{\mathbb{O}})$*, written* $\mathbb{O} \leq_{\mathrm{det},\gamma} \mathbb{O}'$*, iff firstly* $\mathsf{M}_{\mathbb{O}} \times \mathsf{M}''_{\mathbb{O}} = \mathsf{M}_{\mathbb{O}'}$*, secondly, there exists* $\bar{m}''_{\mathbb{O}}$ *such that* $(\bar{m}_{\mathbb{O}}, \bar{m}''_{\mathbb{O}}) = \bar{m}'_{\mathbb{O}}$ *and* $\gamma(\bar{m}_{\mathbb{O}}) = \delta_{\bar{m}''_{\mathbb{O}}}$*, and lastly for all* $o \in \mathsf{N}_{\mathbb{O}}$*,* $q \in \mathsf{In}(o)$*,* $a \in \mathsf{Out}(o)$*,* $m_1, m_2 \in \mathsf{M}_{\mathbb{O}}$ *and* $m''_2 \in \mathsf{M}''_{\mathbb{O}}$*:*

$$\Pr[\gamma(m_2) = m''_2]\, p_1 = \sum_{m''_1 \in \mathsf{M}''_{\mathbb{O}}} \Pr[\gamma(m_1) = m''_1]\, p_2(m''_1)$$

where:

$$\begin{aligned} p_1 &= Pr[\mathsf{Imp}_{\mathbb{O}}(o)(q, m_1) = (a, m_2)] \\ p_2(m''_1) &= Pr[\mathsf{Imp}_{\mathbb{O}'}(o)(q, (m_1, m''_1)) = (a, (m_2, m''_2))] \end{aligned}$$

We start with the proof of a lemma formally linking probabilities of partial executions in both systems. We define a projection function π from $\mathbb{A} \mid \mathbb{O}'$-partial executions to $\mathbb{A} \mid \mathbb{O}$-partial executions by extending the projection from $\mathsf{M}_{\mathbb{O}} \times \mathsf{M}''_{\mathbb{O}}$ to $\mathsf{M}_{\mathbb{O}}$ to executions.

Informally, we can foresee that if we consider a partial execution η in $\mathbb{A} \mid \mathbb{O}$ finishing with state m, we have to gather in a set all partial executions in $\mathbb{A} \mid \mathbb{O}'$ finishing with state (m, m'') for a given m'' and projecting to η. Then, from the equation imposed for one query by the definition of determinization, we can extrapolate that the set of $\mathbb{A} \mid \mathbb{O}'$-partial executions weighs the same probability as trace η multiplied by

the probability that m is mapped to m'' by γ. This is proven by the lemma below.

LEMMA A.1. *Let $\mathbb{O}$ and $\mathbb{O}'$ be such that $\mathbb{O} \leq_{\det,\gamma} \mathbb{O}'$, and let η be a partial $\mathbb{O}$-execution: $\eta = (\bar{m}_{\mathbb{O}}, \bar{m}_{\mathbb{A}}) \xrightarrow{act_1} \ldots \xrightarrow{(o,q,a)} (m, m_a)$. For every $\mathbb{O}$-adversary $\mathbb{A}$ and every $m'' \in \mathsf{M}''_{\mathbb{O}}$:*

$$Pr(\mathbb{A}|\mathbb{O} : \eta) Pr[\gamma(m) = m''] = \sum_{\substack{\eta' | \pi(\eta') = \eta \\ Last(\eta') = ((m,m''),m_a)}} Pr(\mathbb{A}|\mathbb{O}' : \eta')$$

where τ' is any partial $\mathbb{O}'$-execution.

LEMMA A.2. *We consider two compatible oracle systems $\mathbb{O}$ and $\mathbb{O}'$.*

$$\frac{\mathbb{O} \leq_{\det,\gamma} \mathbb{O}'}{\mathbb{O} \sim_0 \mathbb{O}'} \textit{ I-Det}$$

Proof: The previous lemma A.1 immediately results in $\Pr(\mathbb{A}|\mathbb{O} : E) = \Pr(\mathbb{A}|\mathbb{O}' : E \circ \pi)$ for every $\mathbb{O}$-event E and adversary $\mathbb{A}$. In turn, this equality yields our result. ∎

2) Forward Bisimulation-up-to: The idea behind this bisimulation is the following: states are grouped in classes according to an equivalence relation R. This relation is relevant if given two states in a same class, they offer the same possibility of evolution with the same probabilities.

Definition A.2. *Let $\phi \subseteq \mathsf{Xch} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be a step-predicate and let $R \subseteq \mathsf{M}_{\mathbb{O}+\mathbb{O}'} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be an equivalence relation. $\mathbb{O}$ and $\mathbb{O}'$ are in forward bisimulation up to ϕ, written $\mathbb{O} \equiv_{R,\phi} \mathbb{O}'$, iff $\bar{m}\ R\ \bar{m}'$, and for all $m_1 \xrightarrow{(o,q,a)}_{>0} m_2$ and $m_3 \xrightarrow{(o,q,a)}_{>0} m_4$ such that $m_1\, R\, m_3$, the following properties hold:*

- stability: *if $m_2\, R\, m_4$ then*

$$\phi((o,q,a), m_1, m_2) \quad \Leftrightarrow \quad \phi((o,q,a), m_3, m_4)$$

- compatibility: *if $\phi((o,q,a), m_1, m_2)$, then*

$$Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1) \in (a, \mathcal{C}(m_2))] = Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_3) \in (a, \mathcal{C}(m_2))]$$

where $\mathcal{C}(m_2)$ is the equivalence class of m_2 under R, and

$$Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1) \in (a, \mathcal{C}(m_2))] = \sum_{m\, R\, m_2} Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1) \in (a, m)].$$

we consider fixed $\mathbb{O}$, $\mathbb{O}'$, R and ϕ satisfying the above definition. The relation defined on states can be lifted to (partial) executions quite easily.

Definition A.3. *Let η and η' be two partial executions of $\mathbb{A} \mid \mathbb{O}$ or $\mathbb{A} \mid \mathbb{O}'$ of length k such that $\eta = (m_0, m_a^0) \xrightarrow{act_1} (m_1, m_a^1) \xrightarrow{act_2} \ldots \xrightarrow{act_k} (m_k, m_a^k)$ and $\eta' = (m'_0, m_a^0) \xrightarrow{act_1} (m'_1, m_a^1) \xrightarrow{act_2} \ldots \xrightarrow{act_k} (m'_k, m_a^k)$. They are said to be* in relation by R, *denoted $\eta\ R\ \eta'$, iff $m_i\ R\ m'_i$ for all $i \in [0..k]$.*

The equivalence class of η is defined by $\mathcal{C}(\eta) = \{\eta' \mid \eta\ R\ \eta'\}$. The $\mathbb{O}$-class of η, denoted $\mathcal{C}_{\mathbb{O}}(\eta)$, is the intersection with $\mathbb{O}$-traces of $\mathcal{C}(\eta)$. Its probability is given by:

$$Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta)] = \sum_{\eta' \in \mathcal{C}_{\mathbb{O}}(\eta)} Pr[\mathbb{A} \mid \mathbb{O} : \eta']$$

A similar definition can be written for $\mathbb{O}'$.

