Some observations on the theory of cryptographic hash functions

D.R. Stinson
School of Computer Science
University of Waterloo
Waterloo Ontario, N2L 3G1, Canada
dstinson@uwaterloo.ca

January 16, 2004

Abstract

In this paper, we study issues related to the notion of "secure" hash functions. Several necessary conditions are considered, as well as a popular sufficient condition (the so-called random oracle model). We study the security of various problems that are motivated by the notion of a secure hash function. These problems are analyzed in the random oracle model, and we prove that the obvious trivial algorithms are optimal. As well, we look closely at reductions between various problems. In particular, we consider the important question "does preimage resistance imply collision resistance?". We provide partial answers to this question — both positive and negative! — based on uniformity properties of the hash function under consideration.

1 Introduction

Hash functions are of fundamental importance in cryptographic protocols (for a recent survey, see Preneel [9]). Security of hash functions has been considered in many papers, but it is still not completely clear what it means for a cryptographic hash function to be "secure". The notion of a secure hash function is closely tied to the most important application of hash functions, namely, signature schemes. In this regard, several properties of hash functions are clearly necessary in order for it to be considered secure. It is generally accepted that a secure hash function should be *one-way* (i.e., *preimage resistant*), *second preimage resistant* and *collision resistant*. There are additional properties that may be necessary when a hash function is used in conjunction with a particular signature scheme. A recently observed example is that of *zero-resistance*, which is a property required of a hash function used in the Digital Signature Algorithm (DSA) and the Elliptic Curve Digital Signature Algorithm (ECDSA) if these schemes are to be secure (see [4]).

On the other hand, there is a widely used idealized model in which protocols using hash functions can be proven secure. This is the *random oracle model*, which was introduced by Bellare and Rogaway in [1]. In this model, a hash function is modelled as a random function and an adversary is only permitted oracle access to the function. Clearly these are very strong assumptions – they are so strong, in fact, that it seems that a random oracle cannot realistically be implemented in practice. Nevertheless, the random oracle model is a very useful and "clean" mathematical model that permits formal security proofs of protocols to be given. We discuss these issues more thoroughly in Section 1.1.

There is a considerable gap between the above-mentioned necessary and sufficient security conditions for hash functions, and it is certainly an interesting problem to try and close these gaps. Progress along these lines has recently been reported in [4], where some security proofs for ECDSA are presented in the generic group model. In this model, it can be shown that there is a fairly small gap between the necessary and sufficient conditions imposed on hash functions for the ECDSA signature scheme to be proven secure.

We begin with a brief discussion of the random oracle model, in Section 1.1. One main point that we emphasize is that a proof of security in the random oracle model is nothing more or less than a proof of security against hash-generic attacks. Next, we give some definitions of hash functions in Section 1.2. Four problems relevant to the study of hash functions are presented in Section 1.3, and motivation for studying these problems is briefly mentioned in Section 1.4.

In Section 2, we study the "difficulty" of the four problems in the random oracle model. We state a fundamental "independence" property that holds in the random oracle model, and we use this independence property to prove that the obvious trivial algorithms are optimal (these are simple results, but we think they are worth recording formally).

We discuss reductions between the four problems in Section 3. In particular, we consider the important question "does preimage resistance imply collision resistance?". Our approach is to give an exact analysis of the success probability of a reduction between the two problems in terms of certain uniformity properties of the hash function. We believe that this approach is useful, in particular, because it avoids the problematic questions of asymptotic complexity that often arise in the context of cryptographic hash functions. We find that preimage resistance implies collision resistance under certain circumstances, such as, for example, when the hash function is "close to" uniform. However, there does not appear to exist any general reduction which works for an arbitrary hash function and an arbitrary preimage-finding oracle.

In the opposite direction, we note that Simon [10] showed that "no provable construction of a collision-intractible hash function can exist based on a 'black-box' one-way permutation". This shows, in a strong sense, that finding preimages cannot be reduced to finding collisions.

We emphasize that all the analyses done in this paper are exact analyses, very much in the vein of "exact security" as defined by Bellare and Rogaway (see, for example, [2]). We do not, in general, address questions about the asymptotic complexity of the problems and reductions we study.

1.1 The Random Oracle Model

In this section, we briefly discuss the random oracle model, and how proofs in this model should be interpreted.

The following result from [3] is a typical example of the kind of thing that can be proven in the random oracle model:

> "Full domain hash is secure against an adaptive chosen message attack with respect to existential forgeries in the random oracle model, provided that it is infeasible to invert a trapdoor one-way permutation, f."

The proof is actually a reduction: given a hypothetical forging algorithm, say A, for full domain hash (which is a signature scheme), that has some specified probability of success, say ϵ, we construct an inverting algorithm for f, which inverts the function f on randomly chosen inputs with some specified probability of success, say ϵ'. If ϵ' is not too much smaller than ϵ, and we believe that it is infeasible to invert f, then we have a contradiction, and we conclude that the hypothesized forging algorithm cannot exist.

