On a Probabilistic Appproach to the Security Analysis of Cryptographic Hash Functions

G. Laccetti, G. Schmid
Istituto per il Calcolo e le Reti ad Alte Prestazioni
Sezione di Napoli
Complesso Universitario M.S. Angelo
Via Cintia, 80126 Napoli

September 30, 2004

Abstract

In this paper we focus on the three basic security requirements for a cryptographic hash function, commonly referred as preimage, second preimage and collision resistance. We examine these security requirements in the case of attacks which do not take advantage on how the hash function is computed, expressing them as success probabilities of suitable randomized algorithms. We give exact mathematical expressions for such resistance indices, and obtain their functional behaviour in relation to the amount of uniformity in the hash function outcomes. Our work provides a mathematical framework for the study of cryptographic hash functions, which enable us to give proofs for some prevailing beliefs.
Key words. Hash functions, Brute force attacks, resistance.

1 Introduction

Cryptographic hash functions play a basic role in modern cryptography: they are especially employed for digital signatures and message authentication, but are used for many other cryptographic applications, including data protection and key derivation [10]. Hash functions take a message of variable length as input and produce a fixed-length string as output, referred to as *hash-code* or simply *hash* of the input message. The basic idea of cryptographic hash functions is the use of hash-codes as compact and non-ambiguous images of messages, from which the latter cannot be deduced. The term non-ambiguous refers to the fact that the hash-code can be used as it were uniquely identifiable with the source message. For that reason, a cryptographic hash is also called *digital fingerprint* or *digest* of the message. Both non-ambiguity of hash-codes and preimages non-computability are important requirements for the security

of communication schemes which use an hash function to provide message authentication without confidentiality [8, 14]; nevertheless, those properties are impossible to be achieved in a strict sense for practical hash functions.

First, denoted by $h : X \to Y$ a candidate hash function, it should be clear that the only way to guarantee about non-computability of preimages $x \in X$ from their digests $y = h(x)$ would be to assign y in a truly random way, i.e. to define h as a really random function; but that results in a useless function for virtually all cryptographic applications.

Furthermore, for a domain X and a range Y with cardinalities $|X| > |Y|$, the function $h : X \to Y$ is many-to-one, implying the existence of at least a *collision*, i.e. a pair of input messages with the same digest. Since in common applications $|X| \gg |Y|$, not only "perfect" non-ambiguity of digests is impossible to achieve, but it can be difficult to design an hash function good with such respect, too.

Thus, both for evaluating the fitness of existing hash functions for cryptographic purposes and for designing new ones, it turns out the importance of assessing how much computational effort is required by an hypothetical adversary to find preimages and/or collisions for the hash function h under deployment or investigation.

A rigorous approach should clearly be based on a computational model which enables to specify both the kind of attacks that could be mounted against h and the amount of resources available to the adversaries for such attacks. For a general model, which aims to describe a *generic* hash function h, it makes no sense to try to encompass all the different kinds of attack that could be mounted against h. Two radically different ways have been followed in modern cryptography to overcome the above difficulty:

(i) the adoption of complexity-theoretic settings, where assumptions do not regard specific strategies that an adversary may use with respect to a given hash function, but only refer to upper bounds on his/her computational ability;

(ii) the imposition of suitable restrictions on the hash function and/or on the kind of attacks that could be mounted against it.

On the framework (i) of complexity theory rely both the probabilistic Turing machine model, which nowadays founds modern cryptography (for a comprehensive, rigorous treatment, see [4]), and the random access machine (RAM) model, that was introduced in the context of hashing in [2], and used in [10] as an alternative approach to the first one. Both these models allow formal definitions for the three notions of *resistance* which were recognized (and only informally defined) as basic security properties for an hash function in previous works [9, 11], and which are often referred to in literature as *preimage*, *second preimage* and *collision resistance* [8].

A typical approach which pertains to (ii) is to assume that the hash function appears, from the attacker point of view, as a random function whose outcomes

are uniformly distributed in its range. This is the case for the *algorithmic-independent* attack approach [8] or equivalently, as pointed out in [15], for the so called *random oracle model* [1]. This approach allows to derive helpful probabilistic results; however it *postulates* an output behavior for the hash function, rather than to provide means to *evaluate* it [15].

A recent work [3] takes the opposite direction with respect to such a typical approach: first, the authors introduce the notion of *balance* for an hash function, as a measure of its "closeness" to have a uniformly distributed output; secondly, they provide estimates of the success-rate of the birthday attack, and the expected number of trial to find a collision, as a function of the balance of the hash function being attacked; thirdly, they estimate experimentally the balance of the restriction to various length inputs of the popular hash function SHA-1. As pointed out by the authors, their work is intended to provide analytical tools that contribute toward the goal of better understanding the security of existing hash functions or building new ones.

In the present work we follow the new point of view adopted in [3], purposely referring to the concept of resistance in a different way than in textbooks on modern cryptography, i.e. as a quantifiable measure of the difficulty to solve a computational problem (related to a given hash function) by using a well-specified class of algorithms. On the basis of the preliminary work [15], we formalize the notion of (randomized) *brute force attack* and generalize some previous results [15] to hash functions with arbitrary distributed outcomes, giving exact mathematical expressions for the success probabilities of three algorithms designed to solve the preimage, second preimage and collision problem, respectively. Since the above algorithms are optimal, in the sense that no algorithms in the same class exist with larger success probabilities, it is natural to regard these probabilities as measures of resistance for a generic hash function with respect to brute force attacks.

We then analyze how those resistance indices depend from the hash function being considered, obtaining a characterization of their functional behavior in relation to the "amount of uniformity" in the *probability distribution* of that function. We make use of the notion of *majorization* [5, 7] to measure the amount of uniformity in hash functions outcomes. Majorization is indeed a preordering relationship in the set of N dimensional real arrays, such that arrays are ordered with respect to the amount of differences between their components, the biggest elements being those with no differences at all (i.e. all equal components). Our results prove that, with respect to such preordering relationship, collision resistance is an increasing function in the set of all possible probability distribution for a range N hash function. The opposite is true for preimage resistance. Thus, as conjectured by some authors, the less uniform is an hash function, the more effective results a birthday attack mounted against it. Conversely, preimage attacks becomes more effective as uniformity increases. At a first glance, the above results seem to indicate the impossibility to find the best hash function with respect to both preimage and birthday attacks. However, an analysis of the relationships among the three given notions of resistance demonstrates that collision resistance is the most stringent security requirement for

an hash function exposed to a brute force attack. Again, the previous result has been so far only conjectured in literature, usually assuming that the hash output approximates a uniform random variable.

Finally, we examine the notion of *reduction* among preimage and collision searching problems in the context of brute force attacks, integrating some previous results in [15].

2 Preliminaries

This section provides some basic results and terminology, which we need later. In § 2.1, we introduce the notion of distribution probability for a (finite domain) hash function $h : X \to Y$, showing that the multivariate random variable representing the numbers of occurrences of $x \in X$ such that $h(x) = y$ for any fixed $y \in Y$ has multinomial distribution. In § 2.2, we deal with some concepts and results from the theory of majorization, which will be useful to study the effects of varying the probability distribution of h on its resistance.

2.1 Hash functions and related distributions

Given an alphabet $\mathcal{A}$, an (unkeyed) hash function is informally defined in the computer science literature as a function which maps messages of arbitrary finite length to message of finite length from $\mathcal{A}$. Mathematically speaking, an hash function h is any function

$$h : \mathcal{A}^* \longrightarrow \mathcal{A}^n$$

where $\mathcal{A}^n$ denotes the n-times Cartesian product of $\mathcal{A}$ and $\mathcal{A}^* = \cup_{i=1}^{\infty} \mathcal{A}^i$.

Virtually any application based on the use of an hash function h requires - at a minimum - that, for any input message x, $h(x)$ depends of *all* the symbols in x [10]; therefore it is impossible in practice to evaluate an hash function of arbitrary length strings, and we can assume without a real restriction that $\mathcal{A}^* = \cup_{i=1}^{m} \mathcal{A}^i$ for a suitable $m \gg n$. Moreover, in the sequel we are interested in properties of hash functions depending mostly on the cardinality of their domain and range, rather than on the nature of these two sets. Thus, we adopt from [15] the following:

Definition 1 *Let M, N be positive integers such that $M > N$. An (M, N) hash function is any $h : X \longrightarrow Y$ surjective function, where X and Y are finite sets of cardinality $|X| = M$ and $|Y| = N$.*

In the sequel we will assume, without loss of generality, that

$$h(X) = Y = \{y_1, y_2, ..., y_N\}.$$

Let h be an (M, N) hash function, since:

$$\bigcup_{n=1}^{N} h^{-1}(y_n) = X, \qquad h^{-1}(y_i) \bigcap h^{-1}(y_j) = \emptyset \quad (1 \leq i \neq j \leq N)$$

it follows that

$$D : y \in Y \longrightarrow p_y = P(h^{-1}(y))$$

is a discrete probability distribution on Y for any probability space (X, Ω, P) defined on X such that $\{h^{-1}(y) : y \in Y\} \subseteq \Omega$.