Consider a state $m_1 \in \mathsf{M}_{\mathbb{O}}$, in relation with $m_3 \in \mathsf{M}_{\mathbb{O}'}$. According to the definition of bisimulation up to, if we perform one step for which ϕ holds from m_1 and its successor or m_3 and its successor, then we can move to the same equivalence classes of states with the same probability. Say we have gone through such a step: $m_1 \xrightarrow{act} m_2$ and $m_3 \xrightarrow{act} m_4$, and $m_2\ R\ m_4$. We can iterate the same reasoning on m_2 and m_4. Informally, we can anticipate that if we perform a series of steps for which ϕ holds, it yields equivalence classes on executions with same probabilities in $\mathbb{O}$ and $\mathbb{O}'$.

LEMMA A.3. *Let η be a partial execution of $\mathbb{A} \mid \mathbb{O}$ of length k such that $\eta = (m_0, m_a^0) \xrightarrow{act_1} (m_1, m_a^1) \xrightarrow{act_2} \ldots \xrightarrow{act_k} (m_k, m_a^k)$.*

$$Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta)] = \Pi_{i=1}^{k} Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}((m_{i-1}, m_a^{i-1}) \xrightarrow{act_i} (m_i, m_a^i))]$$

where

$$Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}((m_{i-1}, m_a^{i-1}) \xrightarrow{act_i} (m_i, m_a^i))] = \sum_{\tilde{m}_i\ R\ m_i} \Pr[(m_{i-1}, m_a^{i-1}) \xrightarrow{act_i} (\tilde{m}_i, m_a^i)]$$

We can now show that given related partial executions for which ϕ holds at every step, we have equal probabilities to make a next step *not* verifying ϕ when interacting with $\mathbb{A} \mid \mathbb{O}$ as when interacting with $\mathbb{A} \mid \mathbb{O}'$.

LEMMA A.4. *Let η be a partial execution of $\mathbb{A} \mid \mathbb{O}$ of length k such that ϕ holds for each of its steps: $\eta = (m_0, m_a^0) \xrightarrow{act_1} (m_1, m_a^1) \xrightarrow{act_2} \ldots \xrightarrow{act_k} (m_k, m_a^k)$ and $\forall i = 1..k, \phi(x_i, m_{i-1}, m_i)$*

- *Let $\sigma = (m_k, m_a^k) \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})$ be a step. Let $\eta' = \eta \cdot \sigma$. If $\phi(x_{k+1}, m_k, m_{k+1})$ then $Pr[\mathbb{A} \mid \mathbb{O} : \mathcal{C}_{\mathbb{O}}(\eta')] = Pr[\mathbb{A} \mid \mathbb{O}' : \mathcal{C}_{\mathbb{O}'}(\eta')]$*
- $Pr[\mathbb{A} \mid \mathbb{O} : \eta_0 \cdot \sigma_0 \wedge (\eta_0\ R\ \eta) \wedge \neg\phi(\sigma_0)] = Pr[\mathbb{A} \mid \mathbb{O}' : \eta_0 \cdot \sigma_0 \wedge (\eta_0\ R\ \eta) \wedge \neg\phi(\sigma_0)]$

LEMMA A.5. *We consider two compatible oracle systems $\mathbb{O}$ and $\mathbb{O}'$ such that $\mathbb{O} \equiv_{R,\varphi} \mathbb{O}'$. Then* $\Pr[\mathbb{A} \mid \mathbb{O} : \mathtt{R} = \mathsf{true} \wedge \mathsf{G}_\phi] = \Pr[\mathbb{A} \mid \mathbb{O}' : \mathtt{R} = \mathsf{true} \wedge \mathsf{G}_\phi]$.

3) Backwards Bisimulation-up-to: Forward bisimulation-up-to is powerful, but ill-conceived to capture arguments meant to tamper with values computed in past steps instead of changing the way we compute values in the current step or in future steps. This justifies the notion of backwards bisimulation-up-to, formally defined as follows.

Definition A.4. *Let $R \subseteq \mathsf{M}_{\mathbb{O}+\mathbb{O}'} \times \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ be an equivalence relation and ϕ be a predicate. $\mathbb{O}$ and $\mathbb{O}'$ are in* backwards bisimulation with R up to ϕ *iff the initial states is alone in their equivalence class and for all $m_1 \overset{(o,q,a)}{\longrightarrow}_{>0} m_2$ and m_2' such that $m_2 \, R \, m_2'$ we have:*

- *stability on an equivalence class: for all $m_1' \in \mathsf{M}_{\mathbb{O}+\mathbb{O}'}$ such that $m_1' \overset{(o,q,a)}{\longrightarrow}_{>0} m_2'$ and $m_1' \; R \; m_1$,*

$$\phi((o,q,a), m_1, m_2) \Leftrightarrow \phi((o,q,a), m_1', m_2')$$

- *backwards compatibility: if $\phi((o,q,a), m_1, m_2)$*

$$\Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, \mathcal{C}(m_1)) = (a, m_2)] = \Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, \mathcal{C}(m_1)) = (a, m_2')]$$

where $\mathcal{C}(m_1)$ is the equivalence class of m_1 under R, and

$$\Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, \mathcal{C}(m_1)) = (a, m_2)] = \sum_{m_1' \; R \, m_1} \Pr[\mathsf{Imp}_{\mathbb{O}+\mathbb{O}'}(o)(q, m_1') = (a, m_2)]$$

We define a projection $\mathcal{A}dv\mathcal{T}$ on partial executions which erases all oracle memories (only exchanges and adversarial memories are left). It defines the set of *partial adversarial traces*, for which we often use meta-variable α. The fundamental property of backwards bisimulation is captured by the following lemma. It mostly states that the probability that a partial execution ends up in states (m, m_a) is constant on equivalence classes: it does not depend on the actual class representative m.

LEMMA A.6. *Let α be a partial adversarial trace of length k $\alpha = m_a^0 \xrightarrow{act_1} m_a^1 \xrightarrow{act_2} \ldots \xrightarrow{act_k} m_a^k$. Then, for all exchange act_{k+1}, all adversary memory m_a^{k+1} and for all $m_{k+1}, m_{k+1}' \in \mathsf{M}_{\mathbb{O}}$ such that $m_{k+1} \; R \; m_{k+1}'$:*

$$\sum_{\substack{\eta \in \mathsf{PExec}(\mathbb{A}|\mathbb{O}) \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})] = \sum_{\substack{\eta \in \mathsf{PExec}(\mathbb{A}|\mathbb{O}) \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_{k+1}} (m_{k+1}', m_a^{k+1})))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_{k+1}} (m_{k+1}', m_a^{k+1})]$$

And if $m_{k+1} \in \mathsf{M}_{\mathbb{O}}, m_{k+1}' \in \mathsf{M}_{\mathbb{O}'}$ such that $m_{k+1} \; R \; m_{k+1}'$:

$$\sum_{\substack{\eta \in \mathsf{PExec}(\mathbb{A}|\mathbb{O}) \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})))}} \Pr[\mathbb{A} \mid \mathbb{O} : \eta \xrightarrow{x_{k+1}} (m_{k+1}, m_a^{k+1})] = \sum_{\substack{\eta \in \mathsf{PExec}(\mathbb{A}|\mathbb{O}') \mid \mathcal{A}dv\mathcal{T}(\eta) = \alpha \\ \mathsf{G}_\phi(\mathcal{T}(\eta \xrightarrow{x_{k+1}} (m_{k+1}', m_a^{k+1})))}} \Pr[\mathbb{A} \mid \mathbb{O}' : \eta \xrightarrow{x_{k+1}} (m_{k+1}', m_a^{k+1})]$$

LEMMA A.7. *The following probabilities coincide*

$$\Pr[\mathbb{A} \mid \mathbb{O} : \mathtt{R} = \mathsf{true} \wedge \mathsf{G}_\phi] = \Pr[\mathbb{A} \mid \mathbb{O}' : \mathtt{R} = \mathsf{true} \wedge \mathsf{G}_\phi]$$

4) Composition of Bisimulations: If we were to use bisimulations one after the other with the same condition ϕ, we would count twice the same bad simulation event, augmenting artificially by a factor of two the indistinguishability bound in our conclusion. To tackle this problem, we propose the following rule, which follows from A.5 and A.7.