So far, this is all pretty standard, and not too difficult to follow. However, the proof is taking place in the "random oracle model" which means that some additional assumptions are being made regarding the hash function being used, and the proof of security is valid only if these assumptions are valid.

The random oracle model is usually described by saying that the hash function is a random function. Of course, when full domain hash is actually used in practice, a particular hash function must be specified, and so the assumption used in the random oracle model is not valid. Indeed, there are examples of protocols that have been proven secure in the random oracle model, but which become insecure whenever the hash function used in the protocol is specified (see Canetti, Goldreich and Halevi [5] and Nielsen [8]). As a consequence, the random oracle model is somewhat controversial.

The correct way to interpret a proof of security for a protocol P in the random oracle model is to view it as a proof of security against certain types of attacks on the protocol P. More precisely, the proof shows that the protocol P is secure against what might be termed "hash-generic" attacks. This means that any attack which treats the hash function as a random function will not be successful (regardless of whether the hash function actually is a random function). In other words, it is better to think of a proof in the random oracle model as a proof in which we make an assumption about the attacking algorithm rather than an assumption about the hash function.

To summarize, a security proof in the random oracle model is precisely a proof of security against hash-generic algorithms. It is of course possible that an algorithm can break a protocol for some particular hash functions (or even for all possible hash functions) by somehow taking advantage of how the hash function is computed. A proof in the random oracle model is therefore no more than plausible evidence of security when the random oracle is replaced by a particular hash function. This was the "thesis" proposed by Bellare and Rogaway when they introduced the random oracle model in [1]. In favour of this thesis, it should be noted that no practical protocol proven secure in the random oracle model has been broken when used with a "good" hash function, such as SHA-1 (for a description of SHA-1, see, for example, [7, Chapter 9]).

1.2 Definitions of Hash Functions

We will use the following notations and definitions throughout the rest of this paper.

An (N, M) *hash function* is any function $f : X \to Y$, where X and Y are finite sets with $|X| = N$ and $|Y| = M$. Usually we assume that $N \geq 2M$. (A hash function with finite domain is also known as a *compression function.*) An (N, M) *hash family* is a set $\mathcal{F}$ of functions such that $f : X \to Y$ for each $f \in \mathcal{F}$, where $|X| = N$ and $|Y| = M$.

Let $\mathcal{F}^{X,Y}$ denote the set of all functions from X to Y. Clearly $|\mathcal{F}^{X,Y}| = M^N$. Using this notation, an (N, M) hash family is a subset of functions $\mathcal{F} \subseteq \mathcal{F}^{X,Y}$.

Suppose that $|X| = N$, $|Y| = M$ and $M|N$. A hash function $f \in \mathcal{F}^{X,Y}$ is said to be *uniform* if $|f^{-1}(y)| = N/M$ for all $y \in Y$.

Suppose that $|X| = N$ and $|Y| = M$. For any $f \in \mathcal{F}^{X,Y}$, define

$$\mathsf{UP}(f) = \{y \in Y : |f^{-1}(y)| = 1\}.$$

$\mathsf{UP}(f)$ consists of all elements $y \in Y$ that have unique preimages with respect to f ("UP" is an abbreviation for "unique preimage").

1.3 Four Problems

We define four problems which will be studied in the rest of the paper. In reference to Problem 1, we assume that $0 \in Y$.

Problem 1:	Zero preimage
Instance:	A hash function $f : X \to Y$.
Find:	$x \in X$ such that $f(x) = 0$.

Problem 2:	Preimage
Instance:	A hash function $f : X \to Y$ and an element $y \in Y$.
Find:	$x \in X$ such that $f(x) = y$.

Problem 3:	Second preimage
Instance:	A hash function $f : X \to Y$ and an element $x \in X$.
Find:	$x' \in X$ such that $x' \neq x$ and $f(x') = f(x)$.

Problem 4:	Collision
Instance:	A hash function $f : X \to Y$.
Find:	$x, x' \in X$ such that $x' \neq x$ and $f(x') = f(x)$.

1.4 Hash functions and signature schemes

In the usual "hash-then-sign" paradigm, a message is first hashed, and then the resulting *message digest* is signed. In this context, it is well-known that if an adversary can find a solution to Second preimage or Collision (for the given f), then a signature on a new message can be forged. Further, a solution to Preimage can lead to an attack on the RSA signature scheme. (For descriptions of all these attacks, see [7, p. 324].)

It seems to be less well known that attacks on certain signature schemes can be carried out if a prespecified element (as opposed to a random element $y \in Y$) can be inverted with respect to a given hash function f. It is shown in [4] that solving Zero preimage can permit attacks on DSA and ECDSA. We describe this attack on the DSA, which can be carried out if Zero preimage can be solved for the hash function being used (namely, SHA-1 or a variant of it, in the case of the Digital Signature Standard). This attack provides an existential forgery with respect to a key-only attack.

