New Constructions for Secure Hash Functions
(Extended Abstract)

William Aiello[1], Stuart Haber[2], and Ramarathnam Venkatesan[3⋆]

[1] BellCore (aiello@bellcore.com)
[2] Surety (stuart@surety.com)
[3] Microsoft Research (venkie@microsoft.com)

Abstract. We present new, efficient and practical schemes for construction of collision-resistant hash functions, and analyze some simple methods for combining existing hash-function designs so as to enhance their security.
In our new constructions, we first map the input to a slightly longer string using a primitive we introduce called *secure stretch functions*. These are length-increasing almost surely injective one-way functions that sufficiently randomize their inputs so that it is hard for an adversary to force the outputs to fall into a target set. Then we apply a *compression function* to the output of the stretch function. We analyze the security of these constructions under different types of assumptions on both stretch and compression functions. These assumptions combine random-function models, intractability of certain "biasing" tasks, and the degeneracy structure of compression functions. The use of stretching seems to allow reduced requirements on the compression function, and may be of independent interest.
These constructions allow one to use popular and efficient primitives such as MD5, SHA-1, and RIPEMD that may exhibit weaknesses as collision-resistant functions. But no attacks are currently known on their one-way and randomizing properties, when they are used as stretch functions as in our constructions. There are several collision-resistant hash functions based on DES for which there are no known effective attacks, but which are too slow for most practical applications. Our use of stretch functions enable us to base our compression function on DES so that the resulting hash function achieves practical speeds: a test implementation runs at 40% of the speed of MD5.
We also suggest some imperfect random-oracle models, showing how to build better primitives from given imperfect ones. In this vein, we also analyze how to defend against a collision-finding adversary for a given primitive by building "independent" primitives.

1 Introduction

In this work we present new and practical constructions for secure hash functions, and analyze their security. In addition, we present simple methods for combining existing hash-function designs so as to enhance their security. There is a compelling need for better understanding of the principles of secure hash-function design.

⋆ Part of this work was done while with Bellcore and Surety.

S. Vaudenay (Ed.): Fast Software Encryption – FSE'98, LNCS 1372, pp. 150–167, 1998.

Many cryptographic procedures that handle very long bit-strings make use of a hash function. The security of these procedures relies on the *collision resistance* of the hash function in use, or on the function's "randomizing" effect. A hash function f is collision-resistant if it is infeasible to find a pair of distinct arguments $x \neq x'$ such that $f(x) = f(x')$.

There are several approaches to the design of such hash functions. While it is not known whether any current designs achieve the desired properties, they generally fall into two categories: designs based on an existing block cipher (or other cryptographic primitive), and custom designs "from scratch."

Customized hash functions: There have been a number of proposals for a "practical" secure hash function, one that admits fast software implementations and for which it is hoped that the cost of computing hash collisions is infeasible in practice [Riv 90,Riv 92,NIST 94,BP 95,DBP 96,AB 96]. Several of these are in widespread use. However, the general design principles for cryptographic hash functions are not well understood. As in the case of block ciphers, in practice a good hash function is simply one that survives the current attacks. Recent collision-finding attacks due to Dobbertin, using a differential approach, have been successful against RIPEMD, MD4, and MD5 [Dob 97,Dob 96a,Dob 96b]. More recently, even the one-wayness of MD4 has been challenged [Pre 97,Dob 98].

One approach would be to try to build on existing primitives. For example, one can concatenate the outputs of two different hash functions, hoping that the two functions behave "independently" (see [Pre 93, §2.4.5] and certain commercial designs, e.g. [Sur 95]). But one's hope is weakened by a cursory look at the source code for the popular hash functions, and even more so by Dobbertin's attacks on MD4-256, which derives two 128-bit values in this manner [Dob 96a].

Current methods to extend or strengthen previous designs include the following: increase the number of rounds (as in MD5); add some coding or scrambling steps (as in SHA-1); increase the buffer size and make the mixing step vary with the round. All of these are natural attempts to increase the security of a hash-function design, but an analysis based on a set of plausible heuristic assumptions would better enhance our confidence in the result. An example of such an assumption is the ideal-cipher model for DES, discussed below.