LEMMA A.8. *The following rule is sound:*

$$\frac{\mathbb{O} :_\epsilon \mathsf{F}_{\neg\phi} \qquad \mathbb{O} \equiv^b_{R,\phi} \mathbb{O}'' \qquad \mathbb{O}'' \equiv_{R',\phi} \mathbb{O}'}{\mathbb{O} \sim_\epsilon \mathbb{O}'} \textit{ I-2-Bis}$$

C. Proof Of The Theorem

In this section we provide a detailed proof in CIL for the generic theorem V.1. The trees summing up the proof can be found in figure 8. Here is the outline of our reasoning. The proof starts with a layered oracle system implemented as in the definition, which we must relate to the anticipating system $\mathbb{O}_{ant}$. The formal relation between the real setting and the anticipating system is mostly one of determinization, though it seems easier to introduce intermediate systems $\mathbb{Q}_0$ and $\mathbb{Q}_1$ to write the underlying distribution properly. This is developped in C1 and corresponds to the left tree in figure 8.

Then, the anticipating system is progressively transformed into a system $\mathbb{F}wd\mathbb{S}pl$ closer to the simulated setting. We show that the probability to distinguish between $\mathbb{O}_{ant}$ and $\mathbb{F}wd\mathbb{S}pl$ is bounded by the same bound as our theorem: $\Pr[\mathbb{A}|\mathbb{O}_{ant} : \mathsf{F}_{\mathsf{Collide} \vee \mathsf{Reveal} \vee \mathsf{Link}}]$. To justify this, we successively present a series of modified systems, from $\mathbb{Q}_2$ to $\mathbb{Q}_4$, and the formal link

existing between one and the next, before being able to conclude in C6. This corresponds to the middle tree in figure 8.

Eventually, we argue that $\mathbb{F}wd\mathbb{S}pl$ is determinized by the simulated setting. The global conclusion finally follows from transitivity of the indistinguishabiltiy relation.

1) Relation Between $(\mathcal{H}^{\mathbb{O}}, \mathbb{O})$ and $\mathbb{O}_{ant}$ - Left Tree: We provide the specification of the intermediate system $\mathbb{Q}_0$ in figure 9. It mostly consists in the anticipating system but the anticipation part. Namely the visibility labels are added and computed dynamically, and the branching is modified in o_{piv}, but $\mathcal{H}$ is not called by o_{piv} to anticipate post-pivot queries. Moreover, a list $Pivot$ is added to the memory, to collect detected pivot queries, their answers and the value of x output by the pathfinder PathFinder. Memories of $(\mathcal{H}^{\mathbb{O}}, \mathbb{O})$ contain lists L_{o_i}, $L_{\mathcal{H}}$. Memories of $\mathbb{Q}_0$ contain a shared table $L_{\mathcal{S}}$ collecting all tuples of the form (o_i, q, y, lbl), a list $L_{\mathcal{H}}$ and list $Pivot$. This system is in bisimulation up to with the real setting, for relation R defined as follows. Memories m and m' are in relation iff they are equal when they belong to the same memory space and if $m \in \mathsf{M}_{(\mathcal{H}^{\mathbb{O}}, \mathbb{O})}$ and $m' \in \mathsf{M}_{\mathbb{Q}_0}$:

- $(x, a^f, Q) \in m.L_{\mathcal{H}}$ iff $(x, a^f, Q) \in m'.L_{\mathcal{H}}$, but the order of appearance might not be the same,
- lists $(m.L_{o_i})_i$ and list $m'.L_{\mathcal{S}}$ contain the same queries and answers, which we formalize as:
  (1.) $\forall (q, y) \in m.L_{o_i}$, there exists a label lbl such that $(o_i, q, y, lbl) \in m'.L_{\mathcal{S}}$;
  (2.) $\forall (o_i, q, y, lbl) \in m'.L_{\mathcal{S}}$, $(q, y) \in m.L_{o_i}$.

We then have $(\mathcal{H}^{\mathbb{O}}, \mathbb{O}) \equiv_{R, \mathsf{true}} \mathbb{Q}_0$.

We now define a second intermediate system, $\mathbb{Q}_1$, which is similar to $\mathbb{Q}_0$ but for the four lines starting with † in the implementation of o_{piv}, which are replaced by:

$$
\begin{array}{l}
\mathsf{let}\ t \leftarrow \mathcal{H}(x)\ \mathsf{in} \\
(o_{\mathsf{piv}}, q, y) : L := \Pi_3(L_{\mathcal{H}}(x))_{j \geq \mathsf{piv}(x)}; \\
L_{\mathcal{S}} := L_{\mathcal{S}}.(L, Vis);
\end{array}
$$

To apply a determinization rule, we should separate $L_{\mathcal{S}}$ into two tables $L_{\mathcal{S}}$ and $L^{ant}_{\mathcal{S}}$. However, we bypass this step and just provide the distribution γ induced by a memory on $L^{ant}_{\mathcal{S}}$, table of anticipated queries. System $\mathbb{Q}_0$ determinizes $\mathbb{Q}_1$ for distribution γ for which we provide a constructive definition:

$$
\begin{array}{rl}
\gamma(m) = & L^{ant}_{\mathcal{S}} := [\,]; \\
& \mathsf{for}\ q\ \mathsf{in}\ \mathsf{dom}(Pivot)\ \mathsf{do} \\
& \mathsf{let}\ (y, x) \leftarrow Pivot(q)\ \mathsf{in} \\
& \mathsf{let}\ t \leftarrow \mathsf{Imp}_{\mathbb{Q}_0}(\mathcal{H})(x)\ \mathsf{in} \\
& L := \Pi_3(\mathsf{Last}(L_{\mathcal{H}}(x))); \\
& L_{\mathcal{S}} := L_{\mathcal{S}}.([L]_{j > \mathsf{piv}(x)}, Vis); \\
& L^{ant}_{\mathcal{S}} := L^{ant}_{\mathcal{S}}.([L]_{j > \mathsf{piv}(x)}, Vis); \\
& \mathsf{endfor} \\
& \mathsf{return}\ L^{ant}_{\mathcal{S}} - m.L_{\mathcal{S}}
\end{array}
$$

Finally, the justification of the step from $\mathbb{Q}_1$ to $\mathbb{O}_{ant}$ is again a perfect bisimulation relation R' induced by equality on lists $L_{\mathcal{H}}$ and $L_{\mathcal{S}}$.