Briefly, the DSA employs appropriate primes p and q such that $q|(p-1)$. $\alpha \in \mathbb{Z}_p{}^*$ is an element of order q; $a \in \{1, \ldots, q-1\}$ is chosen randomly; and $\beta = \alpha^a \bmod p$. The public key is (p, q, α, β), and the value a is secret. f is any hash function taking on values in the set $\mathbb{Z}_q$.

A signature $(r, s) \in \mathbb{Z}_q \times \mathbb{Z}_q$ on a message m is valid if and only if the following DSA verification algorithm returns the value "accept".

Algorithm 1.1: *DSA-SignatureVerification*$(p, q, \alpha, \beta, (r, s), m)$

if $(r = 0)$ **or** $(s = 0)$
  then return (reject)
$w \leftarrow s^{-1} \bmod q$
$u_1 \leftarrow wf(m) \bmod q$
$u_2 \leftarrow wr \bmod q$
$v \leftarrow (\alpha^{u_1}\beta^{u_2} \bmod p) \bmod q$
if $v = r$
  then return (accept)
  else return (reject)

For the purposes of the attack, we assume that we know a value m such that $f(m) = 0$. In this situation, it is not difficult to see that (r, s) is a valid signature on m, where

$$r = s = \beta \bmod q,$$

provided that $\beta \not\equiv 0 \pmod q$. This is easily seen as follows: in Algorithm 1.1, we will compute $u_1 = 0$ and $u_2 = 1$. Then $v = \beta \bmod q = r$, so the signature is valid. (If $\beta \equiv 0 \pmod q$, then the attack does not work, because $(r, s) = (0, 0)$ is rejected as a valid signature by Algorithm 1.1.)

2 Algorithms in the Random Oracle Model

In this section, we consider the complexity of the four problems defined in §1.3 in the random oracle model. To reiterate, this means that the following two assumptions are made:

1. $f \in \mathcal{F}^{X,Y}$ is a randomly chosen function.
2. An algorithm is given only oracle access to f.

As a consequence of these two assumptions, it is obvious that the following *independence* property holds:

Property 2.1 *Suppose that $f \in \mathcal{F}^{X,Y}$ is chosen randomly, and let $x_1, \ldots, x_\ell \in X$. Suppose that the values $y_i = f(x_i)$ have been determined (by consulting an oracle for f) for $1 \leq i \leq \ell$. Then*

$$\mathbf{Pr}[f(x) = y | f(x_1) = y_1, \ldots, f(x_\ell) = y_\ell] = \frac{1}{M}$$

for all $x \in X \backslash \{x_1, \ldots, x_\ell\}$ and all $y \in Y$.

Note that Property 2.1 is expressed as a conditional probability, computed over all functions f that take on the specified values when $x = x_1, \ldots, x_\ell$. Property 2.1 is the key property used in all proofs involving the random oracle model, though it does not usually seem to be stated in an explicit form such as we have done.

In the rest of this section, we consider (ϵ, q) randomized algorithms for the four problems defined in §1.3. The parameter ϵ denotes the probability that the algorithm correctly solves the desired problem, and q denotes the number of oracle queries (i.e., evaluations of f) made by the algorithm. (We are not including time as an explicit parameter, though it would be straightforward to do so.)

We will perform average-case analyses of the algorithms. Averages will be computed over the following sources of randomness:

- the random coin tosses made by the algorithm,
- the random choice of $f \in \mathcal{F}^{X,Y}$,
- the random choice of $x \in X$, in the case of Second preimage,
- the random choice of $y \in Y$, in the case of Preimage.

The main observation of this section is that the trivial algorithms that evaluate $f(x)$ for q values of $x \in X$ are optimal in the model described above. In fact, the complexity is independent of the choice of the q values of x because we are averaging over all functions $f \in \mathcal{F}^{X,Y}$.

We first consider an algorithm that attempts to solve Zero preimage by evaluating f at q points.

Algorithm 2.1: *FindZeroPreimage*(f, q)

choose $X_0 \subseteq X, |X_0| = q$
for each $x \in X_0$
  do $\begin{cases} \textbf{if } f(x) = 0 \\ \quad \textbf{then return } (x) \end{cases}$
return (failure)

Theorem 2.1 *For any $X_0 \subseteq X$ with $|X_0| = q$, the success probability of Algorithm 2.1 is $\epsilon = 1 - (1 - 1/M)^q$.*

Proof. Let $X_0 = \{x_1, \ldots, x_q\}$. For $1 \leq i \leq q$, let let E_i denote the event "$f(x_i) = 0$". It follows from Property 2.1 that the E_i's are independent events, and $\mathbf{Pr}[E_i] = 1/M$ for all $1 \leq i \leq q$. Therefore, it holds that

$$\mathbf{Pr}[E_1 \vee E_2 \vee \cdots \vee E_q] = 1 - \left(1 - \frac{1}{M}\right)^q.$$

□

Algorithm 2.2: *FindPreimage*(f, y, q)

choose $X_0 \subseteq X, |X_0| = q$
for each $x \in X_0$
  do $\begin{cases} \textbf{if } f(x) = y \\ \quad \textbf{then return } (x) \end{cases}$
return (failure)

The success probability of Algorithm 2.2, for any fixed y, is the same as that of Algorithm 2.1. Therefore, the success probability averaged over all $y \in Y$ is identical, too.