In particular, we can choose $(X, \mathcal{P}(X), P^*)$ as probability space on X, where $\mathcal{P}(X)$ is the family of subsets of X and P^* is the probability function which assigns the same value to the elementary events of $\mathcal{P}(X)$. In other words, we consider the probability distribution induced by h on Y, given the uniform probability distribution on X:

Definition 2 *Let $h : X \to Y$ be an (M, N) hash function. The discrete distribution*

$$D_h : y_n \in \{y_1, y_2, ..., y_N\} \longrightarrow p_n = P^*(h^{-1}(y_n)) = |h^{-1}(y_n)|/M \qquad (1)$$

is the probability distribution of h.

Obviously, the simplest case is when h maps the uniform distribution on X to the uniform distribution on Y . For such case, it is natural to give the following definition [15]:

Definition 3 *An (M, N) hash function h is said to be uniform if D_h is the uniform distribution on Y.*

Remark 1 *If h is a uniform (M, N) hash function, then N divides M and each subset $h^{-1}(y)$ has cardinality equal to M/N.*

Given an (M, N) hash function $h : X \longrightarrow Y$, consider the trial $\mathcal{T}$ of evaluating $y = h(x)$ where x is randomly chosen[1] in X. Let $\mathcal{T}_1, ...\mathcal{T}_q$ $(q \in \mathbb{N})$ be a series of q trials $\mathcal{T}$ related to the same h. Obviously, $\mathcal{T}_1, ..\mathcal{T}_q$ is a series of independent trials, in each of which just one of the N mutually exclusive events $\mathcal{E}_n = \{x \in X : h(x) = y_n\}$ $(1 \leq n \leq N)$ can occur. Moreover, the probability of occurrence of event $\mathcal{E}_n$ is $P^*(\mathcal{E}_n) = P^*(h^{-1}(y_n)) = p_n$ for each trial $\mathcal{T}_i$.

If A_n $(1 \leq n \leq N)$ is the random variable representing the number of occurrences of the event $\mathcal{E}_n$ in $\mathcal{T}_1, ...\mathcal{T}_q$, then the joint probability distribution of the random vector $(A_1, ..., A_N)$ is the multinomial distribution of parameters $(q, p_1, ..., p_N)$ [6]:

$$P(A_1 = q_1, ..., A_N = q_N) = \frac{q!}{q_1!...q_N!}\, p_1^{q_1}...p_N^{q_N}, \qquad (2)$$

where $q_n \in \mathbb{N}_0$ for any $1 \leq n \leq N$ and $\sum_{n=1}^{N} q_n = q$. Moreover, any A_n has binomial distribution with parameters (q, p_n)[6]:

$$P(A_n = q_n) = \frac{q!}{q_n!(q - q_n)!}\, p_n^{q_n}(1 - p_n)^{q - q_n} \qquad (0 \leq q_n \leq q). \qquad (3)$$

[1] We use this expression to concisely indicate the selection of an element x from a set X whose elements have the same probability to be extracted.

2.2 Majorizations

The following notation and terminology was introduced in [5], but the notion of majorization has various origins and appeared in previous works; for a comprehensive treatment of theory of majorization see [7].

Definition 4 *Let* $\mathbf{x} = (x_1, x_2, ..., x_n)$ *and* $\mathbf{y} = (y_1, y_2, ..., y_n)$ *be two elements of* $\mathbb{R}^n$. *We say that* $\left(x_{[1]}, x_{[2]}, ..., x_{[n]}\right)$ *is a decreasing reordering of* $\mathbf{x}$ *if* $x_{[1]}x_{[2]}...x_{[n]}$ *constitutes a permutation of* $x_1x_2...x_n$ *such that* $x_{[i]} \geq x_{[i+1]}$ *for* $i = 1, .., n-1$. *Moreover, we say that* $\mathbf{x}$ *is majorized by* $\mathbf{y}$ *(* $\mathbf{y}$ *majorizes* $\mathbf{x}$ *) and use the notation* $\mathbf{x} \prec \mathbf{y}$ *if are satisfied the conditions*

$$\begin{aligned} \sum_{i=1}^{k} x_{[i]} &\leq \sum_{i=1}^{k} y_{[i]} \qquad (k = 1, .., n-1), \\ \sum_{i=1}^{n} x_{[i]} &= \sum_{i=1}^{n} y_{[i]} \ , \end{aligned}$$

where $\left(x_{[1]}, x_{[2]}, ..., x_{[n]}\right)$ *and* $\left(y_{[1]}, y_{[2]}, ..., y_{[n]}\right)$ *are decreasing rearrangements of* $\mathbf{x}$ *and* $\mathbf{y}$, *respectively.*

Majorization is a preordering relationship, because it satisfies the reflective and transitive properties. Moreover, it is a partial ordering on the set

$$D = \left\{\mathbf{x} \in \mathbb{R}^n : x_1 \geq x_2... \geq x_n\right\},$$

since $\mathbf{x} \prec \mathbf{y}$ and $\mathbf{y} \prec \mathbf{x}$ imply that exists an order n permutation matrix Π such that $\mathbf{x} = \Pi\mathbf{y}$. The following are some simple examples of majorizations which are useful to illustrate the previous definition.

Example 1 *Let* $\mathbb{R}^n_+$ *be the set of* n *dimensional arrays of non-negative real numbers, and let* $\mathbf{x} = (x_1, x_2, ..., x_n)$ *be any point in* $\mathbb{R}^n_+$ *with* $\sum_{i=1}^n x_i = c > 0$. *Then*

$$\left(\frac{c}{n}, \frac{c}{n}, ..., \frac{c}{n}\right) \prec (x_1, x_2, ..., x_n) \prec (c, 0, ..., 0) \ ;$$

This first example shows that, for any $c > 0$, the set $S_c = \left\{\mathbf{x} \in \mathbb{R}^n_+ : \sum_{i=1}^n x_i = c\right\}$ is bounded with respect to the ordering of majorization, with

$$\min_{S_c} \mathbf{x} = \left(\frac{c}{n}, \frac{c}{n}, ..., \frac{c}{n}\right), \qquad \max_{S_c} \mathbf{x} = (c, 0, ..., 0).$$

A simple way to "traverse" the set S_c with an increasing sequence of points is by means of the function

$$\mathbf{f} : t \in [1, n] \rightarrow \left(\frac{tc}{n}, \frac{n-t}{n^2-n}c, ..., \frac{n-t}{n^2-n}c\right). \tag{4}$$

Example 2 *Let* $\mathbf{x} = (x_1, x_2, ..., x_n)$ *be any point in* $\mathbb{R}^n$. *If* $x_i \geq x_j$, *then for any* $\epsilon \geq 0$

$$(x_1, x_2, ..., x_n) \prec (x_1, .., x_i + \epsilon, .., x_j - \epsilon, ..., x_n)$$

else if $x_i \leq x_j$, *then for any* $0 \leq \epsilon \leq x_j - x_i$

$$(x_1, x_2, ..., x_n) \succ (x_1, .., x_i + \epsilon, .., x_j - \epsilon, ..., x_n)$$

Example 3 *Let* $\mathbf{x} = (x_1, x_2, ..., x_n)$ *be any point in* $\mathbb{R}^n$ *and let* $\left(x_{[1]}, x_{[2]}, ..., x_{[n]}\right)$ *a decreasing reordering of* $\mathbf{x}$. *Then for any* $\epsilon \geq 0$

$$\begin{array}{l} \left(x_{[1]} + \epsilon, .., x_{[i]} + \epsilon, x_{[i+1]}, ..., x_{[n]} - i\epsilon\right) \prec \\ \left(x_{[1]} + \epsilon, .., x_{[j]} + \epsilon, x_{[j+1]}, ..., x_{[n]} - j\epsilon\right) \end{array}$$

only if $i \leq j$.

All the above examples indicate that the notion of majorization is a way to precise the idea that the components of a vector $\mathbf{x}$ are "less spread out" or "more nearly equal" than the components of a vector $\mathbf{y}$. Indeed, the aim of the theory of majorization is to allow the study of inequalities of the form

$$f(x_1, x_2, ..., x_n) \leq f(y_1, y_2, ..., y_n),$$

where $x_1, x_2, ..., x_n$ need not be all equal (as in many well-known elementary inequalities), but only less spread out than $y_1, y_2, ..., y_n$. For the ordering of majorization, a real-valued, order-preserving function is said a *Schur-convex* function [7].

Definition 5 *A real-valued function* f *defined on a set* $I \subseteq \mathbb{R}^n$ *is said to be Schur-convex on* I *if* $\mathbf{x}, \mathbf{y} \in I$ *and* $\mathbf{x} \prec \mathbf{y}$ *implies* $f(\mathbf{x}) \leq f(\mathbf{y})$. *If* $f(\mathbf{x}) < f(\mathbf{y})$ *whenever* $\mathbf{x} \prec \mathbf{y}$ *but* $\mathbf{x}$ *is not a permutation of* $\mathbf{y}$, *then* f *is strictly Schur-convex on* I .*Similarly,* f *is Schur-concave on* I *if* $\mathbf{x}, \mathbf{y} \in I$ *and* $\mathbf{x} \prec \mathbf{y}$ *implies* $f(\mathbf{x}) \geq f(\mathbf{y})$, *and strictly Schur-concave on* I *if* $f(\mathbf{x}) > f(\mathbf{y})$ *whenever* $\mathbf{x} \prec \mathbf{y}$ *but* $\mathbf{x}$ *is not a permutation of* $\mathbf{y}$.