2) Redrawing Some Invisible Vertices: The idea behind this step is to allow the oracles to redraw new images for values of which the image has already been used, or in other words to resample some vertices. When can such a resampling be a problem for coherence of the simulation? The idea is to preserve the structure of the input characteristic graph during oracle calls. Of course, if the image we consider is visible or partially visible, we do not redraw it. Furthermore, even when the vertex we want to modify is invisible, we have to be careful. The idea is that we have to preserve paths existing in the input graph. To this end, we introduce a new terminology: a vertex v' is one of the *next neighbors* of a vertex v in graph CG iff there exists an edge $(v, \_, v')$ between v and v'. The set of next neighbors of v in graph CG is denoted $\mathsf{Next}(v, CG)$. Every time we change an invisible vertex into another vertex, we want to modify its next neighbors so that the same edges still exist between them. This is doable only if such neighbors are non-visible. Besides, in case one of the next neighbors is a collision vertex, redrawing suppresses the collision and changes the structure of the graph. This is also a case we want to exclude.

Formally, we define a function ReSamp taking as input a query (o, q) and a memory m such that query (o, q) corresponds to a vertex (o, q, y) in m. The function outputs a boolean corresponding to whether we can redraw vertex (o, q, y) in memory m. The characteristic graph associated to m is $\Gamma(m) = (v_{root}, V, E, \mathcal{V})$. Function $\mathsf{ReSamp} : \mathsf{Que} \times \mathsf{M}_{\mathbb{O}_{ant}} \rightarrow \mathsf{Bool}$ maps $((o, q), m)$ to:

$$
\begin{cases}
\mathsf{true\ if}\ \mathcal{V}((o, q, y)) = Inv, \\
\quad \mathsf{and}\ \mathsf{Next}((o, q, y), \Gamma(m)) \cap \mathcal{V}^{-1}(Vis) = \emptyset \\
\quad \mathsf{and}\ \mathsf{Next}((o, q, y), \Gamma(m)) \cap \mathsf{CollVertex}(\Gamma(m)) = \emptyset \\
\mathsf{false\ otherwise.}
\end{cases}
$$

In particular, we emphasize that for all values corresponding to partially visible and visible vertices, ReSamp outputs false.

To form up again the paths existing in the input

$$\frac{\text{Left tree} \qquad \text{Middle tree} \qquad \dfrac{(\mathcal{U}(\mathcal{H}), \mathbb{S}^{\mathcal{U}(\mathcal{H})}) \leq_{\det,\gamma'} \mathbb{F}wd\mathbb{S}pl}{(\mathcal{U}(\mathcal{H}), \mathbb{S}^{\mathcal{U}(\mathcal{H})}) \sim_0 \mathbb{F}wd\mathbb{S}pl}}{(\mathcal{H}^{\mathbb{O}}, \mathbb{O}) \sim_\epsilon (\mathcal{U}(\mathcal{H}), \mathbb{S}^{\mathcal{U}(\mathcal{H})})}\ \text{I-Det}$$

$$\text{Left tree: } \text{I-Bis}\ \frac{(\mathcal{H}^{\mathbb{O}}, \mathbb{O}) \equiv_{R,\mathsf{true}} \mathbb{Q}_0 \qquad \dfrac{\mathbb{Q}_0 \leq_{\det,\gamma} \mathbb{Q}_1}{\mathbb{Q}_0 \sim_0 \mathbb{Q}_1}\ \text{I-Det} \qquad \dfrac{\mathbb{Q}_1 \equiv_{R',\mathsf{true}} \mathbb{O}_{ant}}{\mathbb{Q}_1 \sim_0 \mathbb{O}_{ant}}\ \text{I-Bis}}{(\mathcal{H}^{\mathbb{O}}, \mathbb{O}) \sim_0 \mathbb{O}_{ant}}$$

$$\text{Middle tree: } \text{UR}\ \frac{\mathbb{O}_{ant} :_\epsilon E \qquad \mathsf{F}_{\neg\phi} \Rightarrow E}{\mathbb{O}_{ant} :_\epsilon \mathsf{F}_{\neg\phi}} \qquad \frac{\mathbb{O}_{ant} \equiv^b_{R'',\phi} \mathbb{Q}_2 \qquad \mathbb{Q}_2 \equiv_{=,\phi} \mathbb{F}wd\mathbb{S}pl}{\mathbb{O}_{ant} \sim_\epsilon \mathbb{F}wd\mathbb{S}pl}\ \text{I-2-Bis}$$

where $E = \mathsf{F}_{\mathsf{Collide} \vee \mathsf{Reveal} \vee \mathsf{Link}}$

Figure 8. Trees Of The Proof Of The Generic Theorem

graph, we define a function named $\mathsf{Stitch} : \mathsf{Xch} \times \{Inv, PVis\} \times \mathsf{M}_{\mathbb{O}_{ant}} \to \mathsf{M}_{\mathbb{O}_{ant}}$, which takes as input a (possibly resampled) vertex $(o, q, \tilde{y})$, a visibility label for this latter and a memory m and outputs a new memory m'. If (o, q, y) appears in the memory m for some y, Stitch modifies the memory so that $(o, q, \tilde{y})$ replaces (o, q, y) with the visibility label given in input of Stitch and next neighbors of the vertex (o, q, y) in $\Gamma(m)$ become next neighbors of the new vertex $(o, q, \tilde{y})$ in $\Gamma(m')$ (with the same edges). Thus, paths existing in the input graph exist in the output graph too.

Formally, if (o, q, y) appears in the memory m and $(o, q, \tilde{y})$ is the new vertex, Stitch outputs m' computed as follows. To build $m'.L_{\mathcal{S}}$, we start with $m'.L_{\mathcal{S}} = m.L_{\mathcal{S}}$ and then proceed in the following way. For all edges $((o, q, y), l, (o', q', y'))$ in $\Gamma(m)$ where $(o, q', y') \in \mathsf{Next}((o, q, y), \Gamma(m))$, if j is an index such that $q' = \mathsf{H}_j(l, (o, q, y))$, then we let $\tilde{q} = \mathsf{H}_j(l, (o, q, \tilde{y}))$. Then, if $(o', \tilde{q})$ does not appear in $m'.L_{\mathcal{S}}$ yet, (o', q', y', lbl) is removed from $m'.L_{\mathcal{S}}$ and $(o', \tilde{q}, y', lbl)$ is added to $m'.L_{\mathcal{S}}$. If $(o', \tilde{q})$ already appears in $m'.L_{\mathcal{S}}$, we do not modify it. Finally, we remove $(o, q, y, \_)$ from $m'.L_{\mathcal{S}}$ and replace it by $(o, q, \tilde{y}, lbl)$, where lbl is given in input of Stitch. List $m'.L_{\mathcal{H}}$ is then built out of $m.L_{\mathcal{H}}$ by rebuilding the third component of every triple it contains: given $(x, a^f, Q) \in m.L_{\mathcal{H}}$, (x, a^f, Q') is put in $m'.L_{\mathcal{H}}$, where Q' is the list of calls necessary to compute $\mathcal{H}(x)$ in $m'.L_{\mathcal{S}}$.