Theorem 2.2 *For any $X_0 \subseteq X$ with $|X_0| = q$, the success probability of Algorithm 2.2 is $\epsilon = 1 - (1 - 1/M)^q$.*

Algorithm 2.3: *FindSecondPreimage*(f, x, q)

$y \leftarrow f(x)$
choose $X_0 \subseteq X \backslash \{x\}, |X_0| = q - 1$
for each $x_0 \in X_0$
  do $\begin{cases} \textbf{if } f(x_0) = y \\ \quad \textbf{then return } (x_0) \end{cases}$
return (failure)

The analysis of Algorithm 2.3 is similar to the two previous algorithms. The only difference is that we require an "extra" application of f to compute $y = f(x)$ for the input value x.

Theorem 2.3 *For any $X_0 \subseteq X \backslash \{x\}$ with $|X_0| = q - 1$, the success probability of Algorithm 2.3 is $\epsilon = 1 - (1 - 1/M)^{q-1}$.*

Algorithm 2.4: *FindCollision*(f, q)

choose $X_0 \subseteq X \backslash \{x\}, |X_0| = q$
for each $x \in X_0$
  do $y_x \leftarrow f(x)$
if $y_x = y_{x'}$ for some $x' \neq x$
  then return (x, x')
  else return (failure)

Algorithm 2.4 is analyzed using the standard "birthday paradox".

Theorem 2.4 *For any $X_0 \subseteq X$ with $|X_0| = q$, the success probability of Algorithm 2.4 is*

$$\epsilon = 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-q+1}{M}\right).$$

Proof. Let $X_0 = \{x_1, \ldots, x_q\}$. For $1 \leq i \leq q$, let E_i denote the event "$f(x_i) \notin \{f(x_1), \ldots, f(x_{i-1})\}$". Using induction, it follows from Property 2.1 that $\mathbf{Pr}[E_1] = 1$ and

$$\mathbf{Pr}[E_i | E_1 \wedge E_2 \wedge \cdots \wedge E_{i-1}] = \frac{M-i+1}{M},$$

for $2 \leq i \leq q$. Therefore, we have that

$$\mathbf{Pr}[E_1 \wedge E_2 \wedge \cdots \wedge E_q] = \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-q+1}{M}\right).$$

The result follows. ∎

Remark. We note that the analyses of the four algorithms remain unchanged if $\mathcal{F}^{X,Y}$ is replaced by any q-wise independent hash family. (A *q-wise independent hash family* is one in which Property 2.1 is satisfied for all $\ell \le q-1$.)

We already mentioned that Algorithms 2.1 – 2.4 are optimal. By this, we mean that there do not exist algorithms where ϵ is larger (as a function of q) than it is in these algorithms. We first provide a proof of this in the case of Algorithm 2.1.

Theorem 2.5 *Suppose that A is an (ϵ, q) algorithm in the random oracle model for* Zero preimage. *Then*

$$\epsilon \le 1 - \left(1 - \frac{1}{M}\right)^q.$$

Proof. The proof is by induction on q. Suppose first that $q = 1$. For any $x \in X$, it holds that

$$\mathbf{Pr}[f(x) = 0] = \frac{1}{M}.$$

Therefore $\epsilon \le 1 - (1 - 1/M)^q$ when $q = 1$.

Now, assume that the bound holds when $q = k-1$. Consider any algorithm, say A, in which $q = k$. Suppose that A computes $f(x_1), \ldots, f(x_k)$ in that order. Let p denote the probability that $f(x_i) = 0$ for some $i \le k-1$. By the induction hypothesis, we have that $p \le 1 - (1 - 1/M)^{k-1}$.

Now consider the kth query, which is to evaluate $f(x_k)$. It follows from Property 2.1 that $\mathbf{Pr}[f(x_k) = 0] = 1/M$. The probability that $f(x_i) = 0$ for some $i \le k$ is therefore at most

$$\begin{aligned} p + \frac{1-p}{M} &= p\left(\frac{M-1}{M}\right) + \frac{1}{M} \\ &\le \left(1 - \left(1 - \frac{1}{M}\right)^{k-1}\right)\left(\frac{M-1}{M}\right) + \frac{1}{M} \\ &= 1 - \left(1 - \frac{1}{M}\right)^{k}. \end{aligned}$$

By induction, the desired bound is proven. ∎

Algorithms 2.2 and 2.3 can be proven to be optimal using similar arguments. Therefore we proceed directly to Algorithm 2.4.

Theorem 2.6 *Suppose that A is an (ϵ, q) algorithm in the random oracle model for* Collision, *where $q \ge 2$. Then*

$$\epsilon \le 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-q+1}{M}\right).$$

Proof. The proof is by induction on q. Suppose first that $q = 2$. In this case, $\epsilon = 1/M$ from Property 2.1, and the stated bound holds.