Hash functions from (ideal) ciphers: Another well studied approach (see e.g., [MMO 85,BC$^+$ 88,Merk 89]) bases the design on an existing trusted block cipher. For security assessments of such schemes see [Merk 89,KP 97,PGV 93a]. (For this and other questions about cryptographic hash functions, [Pre 93] and [MOV 97, chap. 9] are excellent references.) Unfortunately, these designs yield implementations using DES that are slower than MD5 (for example) almost by an order of magnitude, making them unacceptable for many applications. The usual measure of the efficiency of a design based on an n-bit cipher is its *rate*, defined as the number of n-bit blocks of data compressed per applications of the cipher. (Sometimes, as in [Pre 93], "rate" is used to mean the inverse of this ratio.) One of the suggestions here enables us to increase the rate significantly, yielding practical designs.

It is common to use idealizations of block ciphers as random permutations or functions from $(n+\delta)$-bits to n-bits in the analysis. In this case one can construct n-bit valued secure hash functions (see references above). In the case of DES, where $n = 64$, this yields 64-bit hash functions, which are vulnerable to simple birthday attacks. However, it is non-trivial to construct $2n$-bit valued secure hash functions from families of n-bit valued hash functions. The $2n$-bit valued hash function must behave like a $2n$-bit valued random function for up to 2^n queries, but the n-bit primitives run into birthday collisions around $2^{n/2}$ queries, which potentially could be used in an attack against the design. A solution for this output-doubling problem was given in [AV 96]. This construction is expensive, making eight calls to the underlying random function, and hence it is not suitable for a practical $2n$-bit valued compression function.

The analysis of our construction begins by assuming that both of the two functional components are random functions. This is *not* for the purpose of proving the existence of secure hash functions, but rather to examine what security parameters can be achieved. In addition, it motivates the weaker assumptions and the analysis that follow.

1.1 New Constructions

The constructions that we propose first stretch the input string mildly, and then compress the result of this expansion. Here we briefly motivate this approach.

Expansion stage: Our first stage stretches the input mildly. We will use primitives that have reasonable one-wayness and randomizing behavior, so as to obtain an almost surely one-to-one stretch function. This trivially avoids collisions in the first stage, and allows us to analyze this stage using distributional and one-way properties of the primitives we employ. Furthermore, these properties make it infeasible for the adversary to force its outputs into a set of his choice—for example, a set of points for which he has computed collisions for the second stage. We show how to use popular hash functions like MD5 or SHA-1 to do this. We remark that in large randomness tests with MD4 and MD5, it has been observed that both functions have very good distributional properties, even when they are iterated [PV 96].

Compression stage: In our second stage we apply a compression function. This stage could simply use any candidate collision-resistant hash function such as SHA-1 or RIPEMD-160. In fact, the security of our construction does not require collision resistance from the compression stage. For example, an adversary might find collisions for the compression stage. However, the colliding strings may not be in the range of the stretch function, and even those that are will be hard to invert. On the other hand, if the adversary begins by finding many input-output pairs for the stretch function, then a successful attack on the whole construction must find compression-stage collisions from among this restricted set of fairly random points.

Constructions using existing primitives

In a practical setting this work suggests ways to use the hash functions that are

currently broken or partially broken in such a way that we can depend on their one-wayness and randomness or distributional properties, rather than directly on their collision-security, which may be in doubt or already violated. In fact, there are many choices for each of the two components of our construction, and they can be combined independently.

Customized Hash Functions Customized hash functions, for example, MD5, SHA-1, and RIPEMD-160, can be used in either or both stages of our construction.

If the hash function has N bits of output, then it can be used in the stretching stage as follows. If at most N bits are needed for the input to the compression stage, then simply feed ℓ-bit blocks ($\ell < N$) of input text to the hash function. If more than N bits are needed for the input to the compression stage, we propose the following simple chaining. Use the N bits of output above as the first N bits of output of the chaining rule. In addition, concatenate these bits to the next ℓ bits of input to the hash function to get another N bits of output, and continue this chaining rule as needed.

For the compression stage, any of these hash functions can be used directly on the fixed-length output of the expansion stage.