To write our new system $\mathbb{Q}_2$, we introduce an auxiliary procedure Adjust. It takes as input a query (o, q) and a visibility label lbl, resamples the vertex if it is possible and modifies the lists with Stitch, which adds the query and answer to list $L_{\mathcal{S}}(o)$ with the desired visibility label. Its implemntation is as follows:

```
Adjust ((o,q), lbl, m) =
       if q ∈ dom(L_S(o)) then
        if ReSamp((o,q), m) then
         let a ← U(o) in
         m' := Stitch((o,q,a), lbl, m);
        else (a, lbl') := L_o(q);
         L_S := L_S.(o,q,a,max(lbl,lbl'));
        endif
       else let a ← U(o) in
        L_S := L_S.(o,q,a,lbl);
       endif
       return (o, q, L_S(o,q))
```

Then, we can define the implementation of oracle $\mathcal{H}$ in the adjusted system as in figure 10, while both other oracles remain implemented as in $\mathbb{O}_{ant}$. The claim proven above justifies the existence and unicity of related adjusted states when ϕ holds.

To formalize our proof step, we use a relation of backwards bisimulation. Two states are in relation R'' iff they yield graphs with the same structure. Notice that we cannot turn a vertex into a collision vertex when we resample it: the fact that a collision occurs in a vertex depends only on its query part and we only change the answer. However, there is a possibility when we apply Stitch that we change the structure of the characteristic graph. Namely, we can stumble upon a preexisting vertex by computing a value $\tilde{q}$ which already corresponds to a vertex. The set of values $\tilde{y}$ such that it happens is:

$$\mathsf{PbSet}((o, q), m) = \{\tilde{y} \in \mathsf{Out}(o) \mid \exists j\ s.t.\ \tilde{q} = \mathsf{H}_j(l, (o, q, \tilde{y})) \in \mathsf{dom}(m.L_{\mathcal{S}}(o^j))\}$$

$\mathsf{Imp}_{\mathbb{O}_{ant}}(\mathcal{H})(x) =$
if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  $(a^f, Q) := L_{\mathcal{H}}(x)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
else
  $l := \mathsf{init}(x)$;
  $p := \mathsf{piv}(x)$;
  $(x_1, \ldots, x_l) := \Theta(x)$;
  $(o^1, q^1) := \mathsf{H}_1(x_1)$;
  if $q^1 \in \mathsf{dom}(L_{\mathcal{S}}(o^1))$ then
    $(a^1, lbl) := L_{\mathcal{S}}(o^1, q^1)$;
    $Q := [(o^1, q^1, a^1, lbl)]$;
  else let $a^1 \leftarrow \mathcal{U}(o^1)$ in
    $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^1, q^1, a^1, Inv)$;
    $Q := [(o^1, q^1, a^1, Inv)]$;
  endif
  for $j = 2$ to $p - 1$ do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $Q := Q : (o^j, q^j, a^j, lbl)$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, Inv)$;
      $Q := Q : (o^j, q^j, a^j, Inv)$;
    endif
  endfor
  for $j = p$ to l do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, max(PVis, lbl))$;
      $Q := Q : (o^j, q^j, a^j, max(PVis, lbl))$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, PVis)$;
      $Q := Q : (o^j, q^j, a^j, PVis)$;
    endif
  endfor
  $a^f := \mathsf{H}_{post}(x, a^p, [Q]_{j>p})$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
endif

If $o_i \neq o_{\mathsf{piv}}$:
$\mathsf{Imp}_{\mathbb{O}_{ant}}(o_i)(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_i))$ then
  $(y, \_) := L_{\mathcal{S}}(o_i, q)$;
else let $y \leftarrow \mathcal{U}(o_i)$ in
endif
$L_{\mathcal{S}} := L_{\mathcal{S}}.(o_i, q, y, Vis)$;
return y

$\mathsf{Imp}_{\mathbb{Q}_0}(o_{\mathsf{piv}})(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|Vis)$ then
  $(y, Vis) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
elsif $\mathsf{PathFinder}(q, SG) = (\mathsf{true}, x, List)$ then
  † if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|PVis, Inv)$ then
  † $(y, \_) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
  † else let $y \leftarrow \mathcal{U}(o_{\mathsf{piv}})$ in
  † endif
  $Pivot := Pivot.(q, y, x)$;
elsif $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|PVis, Inv)$ then
  $(y, \_) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
else let $y \leftarrow \mathcal{U}(o_{\mathsf{piv}})$ in
endif
return y
  $L_{\mathcal{S}} := L_{\mathcal{S}}.(o_{\mathsf{piv}}, q, y, Vis)$;

Figure 9. Implementations Of $\mathbb{Q}_0$

Moreover, as the number of neighbors of a resampled vertex can potentially be modified by stitching, we impose that it is equal in two states in relation.

Formally, we impose the conditions:

$m \; R'' \; m'$ iff there exist $n \geq 0$ and a list $[(o_1, q_1, a_1), \ldots, (o_n, q_n, a_n)]$ of distinct vertices and labels, such that, if we denote $m^0 = m$ and $m^n = m'$:

- For all $i = 1..n$, $m^i = \mathsf{Stitch}((o_i, q_i, a_i), Inv, m^{i-1})$.
- For all $i = 1..n$, $\mathsf{ReSamp}((o_i, q_i), m^{i-1})$ or $q_i \notin m^{i-1}.L_{\mathcal{S}}(o_i)$.
- For all $i = 1..n$, $a_i \notin \mathsf{PbSet}((o_i, q_i), m^{i-1})$.
- For all i, if y_i is the image of q_i by o_i in state m^{i-1}, then $\mathsf{Card}(\mathsf{Next}((o_i, q_i, y_i), \Gamma(m^{i-1}))) = \mathsf{Card}(\mathsf{Next}((o_i, q_i, a_i), \Gamma(m^i)))$, i.e. the stitch operation conserves the number of neighbors of the resampled vertex.

To be able to apply rule $I - 2 - Bis$, we need a common set of conditions ϕ for backward and forward bisimulation relations. Therefore, we choose for ϕ the conjunction of every condition that we need to require in the next steps determining $\mathbb{F}wd\mathbb{S}pl$. To do so, we express two conditions on the execution of an exchange $m_1 \xrightarrow{xch} m_2$, one is a condition on the characteristic graph from which we start (this is ϕ_1) and one is a condition on what happens during the exchange execution (this is ϕ_2).

The first condition expresses that the input charac-

$\mathsf{Imp}_{\mathbb{Q}_2}(\mathcal{H})(x) =$ if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  $(a^f, Q) := L_{\mathcal{H}}(x)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
else
  $l := \mathsf{init}(x)$;
  $p := \mathsf{piv}(x)$;
  $(x_1, \ldots, x_l) := \Theta(x)$;
  $q^1 := \mathsf{H}_1(x_1)$;
  if $q^1 \in \mathsf{dom}(L_{\mathcal{S}}(o^1))$ then
    $(a^1, lbl) := L_{\mathcal{S}}(o^1, q^1)$;
    $Q := [(o^1, q^1, a^1, lbl)]$;
  else let $a^1 \leftarrow \mathcal{U}(o^1)$ in
    $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^1, q^1, a^1, Inv)$;
    $Q := [(o^1, q^1, a^1, Inv)]$;
  endif
  for $j = 2$ to $p - 1$ do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $Q := Q : (o^j, q^j, a^j, lbl)$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, Inv)$;
      $Q := Q : (o^j, q^j, a^j, Inv)$;
    endif
  endfor
  for $j = p$ to l do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    let $(o^j, q^j, a^j, lbl) \leftarrow \mathsf{Adjust}((o^j, q^j), PVis)$ in
    $Q := Q : (o^j, q^j, a^j, lbl)$;
  endfor
  $a^f := \mathsf{H}_{post}(x, a^p, [Q]_{j>p})$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
endif