Now, assume that the bound holds when $q = k-1$. Consider any algorithm, say A, in which $q = k$. Suppose that A computes $f(x_1), \ldots, f(x_k)$ in that order. For $1 \le i \le q$, let E_i denote the event "$f(x_i) \notin \{f(x_1), \ldots, f(x_{i-1})\}$". Let p denote the probability that $f(x_i) = f(x_j)$ for some i and j with $1 \le i < j \le k-1$. Then

$$1 - p = \mathbf{Pr}[E_1 \wedge E_2 \wedge \cdots \wedge E_{k-1}].$$

Further, by the induction hypothesis, we have that

$$p \leq 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-(k-1)+1}{M}\right).$$

We are interested in the probability

$$\begin{aligned}
&1 - \mathbf{Pr}[E_1 \wedge E_2 \wedge \cdots \wedge E_k] \\
&\quad= 1 - \mathbf{Pr}[E_1 \wedge E_2 \wedge \cdots \wedge E_{k-1}] \times \mathbf{Pr}[E_k | E_1 \wedge E_2 \wedge \cdots \wedge E_{k-1}] \\
&\quad= 1 - (1-p)\left(\frac{M-k+1}{M}\right) \qquad \text{from Property 2.1} \\
&\quad= p\left(\frac{M-k+1}{M}\right) + \frac{k-1}{M} \\
&\quad\leq \left(1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-(k-1)+1}{M}\right)\right)\left(\frac{M-k+1}{M}\right) + \frac{k-1}{M} \\
&\quad= 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-k+1}{M}\right).
\end{aligned}$$

The result follows by induction. ∎

Remark. For any algorithm for Collision, we have that $\epsilon = 0$ whenever $q = 1$.

3 Reductions among the Problems

A hash function f for which Preimage is difficult to solve is often termed *preimage resistant* (or *one-way*). Similarly, f is *second preimage resistant* if Second preimage is hard to solve, etc. One problem with this type of terminology is that, in the case of a fixed hash function, the phrase "hard to solve" can be made mathematically rigourous only in the context of concrete complexity (i.e., when the computations are performed using circuits, decision trees, etc.). Furthermore, there seems to be no satisfactory formalization of the idea of collision resistance for a fixed hash function, because there always exists an algorithm that simply outputs a collision (if one exists). Alternatively, one can talk about infeasibility of problems in an asymptotic context, but this requires having an infinite "keyed" family of functions.

We prefer to instead study reductions among the different problems. We can study reductions of problems without worrying about having to define terms such as "hard to solve" or "infeasible", even if the hash function under consideration is fixed. If we can efficiently reduce a problem P to a problem Q, then we know that, roughly speaking, solving an instance of P is no harder than solving the corresponding instance of problem Q. Consequently, if we believe that solving an instance of P is "infeasible", then we would also believe that solving the corresponding instance of Q is "infeasible".

The main question we investigate in this section is whether a preimage resistant hash function is collision resistant. Of course, this is rephrased in terms of reductions by asking if Collision can be reduced to Preimage. In other words, assuming we have an algorithm (or oracle) to solve Preimage, can we solve Collision? This is a particularly interesting question, because it turns out that the answer can be either positive or negative, depending on the assumptions that are made about the oracle for Preimage, and on certain uniformity properties of the hash function under investigation.

We will study possibly randomized Turing reductions, usually assuming that the hash function f is fixed. The reductions are required to be black-box reductions that treat the hash function f

as an oracle, and the success probability ϵ is computed over random coin flips in the algorithms in question, as well as random choices of the input x or y (as discussed in Section 2).

In general, reductions are most often used in the context of asymptotic complexity, but it is meaningful to consider them in a "finite" context, as well. We will be performing exact, combinatorial analyses of the success probabiltites of the reductions, and we are not making any claims or statements about asymptotic complexity.

Our first comment is that there does not seem to be any obvious reduction from Zero preimage to Preimage or vice versa. Intuitively, it does not seem that being able to invert (with respect to a given hash function f) a random y should be related in any meaningful way to being able to invert the specific value $y = 0$. It is conceivable that inverting $y = 0$ could be easier or harder than inverting a random y.

The most obvious reduction is from Collision to Second preimage. Suppose that *FindSecondPreimage* is any (ϵ, q) algorithm for Second preimage. If we choose $x \in X$ at random and run *FindSecondPreimage*(x), then we obtain a collision with probability ϵ. Thus we obtain the following well-known and easy result.

Theorem 3.1 *Let $f : X \to Y$ be any hash function. If there exists an (ϵ, q) algorithm that solves Second preimage for the hash function f, there exists an (ϵ, q) algorithm that solves Collision for the hash function f.*

It is noted in [4] that it is not necessarily the case that it is always possible to reduce Collision to Zero preimage. For example, it might happen that $0 \in \mathsf{UP}(f)$ for a given hash function f. In this situation, knowing the (unique) preimage of 0 is not of any obvious help in finding a collision. However, as pointed out in [4], an algorithm that succeeds in finding two different preimages of 0 automatically solves Collision.