Suppose that I is a symmetric set in $\mathbb{R}^n$, i.e. $\mathbf{x} \in I$ implies that $\Pi\mathbf{x} \in I$ for any order n permutation matrix Π. Since majorization is invariant under permutations, it follows that if f is Schur-convex or Schur-concave on I then f is a symmetric function on I, namely $f(\mathbf{x})$=$f(\Pi\mathbf{x})$ for all $\mathbf{x} \in I$ and all permutation matrix Π. Conversely, if f is symmetric on a set I and Schur-convex (concave) on $D \cap I$ where

$$D = \{\mathbf{x} \in \mathbb{R}^n : x_1 \geq x_2 ... \geq x_n\},$$

then f is Schur-convex (concave) on I. Thus, in studying the Schur-convexity (concavity) of a function f defined on a symmetric set I, we can restrict our analysis to the set $D \cap I$, provided that f is symmetric on I. On the other hand, for $\mathbf{x}, \mathbf{y} \in D$ the majorization $\mathbf{x} \prec \mathbf{y}$ is equivalent to the conditions:

$$\begin{cases} \widetilde{x}_k \leq \widetilde{y}_k & (k = 1, .., n-1) \\ \widetilde{x}_n = \widetilde{y}_n, \end{cases}$$

where $\widetilde{x}_k = \sum_{i=1}^{k} x_i$ and $\widetilde{y}_k = \sum_{i=1}^{k} y_i$. As a consequence, a function f is Schur-convex on D only if

$$f(x_1, x_2, ..., x_n) = f(\widetilde{x}_1, \widetilde{x}_2 - \widetilde{x}_1, ..., \widetilde{x}_n - \widetilde{x}_{n-1})$$

is increasing in $\widetilde{x}_k$ for $k = 1, ..., n-1$ over the region where $\mathbf{x} = (x_1, x_2, ..., x_n) \in D$. Moreover, because of the symmetry of f, it is sufficient to prove that, fixed $x_3, ..., x_n$, f is an increasing function of $\widetilde{x}_1$ for any $\widetilde{x}_2$ such that $\mathbf{x} = (x_1, x_2, ..., x_n) \in D$.

The previous considerations summarize into the following:

Criterion 1 *Let f be a real-valued function defined on a symmetric set I. Then f is (strictly) Schur-convex on I only if f is symmetric on I and for all $\mathbf{x} \in D \cap I$ the function*

$$\phi(\epsilon) = f(x_1 + \epsilon, x_2 - \epsilon, x_3, ..., x_n) \tag{5}$$

is (strictly) increasing in the set

$$E = \{\epsilon \geq 0 : (x_1 + \epsilon, x_2 - \epsilon, x_3, ..., x_n) \in D \cap I\}.$$

Conversely, f is (strictly) Schur-concave on I only if f is symmetric on I and for all $\mathbf{x} \in D \cap I$ the function $\phi(\epsilon)$ is (strictly) decreasing in the set E.

If f is continuous on D and differentiable in its interior, then the following alternative criterion holds [7]:

Criterion 2 *Let f be a real-valued function, defined and continuous on D and continuously differentiable on the interior of D. Then f is Schur-convex or Schur-concave on D depending that, respectively, the inequalities*

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) \geq \frac{\partial f}{\partial x_{i+1}}(\mathbf{x}) \qquad (i = 1, ..., n-1)$$

or

$$\frac{\partial f}{\partial x_i}(\mathbf{x}) \leq \frac{\partial f}{\partial x_{i+1}}(\mathbf{x}) \qquad (i = 1, ..., n-1)$$

are satisfied for all $\mathbf{x}$ in the interior of D.

We conclude this section by observing that, as suggested by Example 1, the notion of majorization is closely related to the *entropy function*

$$f(x) = -\sum_{i=1}^{n} x_i \log x_i \ ,$$

introduced by C. Shannon [13] as a measure of uncertainty in the information conveyed by a discrete random variable X with probability distribution given by $x_i = P(X = X_i)$, $(i = 1, .., n)$. That relation consists precisely in the fact that, as one could easily deduce by one of the previous criteria, the entropy function is strictly Schur-concave on the set S_1[7].

3 Cryptographic properties for hash functions

In the context of data security, hash functions are typically employed for digital signatures and message authentication. In [14] are described the following six basic uses of an hash function h, to guarantee a receiver B that a message x was actually sent by a sender A:

Scheme 1 *A sends to B* $y = C_k(x\|h(x))$*, where* C *is a symmetric cipher and* k *is a key shared only by A and B. B assumes that* $x_1 = x$ *only if* $C_k^{-1}(y) = x_1\|x_2$ *with* $x_2 = h(x_1)$*;*

Scheme 2 *A sends to B* $y = x\|C_k(h(x))$*, where* C *and* k *are as in the previous case. B assumes that* $y_1 = x$ *only if* $y = y_1\|y_2$ *with* $C_k^{-1}(y_2) = h(y_1)$*;*

Scheme 3 *A sends to B* $y = x\|C'_e(h(x))$*, where* C' *is an asymmetric cipher and* e *is an A's private key used as encryption key. B assumes that* $y_1 = x$ *only if* $y = y_1\|y_2$ *with* $C'_d(y_2) = h(y_1)$*, where* d *is the A's public key related to* e*;*

Scheme 4 *A sends to B* $y = C_k(x\|C'_e(h(x)))$*,where* C*,* k *are as in Case 1 and* C'*,* e *are as in the previous case. B assumes that* $x_1 = x$ *only if* $C_k^{-1}(y) = x_1\|x_2$ *with* $C'_d(x_2) = h(x_1)$*, where* d *is the A's public key related to* e*;*

Scheme 5 *A sends to B* $y = x\|h(x\|s)$ *, where* s *is a string shared only by A and B. B assumes that* $y_1 = x$ *only if* $y = y_1\|y_2$ *with* $y_2 = h(y_1\|s)$*;*

Scheme 6 *A sends to B* $y = C_k(x\|h(x\|s))$*, where* C*,* k *are as in Case 2 and* s *is a string shared only by A and B. B assumes that* $x_1 = x$ *only if* $C_k^{-1}(y) = x_1\|x_2$ *with* $x_2 = h(x_1\|s)$*;*

Schemes 1, 4 and 6 provides both message secrecy and authentication, and both of them follow from the use of conventional encryption rather than h. Indeed, we can assume that plaintext sent from A to B belongs to a small subset of X, i.e. the subset $\mathcal{L}$ of legitimate plaintexts from the alphabet $\mathcal{A}$. That is the case in the majority of applications of message authentication, namely in any case in which the transmitted messages have some internal structure, which depends on the nature of the application.

Under the above hypothesis, we could drop the use of the hash function h for Cases 1, 4 and 6, replacing them with the following simpler one:

Scheme 7 *Given* $x \in \mathcal{L} \subset X$*, A sends to B* $y = C_k(x)$*, where* C *is a symmetric cipher and* k *is a key shared only by A and B. B assumes that* $x_1 = C_k^{-1}(y)$ *is equal to* x *only if* $x_1 \in \mathcal{L}$*.*

Secrecy for Scheme 7 obviously follows from the cryptographic properties of C and the assumption about k.

As regard to authentication, let $|X| = M$ and $|\mathcal{L}| = L$. Then the probability that a randomly chosen $x' \in X$ is a legitimate plaintext and $x' \neq x$ is $(1 - 1/L)L/M$. Thus B can assume with probability

$$p = 1 - \left(1 - \frac{1}{L}\right)\frac{L}{M}$$

that $x_1 = x$ if $x_1 \in \mathcal{L}$. Since typically $L \ll M$, then B is assured with high probability p that the received message was the one sent by A, i.e. of the authenticity of A's message.

On the basis of the previous reasoning, one could wonder that no practical cryptographic protocol exists which employes Scheme 7 to provide message authentication. The reason is that it may be difficult (and/or computationally costly) to define a procedure that determines if x_1 is a legitimate plaintext or not. Moreover, that procedure must clearly depend on the structure of legitimate plaintexts, which in turn - as previously pointed out - is strictly influenced by the considered application.

On the contrary, the use of Schemes 1, 4, 6 leads to efficient procedures to establish if the received message must be accepted or not, provided that the hash function h employed in such schemes can be efficiently evaluated.

The remaining three schemes, namely Schemes 2, 3 and 5, provide message authentication but not confidentiality. When confidentiality is not a requirement, these schemes have an advantage over those that encrypt the entire message, in that less computation is required. Moreover, unlike message encryption based schemes, these methods aren't subject to export control laws in force in some countries. As pointed out in [14], Case 3 is the essence of digital signature technique, which is at present the most important application of hash functions in data security.

However, the effectiveness of such schemes don't follow only by the easy computability of h, but relies on some cryptographic properties of it.

Schemes 2 and 3 both rely on the difficulty to find an $x' \in \mathcal{L} - \{x\}$ such that $h(x') = h(x)$ for any given $x \in \mathcal{L}$. Indeed, for any such x' B assumes erroneously that $y_1 = x'$, so that the lack of the previous requirement expose both procedures to attacks based on message forgery. This cryptographic requirement for an hash function is often referred to in literature as *second preimage resistance*.