We remark here that MD4 may also be sufficient for both stages. For example, as noted above the stretching stage is required to be one-way. Although two rounds of MD4 have recently been inverted [Dob 98], the inverse found is of length 512. Note that there are very many inverses ($2^{512-128}$) for an average 128-bit output. However, MD4 might be used in our stretching stage to expand, for example, 80-bit inputs to 128-bit outputs. In this case, for an overwhelming fraction of outputs, an adversary would be required to find the *unique* inverse. In addition, with a sufficiently random and one-way stretch function, our analysis suggests that requirements for the compression function are considerably relaxed. For example, using a truly random stretch function the compression function need only have a "fairly uniform" preimage structure.

Subset Sum Constructions based on the subset-sum function may be used in the stretching stage. The subset-sum function may also be used in the compression stage, because it is known to yield provably secure hash functions on the assumption that it is infeasible to find almost shortest vectors in lattices [GGH 96]. However, the best lattice-based attacks are quite powerful, forcing the lattices (and the cache needed for the implementation) to be relatively large. We suggest some constructions in the final version of this paper.

DES Any DES-based hash function, e.g., [MMO 85,BC$^+$ 88,Merk 89], may be used in either stage of our construction in the same way as described above for customized hash functions. However, since these hash functions consume few bits of input per DES call (i.e., they have low rate), the resulting hash function will be unacceptably slow for most practical applications.

In this paper we propose a new DES-based construction for the compression stage. Because of the properties of the first stage, our construction uses only two DES calls to obtain a 128-bit output value. The construction is extremely simple. As in [Merk 89] we will use a modified form of DES called MDES, defined

as follows: $\text{MDES}(K, x) = \text{DES}_K(x) \oplus x$ (where K is potentially the $16 * 48$-bit expanded key). The output of the stretching stage is split into two pieces, each of which is used separately as the key to one MDES call. The outputs of the two calls are simply concatenated.

Assuming that the stretch stage is a truly random function, and that DES has an almost regular preimage structure (i.e. all points in the range have approximately the same number of key-plaintext pairs mapping into them), we show that this construction is secure (see §3.3). This is a significant simplification on the requirements of the primitives to be used in a compression function. The same scheme without the randomizing initial stage is insecure; to achieve similar security would require more rigorously random-function like primitives, and many more calls to them (e.g. as in [AV 96]).

In many cipher-based constructions, the string to be hashed is used as a "key" to encrypt some initial or intermediate values of the hash function. The usual DES key scheduling algorithm stretches the given 56-bit key into 48*16 bits. One way to improve the rate of a DES-based hash function would be to skip the key-scheduling algorithm and feed 16*48 bits of input text directly as a key. This idea is swiftly rebuffed: one can use the invertibility of intermediate rounds of DES and mount a meet-in-the-middle attack, as follows (see [C 85] and [MH 81] for related attacks). The attacker can pick the text corresponding to the keys for all but three rounds arbitrarily. He picks the remaining round keys randomly, and expects a birthday collision between one round in the encrypting mode and the next level in the decrypting mode.

Our DES-based scheme, perhaps controversial, allows the thoroughly randomized output from the first stage to be used directly, and thus to skip the key scheduling algorithm. This considerably increases the rate of the resulting construction.

While this proposal clearly needs study, there is evidence to support the claim that keying DES in this manner is secure. We point out that this is similar to scheduling the round keys in DES with independent keys, a method whose security is closer to exhaustive search for the 56-bit key (rather than the extended key of length 16*48) in the sense that it takes about 2^{64} steps (including computational overhead) by current differential attacks, and one may expect this number to be somewhat smaller for linear attacks. Of course, the key is considered hidden for the differential attacks against a block cipher. Differential attacks are far more natural in the context of secure hash functions since the attacker can compute all the required input-output pairs by himself. In addition, the attacker could conceivably mount a meet-in-the-middle attack based on the invertibility of individual rounds of DES, as discussed above; but such attacks are not applicable to our use of DES, because the adversary has little effective control over the key bits.

Example parameters are as follows: we can securely stretch a 512-bit string to a 16*48-bit string and use the latter output as a DES key. Then using the stretched output as a DES key would effectively allow us to compress 512 bits per DES call. With well optimized assembly-language implementations, this results in

implementations that are so much faster than standard DES-based constructions that they can be used in practice. See below for run-times of a preliminary implementation.