Figure 10. Implementation Of $\mathcal{H}$ In System $\mathbb{Q}_2$

teristic graph exhibits no collision or non-resamplable vertex. This is naturally formalized as

$$\phi_1(m) = \begin{cases} \mathsf{CollVertex}(\Gamma(m)) = \emptyset \\ \forall i, \forall (o, q, a) \in (m.L_{\mathcal{S}}(o_i) | Inv), \\ \qquad \mathsf{ReSamp}((o, q), \Gamma(m)) = \mathsf{true} \end{cases}$$

The second condition captures that neither Collide nor Reveal happen during the execution of the exchange, using the function mapping execution of exchanges to graph sequences defined in the previous section. We also impose that no query to oracles $o_i \neq o_{\mathsf{piv}}$ is labeled partially visible (this is P_1) and that all hash queries, if they have a matching pivot query that is visible, are not fresh hash queries (this is P_2).

$$\phi_2(xch, m_1, m_2) = \mathsf{G}_{\neg \mathsf{Collide} \wedge \neg \mathsf{Reveal}}(\mathsf{StTr}(m_1 \xrightarrow{xch} m_2)) \\ \wedge P_1(m_1) \wedge P_2(m_1, m_2)$$

where

$$\begin{cases} P_1(m) \text{ is } (xch = (o_i, q, y) \wedge o_i \neq o_{\mathsf{piv}} \\ \qquad \wedge q \in \mathsf{dom}(m.L_{\mathcal{S}}(o_i))) \Rightarrow \mathcal{V}(q) \neq PVis \\ P_2(m, \tilde{m}) \text{ is } \quad (xch = (\mathcal{H}, x, a^f) \wedge \\ \quad \Pi_3(\tilde{m}.L_{\mathcal{H}})[\mathsf{piv}(x)] \in \mathsf{dom}(m.L_{\mathcal{S}}(o_{\mathsf{piv}}) | Vis)) \Rightarrow \\ \qquad\qquad x \in \mathsf{dom}(m.L_{\mathcal{H}}) \end{cases}$$

We now let $\phi(xch, m_1, m_2) = \phi_1(m_1) \wedge \phi_2(xch, m_1, m_2)$. We must show that R'' is a relation of backwards bisimulation up to ϕ for our oracle system. We start by showing the following useful claim.

Claim. Given $m_1 \xrightarrow{xch}_{>0} m_2$, and a state m_2' such that $m_2' \; R'' \; m_2$, if $\phi(xch, m_1, m_2)$, there exists a unique state m_1' such that $m_1 \; R'' \; m_1'$ and $m_1' \xrightarrow{xch}_{>0} m_2'$. Moreover, the same number of vertices are added in the graph $\Gamma(m_2)$ w.r.t. $\Gamma(m_1)$ and in the graph $\Gamma(m_2')$ w.r.t. $\Gamma(m_1')$.

Proof: We know that $m_2 \; R'' \; m_2'$. Hence there exist $n \geq 0$ and a list $[(o_1, q_1, a_1), \ldots, (o_n, q_n, a_n)]$ of distinct vertices such that, if we denote $m^0 = m_2$ and $m^n = m_2'$, we have:

- For all i between 1 and n, we have $m^i = \mathsf{Stitch}((o_i, q_i, a_i), Inv, m^{i-1})$.
- For all i, $\mathsf{ReSamp}((o_i, q_i), m^{i-1})$ or $q_i \notin m^{i-1}.L_{\mathcal{S}}(o_i)$.
- For all i, $a_i \notin \mathsf{PbSet}((o_i, q_i), m^{i-1})$.
- For all i, if y_i is the image of q_i by o_i in state m^{i-1}, then $\mathsf{Card}(\mathsf{Next}((o_i, q_i, y_i), \Gamma(m^{i-1}))) = \mathsf{Card}(\mathsf{Next}((o_i, q_i, a_i), \Gamma(m^i)))$.

Let us define the following candidate for m_1':

$$m_1' = \mathsf{Stitch}((o_1, q_1, a_1), Inv, \ldots \\ \mathsf{Stitch}((o_n, q_n, a_n), Inv, m_1) \ldots)$$

The state m_1' defined satisfies $m_1 \; R'' \; m_1'$. Indeed, without loss of generality, we can assume that the (o_i, q_i) are distinct. The stitching application has no effect on a state m if its first argument (o_i, q_i, a_i) is such that (o_i, q_i) does not satisfy $q_i \in \mathsf{dom}(m.L_{\mathcal{S}}(o_i))$.

Let us show now that every time a new vertex is added to $m_1.L_{\mathcal{S}}$ during the execution leading to m_2, it is added in any state in relation with m_1 leading to m_2' too.

Suppose that we reason about an exchange xch with an oracle o_i. First, we argue that related states coincide on visible vertices, so in particular on visible parts of the domain of list $L_{\mathcal{S}}(o_i)$. Moreover, the only invisible

queries that can be asked without realizing Reveal are vertices directly linked to the root. If ϕ_1 holds, none of these vertices can be a collision vertex. Therefore, there is no possibility that their query part be resampled as neighbors of another vertex. Consequently, if an invisible query is asked and ϕ holds, it is in the domain of $L_{\mathcal{S}}(o_i)$ for all related memories. Furthermore, in case we ask a partially visible query, either $o_i \neq o_{\mathsf{piv}}$ and it breaks P_1, or it has to be visibly rooted, otherwise Reveal becomes true. Hence, since it has a visible (previous) neighbor and no other previous neighbor (otherwise it is a collision vertex), it cannot be resampled as a next neighbor of some vertex. As a result, it is in the domain of all related memories.

Suppose now that we reason on an exchange xch with $\mathcal{H}$. The trick is to notice that our equivalence relation is built so that the same paths exist in related states. Consequently, if at step j, we meet the first query resulting in the addition of a new vertex in $\Gamma(m_1)$, then it is also the first query resulting in the addition of a vertex in any related memory, or there would exist a rooted path in one graph and not the other. Furthermore, once we start adding vertices during the execution, we have to draw new vertices until the end, or we contradict ϕ by either creating a collision vertex or realizing Reveal. The conclusion follows. ■

Let us first check stability, i.e. that given $m_1 \xrightarrow{xch} m_2$, and m_2' such that $m_2' \; R'' \, m_2$, all states $\tilde{m_1}$ in relation with m_1 such that $\tilde{m_1} \xrightarrow{xch} m_2'$ are such that $\phi(xch, m_1, m_2)$ iff $\phi(xch, \tilde{m_1}, m_2')$. This follows from the claim: if $\phi(xch, m_1, m_2)$, then there is one possibility of state $\tilde{m_1}$, it is m_1'. Moreover, $\phi_1(m_1')$ holds: no collision vertex or non-resamplable vertex can be created. This allows us to say that $\phi_1(m_2)$ holds iff $\phi_1(m_2')$ holds. Therefore, if Reveal happens or a collision vertex is created, then it is in both cases. This justifies stability of $\mathsf{G}_{\neg\mathsf{Collide}\wedge\neg\mathsf{Reveal}}$. Concerning P_1, it only deals with input states. Visible vertices are equal in related states, so we only need to justify that there cannot exist a vertex which is partially visible in one state and invisible in the other. In fact, P_1 is not a stable property, but $\neg\mathsf{Reveal} \wedge \phi_1 \wedge P_1$ is. If $\neg\mathsf{Reveal} \wedge \phi_1$ holds for an exchange, then the only invisible queries that an adversary can perform are directly linked to the root, otherwise Reveal happens, and linked only to the root, since ϕ_1 holds. Since we do not resample the root, the set of invisible queries not breaking $\neg\mathsf{Reveal} \wedge \phi_1$ coincide in related states. Therefore, if visible and invisible queriable vertices coincide, P_1 holds for all or none of the states in relation. Finally, stability of P_2 follows from the visibility property imposed on the pivot: it has the same value in m_1 and m_1', so does $L_{\mathcal{H}}$. Stability follows.