3.1 Reducing Collision to Preimage

The most interesting questions are those concerning reductions from Collision to Preimage. The obvious method to attempt to construct a collision by using a preimage-finding algorithm is presented in Algorithm 3.1.

Algorithm 3.1: *CollisionToPreimage*(f)

external *FindPreimage*
choose $x \in X$ uniformly at random
$y \leftarrow f(x)$
if (*FindPreimage*$(f, y) = x'$) **and** $(x' \neq x)$ **and** $(f(x') = y)$
  then return (x, x')
  else return (failure)

For algorithms solving Preimage with probability $\epsilon = 1$, the following is proven in [11, Theorem 4.5].

Theorem 3.2 *Let $f : X \to Y$ be any hash function. If there exists a $(1, q)$ algorithm that solves* Preimage *for the hash function f, then Algorithm 3.1 is a $(1 - M/N, q + 2)$ algorithm that solves* Collision *for the hash function f.*

Remark. Checking that $f(x') = y$ is unnecessary when $\epsilon = 1$, so the number of oracle queries can be reduced to $q + 1$ in this case.

We prove a more general result now. For all $y \in Y$, let

$$p_y = \mathbf{Pr}[f(x) = y],$$

where $x \in X$ is chosen uniformly at random. It is clear that

$$p_y = \frac{|f^{-1}(y)|}{N}.$$

Further, suppose that *FindPreimage* is an (ϵ, q) algorithm that solves Preimage with respect to the fixed hash function f. Recall from Section 2 that the success probability of *FindPreimage* is defined with respect to a random choice of $y \in Y$. For each $y \in Y$, let

$$q_y = \mathbf{Pr}[\mathit{FindPreimage}(y) \text{ succeeds}].$$

Since $y \in Y$ is chosen randomly and *FindPreimage* has success probability equal to ϵ, it must be the case that

$$\sum_{y \in Y} q_y = M\epsilon.$$

Note also that $q_y = 0$ if $p_y = 0$.

We can now determine the success probability of Algorithm 3.1.

Theorem 3.3 *Let $f : X \to Y$ be a hash function, and suppose that FindPreimage is an (ϵ, q) algorithm that solves* Preimage *for the hash function f. Suppose that p_y and q_y $(y \in Y)$ are as defined above. Then Algorithm 3.1 is an $(\epsilon', q + 2)$ algorithm that solves* Collision *for the hash function f, where*

$$\epsilon' = \sum_{y \in Y} q_y p_y - \frac{M\epsilon}{N}.$$

Proof. For all $x \in X$, define $x \sim x'$ if $f(x) = f(x')$. Clearly $\sim$ is an equivalence relation. Define

$$[x] = f^{-1}(f(x)) = \{x' \in X : f(x') = f(x)\}$$

for all $x \in X$, and define

$$C = \{[x] : x \in X\}.$$

C is a partition of X because $\sim$ is an equivalence relation; the subsets in C are the equivalence classes under $\sim$. For every $c \in C$, define y_c to be the unique element of Y such that $f(x) = y_c$ for all $x \in c$.

Suppose that $x \in X$ is chosen uniformly at random, and $f(x) = y$. Then it is easy to see that

$$\mathbf{Pr}[\mathit{FindPreimage}(f, y) = x] = \frac{q_y}{|[x]|}.$$

Now we calculate the success probability ϵ' as follows:

$$\begin{aligned}
\epsilon' &= \frac{1}{N}\sum_{x\in X}\frac{q_{f(x)}(|[x]|-1)}{|[x]|}\\
&= \frac{1}{N}\sum_{c\in C}\frac{|c|-1}{|c|}\sum_{x\in c}q_{f(x)}\\
&= \frac{1}{N}\sum_{c\in C}q_{y_c}(|c|-1)\\
&= \frac{1}{N}\sum_{c\in C}q_{y_c}(Np_{y_c}-1)\\
&= \frac{1}{N}\sum_{y\in Y}q_y(Np_y-1)\\
&= \sum_{y\in Y}q_yp_y-\sum_{y\in Y}\frac{q_y}{N}\\
&= \sum_{y\in Y}q_yp_y-\frac{M\epsilon}{N}.
\end{aligned}$$

□

We look at some consequences of the above result. First, we observe that Theorem 3.2 is an immediate corollary of Theorem 3.3, obtained by letting $q_y = 1$ for all $y \in Y$. More generally, if $q_y = \epsilon$ for all y (as is typically required in a Las Vegas algorithm), then we obtain the following corollary.