A related, stronger property is *collision resistance*, that can be informally stated as the difficulty to find any two $x, x' \in \mathcal{L}$ such that $x \neq x'$ and $h(x) = h(x')$. Collision resistance is often invoked for Schemes 2 and 3 to protect them against message forgery attacks based on the birthday paradox [14, 8].

Finally, Scheme 5 relies on the difficulty to find, for $y \in h(X)$, a preimage of y under h, which is usually referred to as *preimage resistance*. If indeed preimage resistance wasn't satisfied for the hash function h in such scheme, then an adversary could derive $x\|s$ from its digest and easily compute the shared secret s from x. Preimage resistance is also required for Case 3, i.e. for digital signatures, otherwise an attacker E could achieve a certain level of disruption simply by issuing messages with random content purporting to come from a

legitimate user A. That can be obtained precisely as follows: (a) E chooses a random value z and uses A's public key d to compute $C'_d(z) = y$; (b) since preimage resistance for h doesn't hold, E can compute an $x \in h^{-1}(y)$; (c) at this point, E could claim that the message x was signed by A. Since C'_d and h can be efficiently evaluated, steps (a) and (b) can be eventually repeated many times to find a legitimate message x of practical use for E, leading in such a case to an effective message forgery attack.

The sense of the previous cryptographic properties for an hash function has been sometimes misused in literature, leading to ambiguous theoretical conclusions and deceptive requirements for the practical construction of such functions. The above notions of resistance are indeed only informal and cannot be considered real definitions: as a matter of fact, what one should intend for "*difficult to find*" in the definitions of preimage, second preimage and collision resistance given before?

Many authors (e.g. [8, 14]) employ the expression "*computationally infeasible*" rather than "*difficult*", but that results in no more than conveying the idea that the intended obstacle is algorithmic in nature and impossible to overcome in practice. Others [12] prefer the term "*hard*", perhaps with an implicit reference to complexity theory terminology.

In any way, a rigorous approach clearly requires a precise definition of the model of computation, used by an hypothetical adversary, *to find* elements in X (or, more properly, in $\mathcal{L}$) which satisfy given requirements with respect to their images under an hash function h. Moreover, terms as "*easy*" and "*computationally infeasible*" (and all their clones) should be interpreted relative to precise resource bounds for the adversaries, and expressed in terms of cost units in the above model of computation.

In other words, rigorous notions of *resistance* for an hash function h require a mathematical model which enables to specify both the kind of attacks that could be mounted against h and the amount of resources available to the adversaries for such attacks.

For a model which aims to encompass generic hash functions, that is a difficult task: many efforts have been done on the subject, resulting in different, not equivalent formal definitions of preimage, second preimage and collision resistance. Reviewing them is out of the scope of this article; we simply remark that both approaches based on probabilistic Turing machines, random access machines (RAM) and the random oracle model haven't led to concrete methods to evaluate how much secure an existing hash function is.

In contrast, our aim is to deploy a framework which leads to quantitative measurements of how much *resistance* a real hash function has with respect to a class of attacks of practical relevance.

4 Brute force attacks on hash functions

Brute force attacks are by far the simplest way to seek for breaking a cryptographic primitive. They are typically used to find a cipher secret key, by means of an exhaustive search in the cipher key space, following a "trial-and-error" approach. These methods don't take advantage in any way on *how* the cryptographic primitive is computed, but only on the speed at which this computation occurs. The analysis of their effectiveness (and performance) can thus be accomplished simply by assuming that an adversary, for any suitable input, can compute the output of the primitive; more important, the above analysis only depends on combinatorial properties of the cryptographic primitive involved.

In our context, given an hash function h, brute force attacks simply consist in repeated evaluation of the image $h(x)$ for $x \in \mathcal{L}$ until some condition on $h(x)$, depending of the nature of the attack, is satisfied. Since our analysis is purely combinatorial, we can assume in the sequel, without loss of generality, that $X = \mathcal{L}$. Moreover, instead of considering exhaustive, deterministic brute force algorithms, in which the entire domain X of h is progressively scanned element by element until the condition on $h(x)$ occurs, we will consider randomized ones. More precisely, we will consider algorithms which select x at random in the set X for a maximum number q of times, where q is an input parameter. It turns out that those are Las Vegas algorithms, since they always give a correct answer or do not respond.

Definition 6 *Let $h : X \to Y$ be an (M, N) hash function. Let $\rho : \mathbb{N} \to X$ be a pseudo-random generator with $\rho(\{1, 2, .., M\}) = X$ such that its output is uniformly distributed on X. A Las Vegas brute force (LVBF) algorithm for h is an algorithm which executes at most q steps and returns a correct answer or* `NULL` *with probabilities $p = p(h, q)$ and $1 - p$, respectively. At each step, the algorithm generates elements of the search space X using the pseudo-random generator ρ.*

We stress that, in the previous definition, the pseudo-random generator is required only to have good statistical properties, not to provide a cryptographically secure (i.e. unpredictable) stream of random elements. A such generator can be obtained by applying a suitable bijective transformation from the output of a traditional statistical pseudo-random number generator (PRNG) to the set X, provided that the period length of the PRNG is greater or equal than M. Since a statistical PRNG outputs all the numbers between 0 and its period exactly once and then repeats the sequence , we are guaranteed that LVBF algorithms exist. The following are three of such algorithms to solve preimage, second preimage and collision finding problems, respectively. They implement the generator ρ via the procedure `rho`. As any PRNG based on a deterministic algorithm, `rho` requires an initial value - called *seed* - which is used as a starting point to generate the sequence $\{x_i = \rho(i)\}_i$; this sequence is uniquely determined by the seed value, and different seed values result in different sequences. The choice of the seed value should be done in a truly random way for such algorithms to be effective; we don't detail here how to choose the seed, but simply assume that its randomness in achieved by means of the use of the

"black-box" `random`, which could also represent the output of a suitable physical device.

Algorithm 1 *`FindPreimage` requires in input an hash function $h : X \longrightarrow Y$, a digest $y \in Y$ and an integer q indicating the maximum number of trials to run. It returns x if a preimage x of y was found after at most q steps and `NULL` otherwise.*

```
FindPreimage(h,y,q)
begin
  found=false
  counter=1
  seed=random()
  repeat
    x=rho(counter,seed)
    yc=h(x)
    counter=counter + 1
    if (y=yc)
       found=true
    endif
  until(counter>q OR found)
  if(found)
     return x
  else
     return NULL
  endif
end
```

In the next algorithm q equals as before the maximum number of possible evaluations of h, but the number of trials of the algorithm is now $q-1$, because one function evaluation is spent to compute the image y. This choice is because we measure the performance of these algorithms in terms of the number of evaluations of h.

Algorithm 2 *`Find2ndPreimage` requires as input an hash function $h : X \longrightarrow Y$, a message $x \in X$ and an integer q indicating the maximum number of evaluations of h. It outputs x' in case that a preimage $x' \in h^{-1}(h(x))$ such that $x' \neq x$ was found after at most q steps, `NULL` otherwise.*

```
Find2ndPreimage(h,x,q)
begin
  found=false
  counter=1
  y=h(x)
  seed=random()
  while(counter<q AND !found)
    xc=rho(counter,seed)
```

```
    yc=h(xc)
    counter=counter+1
    if (x!=xc AND y=yc)
       found=true
    endif
  end
  if(found)
     return xc
  else
     return NULL
  endif
end
```

The third and last algorithm is to solve the collision searching problem. It uses the function `ismember(a[1:k],b)` which returns the index `i` of an `a[i]`such that `a[i]=b` if `b` belongs to the vector `a[1:k]`, and zero otherwise. We stress that this algorithm realizes a randomized birthday attack. Birthday attacks derive their name from the birthday paradox [8, 14], and their relevance to hash functions is because it has been conjectured that, in case of *algorithm-independent* attacks, to find a collision is easier than to find a preimage[2]. As we shall see in §6, that conjecture is indeed true in the more general case of LVBF attacks.

Algorithm 3 *`FindCollision` requires as input an hash function $h : X \longrightarrow Y$ and an integer q indicating the maximum number of trials to run. It returns the couple (x, x') if two messages $x' \neq x$ were found after no more than q steps such that $h(x') = h(x)$, `NULL` otherwise.*

```
FindCollision(h,q)
begin
   found=false
   counter=1
   seed=random()
   repeat
      x[counter]=rho(counter,seed)
      y[counter]=h(x[counter])
      if ((i=ismember(y[1:counter-1],y[counter]))>0)
         found=true
      endif
      counter=counter+1
   until(counter>q OR found)
   if(found)
     return (x[i],x[counter])
```

[2] Algorithm-independent attacks are those which treat the hash function being attacked as a black-box, whose only significant characteristics are its output bitlength and the running time for one hash operation [8].

```
    else
      return NULL
    endif
end
```

The above algorithms are very similar to the homonymous ones presented in [15] and, as in that article, we will consider the probability p_{ALG} that the algorithm ALG solves the related problem (i.e. doesn't return `NULL`) at most after q evaluations of the hash function h[3]. However, in [15] these algorithms were analyzed in the random oracle model. That model consists of the following additional assumptions regarding the hash function h:

- $h : X \to Y$ is a randomly chosen function in the set of all functions from X to Y;
- an algorithm is given only *oracle* access to h, i.e. h appears as a black-box and no advantage on how h actually is the algorithm can take at all.