We have to verify compatibility. We consider states m_1, m_2 and m_2' and an exchange $xch = (o, q, a)$ such that $m_1 \xrightarrow{xch}_{>0} m_2$ and $\phi(xch, m_1, m_2)$. The claim proves that there is only one state m_1' such that $m_1' \xrightarrow{xch}_{>0} m_2'$ and that executions starting in states m_1 and m_1' lead to the same number of draws. It yields the equality between probabilities:

$$\Pr[\mathbb{A} \mid \mathbb{O}_{ant} : m_1 \xrightarrow{xch}_{>0} m_2] = \Pr[\mathbb{Q}_3 : m_1' \xrightarrow{xch}_{>0} m_2']$$

Then, we deduce from the one-to-one mapping between m_1 and m_1' that it yields:

$$\Pr[\mathbb{A} \mid \mathbb{O}_{ant} : \mathcal{C}(m_1) \xrightarrow{xch}_{>0} m_2] = \Pr[\mathbb{Q}_2 : \mathcal{C}(m_1') \xrightarrow{xch}_{>0} m_2']$$

3) Replacing Adjust *by Simple Sampling:* We keep the same overall implementations but change the implementation of Adjust into:

Adjust' $((o, q), lbl) =$
let $a \leftarrow \mathcal{U}(o)$ in
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o))$ then
  $m' := \mathsf{Stitch}((o, q, a), lbl, m)$;
else $L_{\mathcal{S}} := L_{\mathcal{S}}.(o, q, a, lbl)$;
endif
return (o, q, a)

In other words, we redraw a value for q, no matter whether it is resamplable, and do not take care of drawing it such that it does not create collisions. This yields a system we name $\mathbb{Q}_3$.

This step is formalized using a bisimulation up to ϕ, with as a relation the equality of states. ϕ is obviously stable for this relation. Now let us check compatibility. Given that only the implementation of $\mathcal{H}$ possibly resamples vertices, the simulation is imperfect during an execution of $\mathcal{H}(x)$ (not necessarily called directly). It can happen if we resample a non-resamplable vertex.

Let v be the first vertex posing a simulation problem during an execution of $\mathcal{H}$.

- If v has been resampled whereas it was partially visible, it means v belongs to the pivot and post-pivot queries of another hash input x'. Necessarily the paths of x and x' meet in some vertex v' (not necessarily distinct of v), which is a collision vertex. The execution of $\mathcal{H}(x)$ realizes Collide at the moment of the query for v'.

- If v has been resampled whereas it was visible and v is not the pivot then Reveal happens: the visibility label of the pivot is partially visible, so that sequence of labels has to increase.
- If v has been resampled whereas it was a visible pivot query matching x, then P_2 is broken.

We conclude that $\mathbb{Q}_2 \equiv_{=,\phi} \mathbb{Q}_3$.

4) Changing Oracles in $\mathsf{N}_{\mathbb{O}}$*:* In this step, we modify the implementation of the oracles in $\mathsf{N}_{\mathbb{O}}$ assuming that pivot queries are on the one hand always detected when queried directly, and on the other hand always asked before any of their matching post-pivot queries. It gives us a new system $\mathbb{Q}_4$, for which the implementations are provided in figure 11.

If the first assumption holds, we can safely simplify the end of the implementation of o_{piv} by replacing the test of belonging to $(L_{\mathcal{S}}(o_{\mathsf{piv}})|PVis, Inv)$ by that of belonging to $(L_{\mathcal{S}}(o_{\mathsf{piv}})|Inv)$. If the second assumption holds, no partially visible query should be directly asked to an oracle $o_i \neq o_{\mathsf{piv}}$. Indeed, the pivot query being queried on before implies that all post-pivot queries become visible vertices. We thus modify the implementation of $o_i \neq o_{\mathsf{piv}}$ by just checking if a query already belongs to $\mathsf{dom}(L_{\mathcal{S}}(o_i)|Inv, Vis)$ before drawing an answer.

The formal justification of this step is that $\mathbb{Q}_3 \equiv_{=,\phi} \mathbb{Q}_4$. Indeed, the simulation is perfect except when:

- o_{piv} is queried on a partially visible vertex, but does not branch in the path-finder branch, meaning the vertex is non-visibly meaningfully rooted. Yet, it is meaningfully rooted since it is partially visible. This is captured by Reveal.
- During an execution of o_i, if we redraw a new answer to a partially visible query, but then P_1 is broken.

5) Changing $\mathcal{H}$*:* In this last step, we define a system $\mathbb{F}wd\mathbb{S}pl$ (see in figure 12) and replace the series of uniform sampling of the pivot and post-pivot vertices, followed by the computation of a^f, by the sampling of a^f and the execution of the forward sampler algorithm. According to the hypotheses we have formulated on this latter, both implementations yield equal distributions on the lists $(L_{\mathcal{H}}, L_{\mathcal{S}})$ as soon as the forward sampler does not stumble on queries for which vertices already exist in the graph. This is taken care of since neither Collide nor Reveal happens. It follows that $\mathbb{Q}_4 \equiv_{=,\phi} \mathbb{F}wd\mathbb{S}pl$

6) Conclusion Of The Tree In The Middle: We start by providing details about the application of rule $I - 2 - Bis$. In the last three transformations, we have created systems $\mathbb{Q}_2$ to $\mathbb{F}wd\mathbb{S}pl$, and such that $\mathbb{Q}_2 \equiv_{=,\phi} \mathbb{Q}_3$, $\mathbb{Q}_3 \equiv_{=,\phi} \mathbb{Q}_4$ and $\mathbb{Q}_4 \equiv_{=,\phi} \mathbb{F}wd\mathbb{S}pl$. From

If $o_i \neq o_{\mathsf{piv}}$:
$\mathsf{Imp}_{\mathbb{Q}_4}(o_i)(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_i)|Inv, Vis)$ then
  $(y, \_) := L_{\mathcal{S}}(o_i, q)$;
else let $y \leftarrow \mathcal{U}(o_i)$ in
endif
$L_{\mathcal{S}} := L_{\mathcal{S}}.(o_i, q, y, Vis)$;
return y