Corollary 3.4 *Let $f : X \to Y$ be a hash function, and suppose that FindPreimage is an (ϵ, q) algorithm that solves* Preimage *for the hash function f with probability ϵ for every $y \in Y$. Then Algorithm 3.1 is an $(\epsilon', q+2)$ algorithm that solves* Collision *for the hash function f, where*

$$\epsilon' = \epsilon\left(1-\frac{M}{N}\right).$$

On the other hand, it is possible to produce "pathological" examples where $\epsilon' = 0$ (see, for example, [7, p. 330]). We present a class of such examples now. Informally, the underlying idea is that, in a worst-case scenario, we might have a preimage-finding algorithm that is successful if and only if $y \in \mathsf{UP}(f)$. In this case, Algorithm 3.1 will never find a collision! In terms of the q_y's, this is described as follows:

$$q_y = \begin{cases} 1 & \text{if } y \in \mathsf{UP}(f)\\ 0 & \text{otherwise.}\end{cases}$$

Denote $\mu = |\mathsf{UP}(f)|$; then $\epsilon = \mu/M$. However, it is easily seen using Theorem 3.3 that $\epsilon' = 0$ in this situation.

It is clear that the "problem" is caused by elements that have a unique preimage under f. The following general bound can be proven about this reduction, given a specified value for the quantity $|\mathsf{UP}(f)|$.

Corollary 3.5 *Let $f : X \to Y$ be a surjective hash function, and suppose that FindPreimage is an (ϵ, q) algorithm that solves* Preimage *for the hash function f, where $M\epsilon$ is an integer. Denote $\mu = |\mathsf{UP}(f)|$. Then Algorithm 3.1 is an $(\epsilon', q{+}2)$ algorithm that solves* Collision *for the hash function f, where*

$$\epsilon' \geq \frac{M\epsilon - \mu}{N}.$$

Proof. We have that $\epsilon = j/M$, where j is a positive integer. Without loss of generality, suppose that $Y = \{1, \dots, M\}$ and $0 < p_1 \leq \cdots \leq p_M$. Note that $p_y = a_y/N$, where a_y is a positive integer for all $1 \leq y \leq M$.

Given the distribution $p_1, \dots, p_M$, the quantity

$$\epsilon' = \sum_{y \in Y} q_y p_y - \frac{M\epsilon}{N} = \sum_{y \in Y} q_y p_y - \frac{j}{N}$$

is minimized by defining the q_i's as follows:

$$q_y = \begin{cases} 1 & \text{if } 1 \leq y \leq j \\ 0 & \text{if } j+1 \leq y \leq M. \end{cases}$$

With the above choice of q_y's, we have

$$\epsilon' = \sum_{y=1}^{j} p_y - \frac{j}{N}.$$

Now, assume that $\mu \leq j$ (otherwise, the bound is negative and the result holds trivially). Then it is easily seen that ϵ' is minimized by defining the p_y's (for $1 \leq y \leq j$) as follows:

$$p_y = \begin{cases} 1/N & \text{if } 1 \leq y \leq \mu \\ 2/N & \text{if } \mu+1 \leq y \leq j. \end{cases}$$

With the above choice of p_y's, we have

$$\epsilon' = \frac{\mu + 2(j-\mu) - j}{N} = \frac{M\epsilon - \mu}{N}.$$

□

Remark. Similar results can be proven without assuming that $M\epsilon$ is an integer. They are just a bit messier to write down.

The above result is quite weak, but it is essentially the strongest result that can be proved under the given assumptions. One way to obtain a tighter bound on the success probability of the reduction is to make the stronger assumption that the cardinalities of the preimages $f^{-1}(y)$ ($y \in Y$) are all roughly the same size (i.e., f is "close to" uniform).

Corollary 3.6 *Let $f : X \to Y$ be a hash function, and suppose that FindPreimage is an (ϵ, q) algorithm that solves* Preimage *for the hash function f. Let $\delta > 0$ and suppose that*

$$|f^{-1}(y)| \geq \frac{N(1-\delta)}{M}$$

for every $y \in Y$. Then Algorithm 3.1 is an $(\epsilon', q+2)$ algorithm that solves Collision *for the hash function f, where*

$$\epsilon' \geq \epsilon \left(1 - \delta - \frac{M}{N}\right).$$

Proof. We have that $p_y \geq (1-\delta)/M$ for all y. Therefore, it holds that

$$\epsilon' = \sum_{y \in Y} q_y p_y - \frac{M\epsilon}{N} \geq \left(\frac{1-\delta}{M}\right) \sum_{y \in Y} q_y - \frac{M\epsilon}{N} = \epsilon \left(1 - \delta - \frac{M}{N}\right).$$

□

Remark. In the case where f is uniform, we have $\delta = 0$ and then

$$\epsilon' \geq \epsilon \left(1 - \frac{M}{N}\right).$$

So far, we have treated f as a fixed hash function. But what if we analyze the average success probability of the reduction over all hash functions $f \in \mathcal{F}^{X,Y}$? (In other words, we regard f as a random hash function.) It turns out that the hypotheses of Corollary 3.6 are likely to be satisfied for a random hash function if M is somewhat larger than N. This is shown in the following theorem using an appropriate tail inequality for the binomial distribution.