As a consequence of these two assumptions, the following independence property holds [15]:

Proposition 3 *Suppose that $h : X \to Y$ satisfies the above assumptions and that the values $y_i = h(x_i)$ have been determined for $1 \leq i \leq q$. Then*

$$P(h(x) = y / y_1 = h(x_1), ..., y_q = h(x_q)) = \frac{1}{N}$$

for any $x \in X - \{x_1, ..., x_q\}$ and any $y \in Y$.

From the previous result it follows that the random oracle model doesn't catch the combinatorial properties of the hash function h being used, in that all functions appear in the model to be as *uniform* hash functions[4]. Thus, the random oracle model is ineffective to establish how much secure a given hash function is with respect to Las Vegas brute force attacks. To solve the question, we need to describe h in terms of its probability distribution, and to analyze our algorithms in the framework of the calculus of probability. This is accomplished by the following theorems, which establish what is the success probability of the above three LVBF algorithms for an input (M, N) hash function of arbitrary probability distribution.

Theorem 4 *Let be $h : X \to Y$ an (M,N) hash function with probability distribution (1). The success probability of algorithm `FindPreimage` , averaged over all $y \in Y$ is given by:*

$$p_{FP} = 1 - \frac{1}{N} \sum_{n=1}^{N} (1 - p_n)^q \tag{6}$$

[3] Analogously to [15], we will denote as (p, q)-LVBF an LVBF algorithm which requires at most q evaluations of h and has a success probability equal to p.

[4] Alternatively, we could say that the random oracle model encompasses only algorithmic-independent attacks.

Proof. Since $y \in Y$, then $y = y_n$ for some $1 \leq n \leq N$. Let $\{x_i = \rho(i), 1 \leq i \leq q\}$ be a q-length sequence obtained by the pseudo-random generator ρ, and let be the trial of evaluating $h(x_i)$. It follows that$\{\mathcal{T}_i, 1 \leq i \leq q\}$ is a sequence of independent trials in each of which the events:

$$\mathcal{E}_n^{(i)} = \{h(x_i) = y_n\},\ 1 \leq i \leq q$$

can occur with probability $p_n = P^*(h^{-1}(y_n))$ for any $1 \leq i \leq q$. From (3) it follows that the probability that at least one of the events $\mathcal{E}_n^{(i)}$ occurs in the q trials $\mathcal{T}_i$ (i.e. the probability of occurrence of $\cup_{i=1}^{q}\mathcal{E}_n^{(i)}$) is given by:

$$P(\bigcup_{i=1}^{q}\mathcal{E}_n^{(i)}) = P(A_n \geq 1) = 1 - P(A_n = 0) = 1 - (1-p_n)^q, \tag{7}$$

where A_n is the random variable indicating the number of occurrences of the event $\mathcal{E}_n = \{h(x) = y_n\}$ in the q trials. Therefore:

$$p_{FP} = \sum_{n=1}^{N}\frac{1}{N}(1-(1-p_n)^q) = 1 - \frac{1}{N}\sum_{n=1}^{N}(1-p_n)^q$$

■

Theorem 5 *Let $h : X \to Y$ be an (M, N) hash function with probability distribution (1).The success probability of algorithm* `Find2ndPreimage`*, averaged over all $x \in X$, is given by:*

$$p_{F2P} = 1 - \sum_{n=1}^{N} p_n \left(1 - p_n + \frac{1}{M}\right)^{q-1} \tag{8}$$

Proof. Say $h(x) = y_n$; we have to consider the events:

$$\mathcal{E}_n^{(i)} = \{h(x_i) = y_n\ \&\ x_i \neq x\},\ 1 \leq i \leq q-1$$

and from the conditional probability rule it follows that:

$$P(\mathcal{E}_n^{(i)}) = P^*(x_i \neq x / h(x_i) = y_n)P^*(h(x_i) = y_n)$$

Since $P^*(h(x_i) = y_n) = p_n$ and

$$P^*(\ x_i \neq x\ /\ h(x_i) = y_n) = \frac{|h^{-1}(y_n)| - 1}{|h^{-1}(y_n)|} = 1 - \frac{1}{Mp_n} \tag{9}$$

from the previous relationship it follows that, for any $1 \leq i \leq q-1$:

$$P(\mathcal{E}_n^{(i)}) = p_n - \frac{1}{M}$$

Now we can apply the same reasoning than in Theorem 1, obtaining that the probability that at least one of the above events $\mathcal{E}_n^{(i)}$ occurs in the q trials $\mathcal{T}_i$ is given by:

$$P(\bigcup_{i=1}^{q-1} \mathcal{E}_n^{(i)}) = 1 - \left(1 - p_n + \frac{1}{M}\right)^{q-1}$$

Finally, averaging over all $x \in X$:

$$p_{F2P} = \sum_{n=1}^{N} p_n \left(1 - \left(1 - p_n + \frac{1}{M}\right)^{q-1}\right) = 1 - \sum_{n=1}^{N} p_n \left(1 - p_n + \frac{1}{M}\right)^{q-1}$$

■

The following theorem gives the exact expression of the success-rate p_{FC} for the randomized birthday attack, rather than estimating it, as in [3].

Theorem 6 *Let $h : X \rightarrow Y$ be an (M, N) hash function with probability distribution (1).The success probability of algorithm* `FindCollision` *is given by:*

$$p_{FC} = \begin{cases} 1 & \text{for } q > N \\ 1 - \sum\limits_{C_q^{(1,\ldots,N)}} q!\, p_{j_1}..p_{j_q} & \text{for } q \leq N \end{cases} \tag{10}$$

where $C_q^{(1,\ldots,N)}$ denotes the set of all combinations of size q from $\{1, 2, .., N\}$.

Proof. The result for $q > N$ is trivial, thus we can assume $q \leq N$. Let $\mathcal{E}_n^{(i)}$ and A_n have the same meaning than in Theorem 1. From (2) it follows that the probability that in q trials no collisions occur is:

$$P(A_1 \leq 1, ..., A_N \leq 1) = \sum_{q_1..q_N \in S} \frac{q!}{q_1!...q_N!}\, p_1^{q_1}...p_N^{q_N},$$

where S is the set of all N dimensional arrays of exactly q ones and $N-q$ zeros. Thus, if $C_q^{(1,\ldots,N)}$ denotes the set of all combinations $j_1\ j_2...\ j_q$ of size q from $\{1, 2, .., N\}$, it follows that:

$$P(A_1 \leq 1, ..., A_N \leq 1) = \sum_{C_q^{(1,\ldots,N)}} q!\, p_{j_1}..p_{j_q}$$

and

$$p_{FC} = 1 - P(A_1 \leq 1, ..., A_N \leq 1) = 1 - \sum_{C_q^{(1,\ldots,N)}} q!\, p_{j_1}..p_{j_q}$$

■

In the special case that h is a uniform (M, N) hash function, we obtain the following:

Corollary 7 *Let $h : X \to Y$ be an (M,N) uniform hash function. Then the success probability of algorithms* `FindPreimage`*,* `Find2ndPreimage` *and* `FindCollision` *is given, respectively, by:*

$$p_{FP} = 1 - \left(1 - \frac{1}{N}\right)^q , \tag{11}$$

$$p_{F2P} = 1 - \left(1 - \frac{1}{N} + \frac{1}{M}\right)^{q-1} \tag{12}$$

and

$$p_{FC} = \begin{cases} 1 & for\ q > N \\ 1 - \frac{N!}{(N-q)!N^q} & for\ q \leq N \end{cases} \tag{13}$$

Proof. The first two equalities easily follows by putting $p_n = 1/N$ in (6) and (8), respectively. To obtain the third one, it suffices to observe than in (10) the set $C_q^{(1,\ldots,N)}$ has cardinality

$$\binom{N}{q} = \frac{N!}{q!(N-q)!}$$

■

We remark that (11), (13) and (12) (with the term $1/M$ neglected, which is a good approximation if, as supposed, $M \gg N$) are the success probabilities derived in [15]. This perfectly agrees with our previous observations about the random oracle model.

5 Effects of the probability distribution

Scope of this section is the analysis of the effects of varying the hash function on the success probabilities of LVBF algorithms introduced in the previous section. As pointed out by many authors, that is a crucial step in the deployment of a soundness approach to hash function design and parameter choices. Since in the framework of LVBF algorithms an hash function is completely characterized by its probability distribution, it suffices to study how, given $N > 0$, those probabilities varies in the set

$$S = \left\{ \mathbf{p} \in \mathbb{R}_+^N : \sum\nolimits_{n=1}^{N} p_n = 1 \right\} . \tag{14}$$

Theorem 8 *Let $h : X \to Y$ be an (M, N) hash function. The success probability p_{FP} of algorithm* `FindPreimage`*(h, y, q) is Schur-concave on S if $q = 1$ and strictly Schur-concave on S if $q > 1$.*

Proof. From (6) it follows that p_{FP} is a symmetric function on the set S and that the function ϕ defined by (5) is given by

$$\phi(\epsilon) = 1 - \frac{1}{N}\left[(1 - p_1 - \epsilon)^q + (1 - p_2 + \epsilon)^q + \sum_{n=3}^{N}(1 - p_n)^q\right] .$$

Moreover for any $\mathbf{p} \in S \cap D$, where $D = \left\{\mathbf{p} \in \mathbb{R}_+^N : p_1 \geq p_2 ... \geq p_N\right\}$, the following equality holds

$$E = \{\epsilon \geq 0 : (p_1 + \epsilon, p_2 - \epsilon, p_3, ..., p_N) \in S \cap D\} = [0, p_2 - p_3]$$

and ϕ is differentiable in $[0,1] \supseteq E$ with

$$\frac{d\phi}{d\epsilon}(\epsilon) = -\frac{q}{N}\left[(1 - p_2 + \epsilon)^{q-1} - (1 - p_1 - \epsilon)^{q-1}\right] .$$

The above result implies that, for any $\mathbf{p} \in S \cap D$, ϕ is constant in E if $q = 1$ and strictly decreasing in E if $q > 1$. Thus the assertion follows from Criterion 1. ■

The behavior of the success probability p_{F2P} for `Find2ndPreimage` algorithm is not so sharp as in the previous case (and, as we will show later, as in the case of the algorithm `FindCollision`). Except than for $q = 2$, it turns out that p_{F2P} is neither convex nor concave on the set S, and that we have to consider the restriction of p_{F2P} to suitable symmetric subsets of S to obtain Schur properties.