$\mathsf{Imp}_{\mathbb{Q}_4}(o_{\mathsf{piv}})^{\mathcal{H}}(q) =$
if $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|Vis)$ then
  $(y, Vis) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
elsif $\mathsf{PathFinder}(q, SG) = (\mathsf{true}, x, List)$ then
  let $t \leftarrow \mathcal{H}(x)$ in
  $(o_{\mathsf{piv}}, q, y) : L := \Pi_3(L_{\mathcal{H}}(x))_{j \geq \mathsf{piv}(x)}$;
  $L_{\mathcal{S}} := L_{\mathcal{S}}.((o_{\mathsf{piv}}, q, y, Vis) : (L, Vis))$;
elsif $q \in \mathsf{dom}(L_{\mathcal{S}}(o_{\mathsf{piv}})|Inv)$ then
  $(y, Inv) := L_{\mathcal{S}}(o_{\mathsf{piv}}, q)$;
else let $y \leftarrow \mathcal{U}(o_{\mathsf{piv}})$ in
endif
$L_{\mathcal{S}} := L_{\mathcal{S}}.(o_{\mathsf{piv}}, q, y, Vis)$;
return y

Figure 11. Implementations Of Oracles In $\mathbb{Q}_4$

these statements, we can deduce that $\mathbb{Q}_2 \equiv_{=,\phi} \mathbb{F}wd\mathbb{S}pl$. As $\mathbb{Q}_2$ is expressed as an adjusted system of $\mathbb{O}_{ant}$, we can apply rule $I - 2 - Bis$.

Furthermore, we want to justify that $\mathsf{F}_{\neg\phi}$ yields that eventually, Collide, Reveal or Link happens, i.e. $\mathsf{F}_{\neg\phi} \Rightarrow \mathsf{F}_{\mathsf{Collide} \vee \mathsf{Reveal} \vee \mathsf{Link}}$. To do so, we prove that $\neg P_1$ and $\neg P_2$ imply that Reveal or Link have happened. Concerning P_1, if when querying $o_i \neq o_{\mathsf{piv}}$ on q, $q \in \mathsf{dom}(m_1.L_{\mathcal{S}}(o_i))$ is part of a partially visible vertex, then this latter is meaningfully rooted. If it is not visibly meaningfully rooted, then we can conclude that Reveal has happened. Otherwise, if all queries on the path from the root to our queried vertex are visible, since it is a post-pivot query, but still tagged with a partially visible label, it means that the matching pivot was not visibly meaningfully rooted at the time of its query. Consequently, we are sure that at some point, a query was issued to one of the o_i's to link two chains of visible vertices, i.e. Link has happened.

Finally, for property P_2, if a fresh query on x is issued to $\mathcal{H}$ with a pivot already visible, it means that the pivot has been directly queried for, but that at the time of query, it was not visibly rooted (otherwise $\mathcal{H}(x)$ would have been called). Similarly to the previous event, we can show that either all queries before the pivot are visible, and at some point Link has happened, or there exists an invisible query on the path from the root to

$\mathsf{Imp}_{\mathbb{F}wd\mathbb{S}pl}(\mathcal{H})(x) =$ if $x \in \mathsf{dom}(L_{\mathcal{H}})$ then
  $(a^f, Q) := L_{\mathcal{H}}(x)$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q)$;
  return a^f
else
  $l := \mathsf{init}(x)$;
  $p := \mathsf{piv}(x)$;
  $(x_1, \ldots, x_l) := \Theta(x)$;
  $(o^1, q^1) := \mathsf{H}_1(x_1)$;
  if $q^1 \in \mathsf{dom}(L_{\mathcal{S}}(o^1))$ then
    $(a^1, lbl) := L_{\mathcal{S}}(o^1, q^1)$;
    $Q := [(o^1, q^1, a^1, lbl)]$;
  else let $a^1 \leftarrow \mathcal{U}(o^1)$ in
    $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^1, q^1, a^1, Inv)$;
    $Q := [(o^1, q^1, a^1, Inv)]$;
  endif
  for $j = 2$ to $p - 1$ do
    $q^j := \mathsf{H}_j(x_j, (o^{j-1}, q^{j-1}, a^{j-1}))$;
    if $q^j \in \mathsf{dom}(L_{\mathcal{S}}(o^j))$ then
      $(a^j, lbl) := L_{\mathcal{S}}(o^j, q^j)$;
      $Q := Q : (o^j, q^j, a^j, lbl)$;
    else let $a^j \leftarrow \mathcal{U}(o^j)$ in
      $L_{\mathcal{S}} := L_{\mathcal{S}}.(o^j, q^j, a^j, Inv)$;
      $Q := Q : (o^j, q^j, a^j, Inv)$;
    endif
  endfor
  $q^p := \mathsf{H}_p(x_p, (o^{p-1}, q^{p-1}, a^{p-1}))$;
  let $a^f \leftarrow \mathcal{U}_{\mathcal{H}}$ in
  let $(a^p, Q') \leftarrow \mathsf{FwdSplr}(x, a^f)$ in
  $L_{\mathcal{S}} := L_{\mathcal{S}}.((o_{\mathsf{piv}}, q^p, a^p, PVis) : (Q', PVis))$;
  $L_{\mathcal{H}} := L_{\mathcal{H}}.(x, a^f, Q : (o_{\mathsf{piv}}, q^p, a^p) : Q')$;
  return a^f
endif

Figure 12. Implementation Of $\mathcal{H}$ In System $\mathbb{F}wd\mathbb{S}pl$

the pivot, and Reveal holds.

This concludes the discussion about the middle tree.

7) Determinization of $\mathbb{F}wd\mathbb{S}pl$ to Obtain The Simulated System: As we did previously, we abuse a little the determinization rule and only provide the distribution yielded by a memory on anticipated queries in $L_{\mathcal{S}}$, which we name $L_{\mathcal{S}}^{ant}$. To build possible anticipated components of state out of a state $m = (L_{\mathcal{S}}, L_{\mathcal{H}})$ of the simulated system, we have to generate the list of queries matching every pair (x, a^f) in $L_{\mathcal{H}}$ and to tag them with visibility labels. Given the first pair (x, a^f), this can be done by executing the implementation $\mathsf{Imp}_{\mathbb{F}wd\mathbb{S}pl}(\mathcal{H})$ given as input x and the list $L_{\mathcal{S}}$ where every vertex has been deemed visible. It provides us with a new table $L_{\mathcal{S}}$, on which to iterate what we have just done with the following pairs in list $L_{\mathcal{H}}$. This provides us with a constructive definition for a distribution γ' such that $(\mathcal{U}(\mathcal{H}), \mathbb{S}^{\mathcal{U}(\mathcal{H})}) \leq_{\mathrm{det},\gamma'} \mathbb{F}wd\mathbb{S}pl$:

$\gamma'(m.L_{\mathcal{S}}, m.L_{\mathcal{H}}) = m'.L_{\mathcal{S}} := (m.L_{\mathcal{S}}, Vis)$;
  $L_{\mathcal{S}}^{ant}, m'.L_{\mathcal{H}} := [\,]$;
  for x in $m.L_{\mathcal{H}}$ do
   let $a^f \leftarrow \mathsf{Imp}_{\mathbb{F}wd\mathbb{S}pl}(\mathcal{H})(x, m')$ in
   $L_{\mathcal{S}}^{ant} := L_{\mathcal{S}}^{ant}.\Pi_3(\mathsf{Last}(m'.L_{\mathcal{H}}))$;
  endfor
  return $L_{\mathcal{S}}^{ant} - m.L_{\mathcal{S}}$