Theorem 3.7 *Suppose that $f : X \to Y$ is a random hash function (i.e., f is chosen uniformly at random from $\mathcal{F}^{X,Y}$). Then the probability that*

$$|f^{-1}(y)| \geq \frac{N(1-\delta)}{M}$$

for every $y \in Y$ is at least $1 - M\, e^{-\delta^2 N/(2M)}$.

Proof. For any $y \in Y$, it is clear that

$$\Pr[|f^{-1}(y)| = i] = \binom{N}{i} \left(\frac{1}{M}\right)^i \left(1 - \frac{1}{M}\right)^{N-i}.$$

The probability that $|f^{-1}(y)| \leq k$ is

$$\sum_{i=0}^{k} \binom{N}{i} \left(\frac{1}{M}\right)^i \left(1 - \frac{1}{M}\right)^{N-i}.$$

Taking $k = N(1-\delta)/M$ and applying a standard tail inequality for for the binomial distribution ([6, p. 502]), we see that the probability that $|f^{-1}(y)| \leq N(1-\delta)/M$ is at most $e^{-\delta^2 N/(2M)}$.

There are M possible values for $y \in Y$. Therefore, the probability that $|f^{-1}(y)| \leq N(1-\delta)/M$ for at least one $y \in Y$ is at most $M\, e^{-\delta^2 N/(2M)}$. The probability that all inverse images have cardinality not less than $N(1-\delta)/M$ is at most $1 - M\, e^{-\delta^2 N/(2M)}$. □

To illustrate one application of Theorem 3.7, let us take $\delta = 1/2$ and $N = 10M \log_e M$ (so N is "moderately larger" than M). Then we obtain the following corollary.

Corollary 3.8 *Suppose that $f : X \to Y$ be a random hash function, where $N = 10M \log_e M$. Suppose that FindPreimage is an (ϵ, q) algorithm that solves* Preimage *for the hash function f. Then, with probability at least $1 - 1/M^{.25}$, Algorithm 3.1 is an $(\epsilon', q+2)$ algorithm that solves* Collision *for the hash function f, where*

$$\epsilon' \geq \epsilon \left(\frac{1}{2} - \frac{1}{10 \log_e M} \right).$$

4 Conclusion

In Section 1, we discussed notions of security relating to hash functions. Four relevant problems were defined, namely, Preimage, Zero preimage, Second preimage and Collision. The random oracle model was also defined, and we gave an interpretation of security in the random oracle model.

In Section 2, we presented algorithms to solve the four problems, and proved that these algorithms were optimal in the random oracle model.

To summarize the results of Section 3, we have seen that it is possible to obtain a "good" reduction from Collision to Preimage for a given hash function $f : X \to Y$, provided that either of the following two assumptions are satisfied:

- the hash function f is "close to uniform"
- the oracle for Preimage has a "good" success probability ϵ for every possible input $y \in Y$.

Neither of these assumptions are entirely satisfactory. The first assumption is likely to be true with high probability for random hash functions; however, it seems to be impossible to verify for hash functions used in practice. The second assumption ignores the possibility that there could exist practical preimage-finding algorithms that are successful on some (but not all) inputs. In light of these results, we suggest that it is best to require both collision resistance and preimage resistance as necessary properties for a hash function to be considered secure, due to the lack of a completely satisfactory reduction from the problem of finding collsions to the problem of finding preimages.

Acknowledgements

Thanks to Dan Brown and the referee for helpful comments. D.R. Stinson's research is supported by NSERC grant RGPIN #203114-02.

References

[1] M. Bellare and 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, pp. 62–73.

[2] M. Bellare and P. Rogaway. Optimal asymmetric encryption. *Lecture Notes in Computer Science*, **950** (1995), 92–111 (Advances in Cryptology – EUROCRYPT '94.)

[3] M. Bellare and P. Rogaway. The exact security of digital signatures: how to sign with RSA and Rabin. *Lecture Notes in Computer Science*, **1070** (1996), 399–416 (Advances in Cryptology – EUROCRYPT '96.)

[4] D. Brown. Generic groups, collision resistance and ECDSA. Research report CORR 2002-06, Department of Combinatorics and Optimization, University of Waterloo.

[5] R. Canetti, O. Goldreich and S. Halevi. The random oracle methodology, revisited. *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, ACM Press, 1998, pp. 209–218.

[6] D.E. Knuth. *The Art of Computer Programming, Third Edition*, Addison-Wesley, 1997.

[7] A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997.

[8] J.B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: the non-committing encryption case. *Lecture Notes in Computer Science*, **2442** (2002), 111–126 (Advances in Cryptology – CRYPTO 2002.)

[9] B. Preneel. The state of cryptographic hash functions. *Lecture Notes in Computer Science*, **1561** (1999), 158–182 (Lectures on Data Security: Modern Cryptology in Theory and Practice).

[10] D.R. Simon. Finding collisions on a one-way street: can secure hash functions be based on general assumptions? *Lecture Notes in Computer Science*, **1403** (1998), 334–345 (Advances in Cryptology – EUROCRYPT '98.)

[11] D.R. Stinson. *Cryptography: Theory and Practice, Second Edition*, CRC Press, 2002.