Theorem 9 *Let $h : X \to Y$ be an (M, N) hash function. Moreover, given an integer $q \geq 2$, let S'_q and S''_q denote the following symmetric subsets of S:*

$$\begin{aligned} S'_q &= \left\{\mathbf{p} \in I : p_{[3]} \geq \frac{2}{q}\left(1 + \frac{1}{M}\right)\right\}, \\ S''_q &= \left\{\mathbf{p} \in I : p_{[1]} + p_{[2]} - p_{[3]} \leq \frac{2}{q}\left(1 + \frac{1}{M}\right)\right\}, \end{aligned}$$

where $(p_{[1]}, p_{[2]}, ..., p_{[n]})$ is a decreasing reordering of $\mathbf{p} = (p_1, p_2, ..., p_n)$. For any q, the success probability p_{F2P} of algorithm `Find2ndPreimage`(h, x, q) is strictly Schur-concave on S'_q and strictly Schur-convex on S''_q .

Proof. The expression (8) for p_{F2P} shows that p_{F2P} is a symmetric function on the set S; moreover, from (5) it follows that

$$\begin{aligned} \phi(\epsilon) &= 1 - (p_1 + \epsilon)(1 - p_1 - \epsilon + \frac{1}{M})^{q-1} - (p_2 - \epsilon)(1 - p_2 + \epsilon + \frac{1}{M})^{q-1} + \\ &\quad - \sum_{n=3}^{N} p_n (1 - p_n + \frac{1}{M})^{q-1}. \end{aligned}$$

Recall that for algorithm `Find2ndPreimage`(h, x, q) must be $q \geq 2$. As before, for any $\mathbf{p} \in S \cap D$, $E = [0, p_2 - p_3]$ and ϕ is differentiable in E. On the other hand, simple algebra leads in this case to

$$\frac{d\phi}{d\epsilon}(\epsilon) = q[(c_2 + \epsilon)^{q-1} - (c_1 - \epsilon)^{q-1}] - (q-1)\left(1 + \frac{1}{M}\right)[(c_2 + \epsilon)^{q-2} - (c_1 - \epsilon)^{q-2}],$$

where $c_1 = 1 + \frac{1}{M} - p_1$ and $c_2 = 1 + \frac{1}{M} - p_2$. It is easy to verify that, if $\mathbf{p} \in S \cap D$ and $0 < \epsilon \leq p_2 - p_3$, then

$$\frac{1}{M} < c_1 - \epsilon < c_2 + \epsilon \leq 1 + \frac{1}{M}$$

If $q = 2$ the previous expression for the derivative of ϕ becomes

$$\frac{d\phi}{d\epsilon}(\epsilon) = 2(c_2 - c_1 + 2\epsilon),$$

thus the statement follows from Criterion 1, since in this case $S'_q = \emptyset$ and $S''_q = I$. If instead $q > 2$, let consider the function:

$$\varphi(x) = qx^{q-1} - (q-1)\left(1 + \frac{1}{M}\right)x^{q-2}$$

It results that

$$\frac{d\phi}{d\epsilon}(\epsilon) = \varphi(c_2 + \epsilon) - \varphi(c_1 - \epsilon)$$

and

$$\frac{d\varphi}{dx}(x) = (q-1)x^{q-3}\left[qx - (q-2)\left(1 + \frac{1}{M}\right)\right].$$

Thus the derivative of ϕ is a strictly negative function of ϵ if

$$\sup_{\epsilon}(c_2 + \epsilon) = 1 + \frac{1}{M} - p_3 \leq \left(1 - \frac{2}{q}\right)\left(1 + \frac{1}{M}\right)$$

and, conversely, it is a strictly positive function of ϵ for

$$\inf_{\epsilon}(c_1 - \epsilon) = 1 + \frac{1}{M} - p_1 - p_2 + p_3 \geq \left(1 - \frac{2}{q}\right)\left(1 + \frac{1}{M}\right).$$

The above inequalities are equivalent, respectively, to

$$p_3 \geq \frac{2}{q}\left(1 + \frac{1}{M}\right)$$

and

$$p_1 + p_2 - p_3 \leq \frac{2}{q}\left(1 + \frac{1}{M}\right);$$

that, again on the basis of Criterion 1, proves the statement for $q > 2$. ■

Remark 2 *Clearly $S'_q \cap S''_q = \emptyset$ for any integer q; moreover, it is easy to show that $S'_q = \emptyset$ for $q \leq 6$ and that $S''_q = \emptyset$ for $q > 2N$. If $q = 2$ then, as previously noted, $S''_q = I$.*

Theorem 10 *Let $h : X \to Y$ be an (M,N) hash function. The success probability p_{FC} of algorithm* `FindCollision`(h, q) *is strictly Schur-convex or Schur-convex on S depending if $q \leq N$ or $q > N$, respectively.*

Proof. The second part of the statement trivially follows from (10). If $q \leq N$, from (10) it follows that p_{FC} is symmetric on S and that

$$\begin{aligned}\phi(\epsilon) &= 1-(p_1+\epsilon)\sum_{C_{q-1}^{(3,\ldots,N)}} q!\,p_{j_1}..p_{j_{q-1}} - (p_2-\epsilon)\sum_{C_{q-1}^{(3,\ldots,N)}} q!\,p_{j_1}..p_{j_{q-1}} \\ &\quad -(p_1+\epsilon)(p_2-\epsilon)\sum_{C_{q-2}^{(3,\ldots,N)}} q!\,p_{j_1}..p_{j_{q-2}} - \sum_{C_q^{(3,\ldots,N)}} q!\,p_{j_1}..p_{j_q},\end{aligned}$$

where $C_q^{(3,\ldots,N)}$, $C_{q-1}^{(3,\ldots,N)}$ and $C_{q-2}^{(3,\ldots,N)}$ are the sets of all combinations $j_1\ j_2...\ j_n$ from $\{3,...,N\}$ of sizes q, $q-1$ and $q-2$, respectively. Since

$$\frac{d\phi}{d\epsilon}(\epsilon) = (p_1-p_2)\sum_{C_{q-2}^{(3,\ldots,N)}} q!\,p_{j_1}..p_{j_{q-2}} + 2\epsilon\sum_{C_{q-2}^{(3,\ldots,N)}} q!\,p_{j_1}..p_{j_{q-2}}$$

and since $\mathbf{p} \in S \cap D$ implies $p_1 - p_2 \geq 0$ and $p_n \geq 0$ $(n = 1,...,N)$, the proof easily follows from Criterion 1. ■

Since, as shown in the Example 1, the set S with respect to the ordering of majorization is a bounded set with:

$$\min_S(p_1,p_2,...,p_N) = \left(\frac{1}{N},\frac{1}{N},...,\frac{1}{N}\right), \qquad \max_S(p_1,p_2,...,p_N) = (1,0,...,0)\,,$$

a first important consequence of the previous results is that the minimum chance to find a collision with a LVBF algorithm is in the case that the hash function being attacked is uniform, whereas for that type of hash function the chance to find a preimage is at its maximum. At a first glance, that seems to indicate the impossibility to find the best hash outcomes distribution with respect to all the three types of attacks. As we shall see in the next section this is not the case, since collision resistance has to be considered the most stringent security requirement for an hash function exposed to a brute force attack.

6 Relationships among properties

Relationships among preimage, second preimage and collision searching problems have been considered by many authors. It is obvious that, both for hash function design and usage, it is very useful to establish if, at least for some suitable models of computation, one of the above problems is more difficult to solve than another.

For conciseness, let us denote respectively with $\mathcal{P}(\Gamma)$, $\mathcal{P}'(\Gamma)$ and $\mathcal{C}(\Gamma)$ the sets of hash functions which satisfy the properties of preimage, second preimage and collision resistance with respect to a given model of computation Γ. The relation among $\mathcal{P}'(\Gamma)$ and $\mathcal{C}(\Gamma)$ is a trivial question. It is obvious that any solution of the second preimage problem is a solution to the collision problem, too; thus

$$\mathcal{C}(\Gamma) \subseteq \mathcal{P}'(\Gamma) \tag{15}$$

independently of Γ. In our approach (15) becomes the following inequality, satisfied for all $q, M, N \in \mathbb{N}$ and any $(p_1, p_2, ..., p_N) \in S$, with S given by (14):

$$p_{F2P}(q, M, N, p_1, p_2, ..., p_N) \leq p_{FC}(q, M, N, p_1, p_2, ..., p_N)$$

which, because of (8) and (10), results in:

$$1 - \sum_{n=1}^{N} p_n \left(1 - p_n + \frac{1}{M}\right)^{q-1} \leq 1 - \sum_{C_q^{(1,...,N)}} q!\, p_{j_1} .. p_{j_q}$$

By considering the limit for $M \to +\infty$ of the first member, it follows that:

$$1 - \sum_{n=1}^{N} p_n \left(1 - p_n\right)^{q-1} \leq 1 - \sum_{C_q^{(1,...,N)}} q!\, p_{j_1} .. p_{j_q}, \quad q \in \mathbb{N} \tag{16}$$

Relation (16) furnishes a very rough lower bound for the more complex expression of success probability for algorithm `FindCollision`; much better (lower and upper) bounds on p_{FC} were derived in [3], but only for q values not greater than a quantity depending both on N and the balance of h.

A more intriguing problem are the relationships among preimage and collision searching problems. Indeed, on the contrary than in the previous case, the answer to this question depends both on the adopted model of computation and on properties of the particular hash function considered.

Under the generic assumption than an adversary knows how the hash function is computed, which is the essence of Kerkhoff's principle, universally adopted in theory and practice of cryptography, examples can be produced of both collision but not preimage resistant hash functions and preimage but not collision resistant ones.

It presents no difficulty to understand why $\mathcal{P}(\Gamma) \nsubseteq \mathcal{P}'(\Gamma)$ which, in view of (15), implies $\mathcal{P}(\Gamma) \nsubseteq \mathcal{C}(\Gamma)$. Functions $h : X \to Y$ such that

$$h \in \mathcal{P}(\Gamma) \text{ and } h \notin \mathcal{P}'(\Gamma)$$

are indeed characterized by the property that it is easy in Γ to derive $x' \in h^{-1}(y) - \{x\}$ from a previously known preimage x of $y \in Y$. A classical example is the modular squaring function:

$$h(x) = x^2 \bmod n,$$

where $n = pq$ and p,q are appropriate, randomly chosen secret primes [8].

At this point, the last matter to be discussed in order to completely solve our problem is that $\mathcal{C}(\Gamma) \nsubseteq \mathcal{P}(\Gamma)$. For this purpose, it turns useful the following example. Given $X = \bigcup_{i=1}^{m} \{0,1\}^i$, $Y = \{0,1\}^l$ $(m > l)$ and an hash function $g : X \to Y$, let us consider the function:

$$h : x \in X \to \begin{cases} x & \text{if } |x| = l \\ g(x) & \text{otherwise} \end{cases}$$

Since $x = y = h^{-1}(y)$ for all $y \in Y$, it follows that $h \notin \mathcal{P}(\Gamma)$. On the other hand, if $h(x) = h(x')$ and $x \neq x'$ then only two alternatives are possible: (a) both $|x|$ and $|x'|$ are different from l; (b) just one of $|x|$ and $|x'|$ is equal to l, say $|x'|$. Finding a collision for h is equivalent to find a collision for g in case (a) and a preimage of g (different from $|x'|$) in case (b), thus $h \in \mathcal{C}(\Gamma)$ if $g \in \mathcal{C}(\Gamma) \bigcap \mathcal{P}(\Gamma)$.

We reiterate that, with the exception of relation (15), the above conclusions are only valid for computational models Γ which retain the Kerkhoff's principle. Clearly, this is not the case for any purely combinatorial model, as the random oracle model and our model based on the brute force approach. In such models, mutual relationships among preimage, second preimage and collision resistance properties for hash functions can be adequately described using the results of §4. Indeed, it is straightforward to realize that algorithms `FindPreimage`, `Find2ndPreimage` and `FindCollision` are optimal, in the sense that no other LVBF algorithms exist which solve the same problems with larger success probabilities. Thus, it is natural to regard the success probabilities derived in §4 as measures of resistance for a generic hash function with respect to brute force attacks. In this respect, inequality (16) has to be considered *the proof* that $\mathcal{C}(\Gamma) \subseteq \mathcal{P}'(\Gamma)$ in our model of computation. In general, for that model, we must conclude that $\mathcal{X}(\Gamma) \subseteq \mathcal{Y}(\Gamma)$ if $p_Y \leq p_X$ for any hash function and any number of trials q, where $\mathcal{X}, \mathcal{Y} \in \{\mathcal{C}, \mathcal{P}, \mathcal{P}'\}$ and $p_Y, p_X \in \{p_{FP}, p_{F2P}, p_{FC}\}$. Since p_{FP}, p_{F2P} and p_{FC} are increasing functions of q, the previous requirement on the number of trials can be relaxed to be true only for $q \geq q_0$, where q_0 is a number of trials such that the attack can be easily mounted in practice. Following this idea, we can prove that, in the context of brute force attacks, it is easier to find collision for an hash function than to find preimages. The above has been so far only conjectured by many authors, usually assuming that the hash output approximates a uniform random variable; the next theorem demonstrates that conjecture in the general case, i.e. for hash functions with arbitrary probability distribution.

Theorem 11 *Let h be an (M, N) hash function with probability distribution (1), and let q, p_{FC} , p_{FP} denote the number of trials and the corresponding success probabilities for algorithms `FindCollision` and `FindPreimage`, respectively. Then, for any $q > 2$ and for any choice of the probability distribution $(p_1, p_2, ..., p_N)$:*

$$p_{FC}(q, M, N, p_1, p_2, ..., p_N) \geq p_{FP}(q, M, N, p_1, p_2, ..., p_N) \ ,$$

where the equality is attained only for $N = 1$.

Proof. The case $N = 1$ is trivial, because then $p_{FC} = p_{FP} = 1$. Let now suppose $N > 1$. Since $p_{FC} = 1$ if $q > N$ and, for N, q such that $N \geq q > 2$, it results:

$$\frac{N!}{(N-q)!N^q} = \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right) < \left(1 - \frac{1}{N}\right)^q ,$$

from Corollary 7 it follows that $p_{FC} > p_{FP}$ for any (M, N) uniform hash function. On the other hand, from Theorems 8 and 10, it follows that $p_{FC} - p_{FP}$ is

a Schur-convex function in the set S defined by (14). Since

$$\min_{S}(p_1, p_2, ..., p_N) = \left(\frac{1}{N}, \frac{1}{N}, ..., \frac{1}{N}\right),$$

that concludes the proof. ■

7 Reductions among problems

In our algorithmic approach to hash function security, it can be useful to know how many chances we have in solving one of the problems of finding preimages, second preimages or collisions by using an algorithm designed to solve a different problem among those. That task pertains to the notion of *reduction among problems*. The general notion of reduction is from complexity theory, where it denotes a procedure which solves a computational problem (the one to be reduced) by using functionally-specified procedure calls to another computational problem. Since we are concerned with reductions of a searching problem $\mathcal{S}$ to a searching problem $\mathcal{S}'$, it is suffices to consider Karp-type reductions, which operate as follows: given an instance u for $\mathcal{S}$, the reduction computes an instance u' for $\mathcal{S}'$, invokes a procedure to compute a solution s' for $\mathcal{S}' = \mathcal{S}'(u')$ and then uses s' to compute a solution s for $\mathcal{S} = \mathcal{S}(u)$. Of course, we are only interested in efficient reductions, which in complexity theory means that the reduction takes polynomial time. Following the preliminary work in [15], we reformulate the above in terms of the success probabilities of Las Vegas algorithms as follows:

Definition 7 *Let $\mathcal{S}$ and $\mathcal{S}'$ be two problems for which Las Vegas solving algorithms exist. $\mathcal{S}$ is said to be ε-reducible to $\mathcal{S}'$ if, given a Las Vegas (p, q)-algorithm ALG which is optimal for $\mathcal{S}'$, it is possible to construct a Las Vegas $(\varepsilon p, q)$-algorithm which optimally solves $\mathcal{S}$ by invoking ALG as procedure.*

As shown in [15], the most interesting questions are those concerning reduction from the collision problem to the preimage problem. It turns out that, for generic Las Vegas algorithms, the amount of reduction can greatly vary, depending on the probability distribution of the hash function under investigation and on the assumptions (in term of success probabilities) that are made about the algorithm ALG designed to solve the preimage searching problem. If, for example, we suppose that $M >> N$ and that ALG has the same success probability to find a preimage of y for any $y \in Y$, then ε can be made very close to one by increasing the computational effort (i.e. the number q of hash evaluations) for ALG. On the other hand, we might have a Las Vegas preimage-finding algorithm that is successful only if $y \in Y$ has a unique preimage, and in that case $\varepsilon = 0$. The following results concern the reduction from collision to preimage in the special case of LVBF algorithms.

Theorem 12 *Let $h : X \to Y$ be an (M, N) hash function with probability distribution (1). Then the collision searching problem for h is ε-reducible to the*

preimage searching problem with

$$\varepsilon = \frac{q}{q+1} \frac{1 - \frac{N}{M} - \sum_{n=1}^{N} (1-p_n)^q \left(p_n - \frac{1}{M}\right)}{1 - \frac{1}{N} \sum_{n=1}^{N} (1-p_n)^q}, \tag{17}$$

where q represents the number of evaluations of h.

Proof. The following is a LVBF algorithm which solves the collision searching problem by calling `FindPreimage` as procedure:

```
Coll2Preimg(h,q)
begin
success=false
   seed=random()
   x=rho(counter,seed)
   y=h(x)
   xc=FindPreimage(h,y,q)
   if (xc!=NULL AND x!=xc)
      success=true
   endif
   return success
end
```

It should be clear that `Coll2Preimg` is an optimal LVBF algorithm for finding a collision by a call to `FindPreimage`. The success probability $p_{C2P}(y_n)$ of `Coll2Preimg` for $y = y_n$ is given by:

$$p_{C2P}(y_n) = P^*(h(x) = y_n)\, p_{FP}(y_n)\, P^*(x' \neq x / h(x) = h(x'))$$

where $p_{FP}(y_n)$ is the success probability of algorithm `FindPreimage` for $y = y_n$. Thus from (7) and (9) it follows that:

$$p_{C2P}(y_n) = p_n(1-(1-p_n)^q)\left(1 - \frac{1}{Mp_n}\right) = (1-(1-p_n)^q)\left(p_n - \frac{1}{M}\right)$$

and the success probability of `Coll2Preimg` is given by:

$$p_{C2P} = \sum_{n=1}^{N} p_{C2P}(y_n) = 1 - \frac{N}{M} - \sum_{n=1}^{N} (1-p_n)^q \left(p_n - \frac{1}{M}\right) \tag{18}$$

On the other hand, `Coll2Preimg` computes at most $q+1$ evaluations of the hash function h, since one evaluation is spent to calculate the input y for `FindPreimage` and q are the evaluations of h computed by the latter algorithm in the worst case. Thus `Coll2Preimg` is a $(p_{C2P}, q+1)$-LVBF algorithm and from (6) it follows the equality (17). ■

For a uniform hash function, clearly we obtain the same result derived in [15] under the assumption that the algorithm designed to solve the preimage problem has the same success probability for any $y \in Y$; namely:

Corollary 13 *Let $h : X \to Y$ be an (M, N) uniform hash function. Then the collision searching problem for h is ε-reducible to the preimage searching problem with*

$$\varepsilon = \frac{q}{q+1}\left(1 - \frac{N}{M}\right),$$

where q is the number of evaluations of h.

If h is a uniform hash function, then $p_n = 1/N$ for all $1 \leq n \leq N$ and from the previous theorem it follows that the collision searching problem is ε-reducible to the preimage searching problem with:

$$\varepsilon = \frac{q}{q+1}\frac{1 - \frac{N}{M} - \sum\limits_{n=1}^{N}\left(1 - \frac{1}{N}\right)^q\left(\frac{1}{N} - \frac{1}{M}\right)}{1 - \frac{1}{N}\sum\limits_{n=1}^{N}\left(1 - \frac{1}{N}\right)^q} = \frac{q}{q+1}\left(1 - \frac{N}{M}\right)$$

This last theorem shows that we can have a better reducibility than in the uniform case only in a suitable "neighborhood" of $\mathbf{p} = (1/N, 1/N, ..., 1/N)$, which became smaller as the number of evaluations of the hash function increases.

Theorem 14 *Let q be an integer and let $h : X \to Y$ be an (M, N) hash function with a probability distribution (1) such that*

$$p_{[1]} \leq \frac{1}{N} + \frac{2}{q+1}\left(1 - \frac{1}{N}\right), \tag{19}$$

where $(p_{[1]}, p_{[2]}, ..., p_{[N]})$ is a decreasing reordering of $\mathbf{p} = (p_1, p_2, ..., p_N) \in S$ and S is the set defined by (14). Then the collision searching problem for h is ε-reducible to the preimage searching problem with

$$\varepsilon \geq \frac{q}{q+1}\left(1 - \frac{N}{M}\right).$$

Proof. The proof relies on the fact that the function $\varepsilon = \varepsilon(\mathbf{p})$ given by (17) is Schur-convex on the subset of S defined by the inequality (19). Indeed, from (17) it follows that

$$\frac{\partial \varepsilon}{\partial p_n}(p_n) = \frac{q}{q+1}\frac{(1-p_n)^{q-1}}{p_{FP}^2}\left[(q+1)p_{FP}p_n - \frac{qp_{FP}}{M} - \frac{qp_{C2P}}{N} - p_{FP}\right],$$

where p_{FP} and p_{C2P} are given by (6) and (18), respectively. Let now consider the function:

$$g(x) = (1-x)^{q-1}\left[(q+1)p_{FP}x - \frac{qp_{FP}}{M} - \frac{qp_{C2P}}{N} - p_{FP}\right].$$

Since

$$\frac{dg}{dx}(x) = q(1-x)^{q-2}\left[\frac{(q-1)p_{FP}}{M} + \frac{(q-1)p_{C2P}}{N} + 2p_{FP} - (q+1)p_{FP}x\right],$$

it follows that $\frac{\partial \varepsilon}{\partial p_n}$ is a decreasing sequence in n for any $(p_{[1]}, p_{[2]}, ..., p_{[N]}) \in S$ such that

$$p_{[1]} \leq \frac{2}{q+1} + \frac{q-1}{q+1}\left(\frac{1}{N}\frac{p_{C2P}}{p_{FP}} + \frac{1}{M}\right),$$

Thus, on the basis of Criterion 2, it follows the Schur-convexity of $\varepsilon = \varepsilon(\mathbf{p})$ on the subset of S defined by the previous inequality. That subset, because of Corollary 12 and the Schur-convexity of ε, contains the one defined by inequality (19). ■

8 Conclusion and future work

Our work provides a mathematical framework for the study of the basic security requirements for an hash function in the case of brute force attacks, which enable us to give rigorous proofs for some widely accepted conjectures in the theory of cryptographic hash functions.

As pointed out in [3], despite of the fact that hash functions are usually designed to resemble to random functions, if one wants the best possible protection against both birthday (which, in view of our results, implies protection against brute force attacks in general) and cryptanalytic attacks, one should design that function subject to being as uniform as possible. This last task is more difficult than designing a "simple" pseudorandom hash function, so that the latter remains the design goal and it is useful to have tools that enable designers to measure the impact of deviations from uniformity in the hash function outcomes and fine tune its output length if necessary.

In view of Theorem 6 and of the results of §§ 5 and 6, it suffices to compute expression (10) to assess how much secure a given hash function is with respect to brute force attacks. However, a direct computation of (10) poses a computational challenge, because of the very large values of N used in practically useful hash functions (e.g., $N = 2^{128} = O(10^{38})$ for MD5 and $N = 2^{160} = O(10^{48})$ for RIPEMID-160 and SHA-1). It turns out the importance of both suitable, simpler estimates for (10) and for its expectation value - as the ones derived in [3] - and the study of computational strategies and algorithms to evaluate them efficiently and accurately.

References

[1] M. Bellare, P. Rogaway, Random Oracles are Practical: a Paradigm for Designing Efficient Protocols, *Proceedings of the First Annual Conference on Computer and Communications Security.* ACM Press, 1993

[2] M. Bellare, O. Goldreich, S. Goldwasser, Incremental Cryptography: The Case of Hashing and Signing, *LNCS **839** - Crypto94 Proceedings.* Springer-Verlag, Y. Desmedt, ed., 1994

[3] M. Bellare, T. Kohno, Hash Function Balance and its Impact on Birthday Attacks, *LNCS **3027**- Advances in Cryptology-EUROCRYPT '04.* Springer-Verlag, C. Cachin and J. Camenisch eds., 2004

[4] O. Goldreich, *Foundations of Cryptography - Basic Tools.* Cambridge University Press, Cambridge 2001

[5] G.H. Hardy, J.E. Littlewood, G. Polya, *Inequalities* (1st ed.). Cambridge University Press, London 1934

[6] N.L. Johnson, S. Kotz, *Distributions in Statistics: Discrete Distributions.* Houghton Mifflin, Boston, 1969

[7] A.W. Marshall, I. Olkin, *Inequalities: Theory of Majorization and its Applications.* Academic Press, R. Bellmann ed., 1979

[8] A. Menezes, P. Oorschot, S. Vanstone, *Handbook of Applied Cryptography.* CRC Press, Boca Raton, 1997

[9] R.C. Merkle, *Secrecy, Authentication and Public Key Systems.* UMI Research Press, Ann Arbor, 1979

[10] B. Preneel, The State of Cryptographic Hash Functions, *LNCS **1561** - Lectures on Data Security: Modern Cryptology in Theory and Practice.* Springer-Verlag, Berlin, 1999

[11] M.O. Rabin, Digitalized Signatures, *Foundations of Secure Computations.* Academic Press, R. DeMillo, D. Dobkin, A. Jones and R. Lipton eds., 1978

[12] B. Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C.* John Wiley, New York, 1996

[13] C. E. Shannon, A mathematical theory of communication, *Bell System Technical Journal, vol. 27.* Bell Laboratories, 1948

[14] W. Stallings, *Cryptography and Network Security*: *Principles and Practice.* Prentice Hall, Upper Saddle River, 1999

[15] D.R. Stinson, *Some Observations on the Theory of Cryptographic hash Functions.* Preprint, January 2004 (http://www.cacr.math.uwaterloo.ca/~dstinson/pubs.html)