We stress that the second stage is not *required* to be derived from a cipher, and hence our first stage is not merely a key scheduling algorithm. Often, as in the case of DES or Tiger, the key scheduling in a hash-function design is reversible, but we demand one-way and randomizing properties in our first-stage functional component. Such reversibility may be more appropriate for ciphers, where the key is held secret, than it is for hash functions, where the collision adversary can choose the inputs. Our stretch functions may actually strengthen block-cipher constructions by helping to avoid weak keys and related-key attacks; we omit details here to obey space constraints.

However, there are attacks on adaptations of DES that skip the key-scheduler. Below we point out how the use of stretch functions can avoid these attacks. Briefly, the attacker must be able to choose some portions of the extended key during the attack, which is what the randomizing expansion step is designed to preclude: With overwhelming probability, the attacker's choice of extended keys will not be in the range of the first stage, and there is no easy way to take a small string and extend it to a string lying in the range of the first stage.

We believe our designs are useful in practice, and allow their security to be analyzed under explicitly stated assumptions on the cryptographic primitives that we use. Finding the weakest assumptions sufficient for the construction of collision-resistant hash functions is a fundamental unsolved problem. Our constructions raise some related issues that may be helpful both for the practical as well as the theoretical point of view. To summarize, the stretching stage simplifies the requirements on the compression function, which is arguably the crux of the task of designing secure functions. This is significant in itself, and may actually lead to faster constructions upon further research.

Performance

We performed a preliminary implementation to test the speed of one version of our construction and found it surprisingly fast compared to several other hash functions. Our test implementation on a 166Mhz Pentium Processor based laptop computer yielded a version running around 60Mbits/second. Here, as described above, we used MD5 for the stretch function, mapping 96*6 bits to 128*6=48*16 bits, and for the compression stage we used the DES-based construction producing 128-bit hash values.

We compared the speed of our construction both to MD5 and to DES-based hash functions. We tested with many variants for the DES-based compression schemes. The speed reported here for these functions is overestimated by assuming that they consume close to 56 bits of input per DES call. The speeds of MD5, our hash construction, and DES-based hashing are in the ratio 1 : 0.43 : 0.032. Our testing did not optimize for platform-dependent parameters such as cache size. The usually quoted speed ratios between MD5, SHA-1, RIPEMD-160, DES are 1 : 0.41 : 0.34 : 0.13. These ratios are at best treated as ap-

proximations, since many parameters can cause these ratios to vary among modern processors. For example, assembly coding can speed up different algorithms at different rates. While our DES code was optimized, our MD5 implementation was a straightforward one. In the final version, which will be available from the authors (or at `http://research.microsoft.com/crypto` and `http://www.surety.com/pub/`), we shall present more detailed performance analysis of more varied schemes.

1.2 Imperfect Random-Function Model Constructions

In §4 below, we suggest some imperfect random-oracle models, and show how to build better primitives from given imperfect ones. In this vein, we also analyze how to defend against a collision-finding adversary for a given primitive by building "independent" primitives.

2 Preliminaries

We will often model functions as random functions. A random function has the following property. When it is evaluated on an input (assumed to be different from all other inputs thus evaluated, since there is no need to evaluate the function more than once on the same input) the output is uniformly distributed and independent of all output values thus far.

We fix a bounding function $B(n)$ (e.g. $2^{0.5n}$) and this will correspond to our notion of an infeasible amount of resources (e.g. run-time or memory). We call functions (e.g. run-times) lower-bounded by $B(n)$ *infeasible* and functions that are smaller than $1/B(n)$ *negligible*. We call probabilities of the form $1 - 1/B(n)$ *overwhelming*.

A function f mapping n bits to m bits is said to be *one-way* if f is efficiently computable (e.g. in polynomial time), and given $y = f(x)$ where x is randomly chosen, any inverting algorithm $I(\cdot)$ with $f(I(y)) = y$ takes at least time $B(n)$ with overwhelming probability (over x). In addition, if $m < n$ and for any (adversary's collision-finding) algorithm C, a successful execution $C(n, m) = (x, x')$ satisfying $f(x) = f(x')$ takes time at least $B(m)$, then we call this function *collision-resistant*. For formal definitions and implementations based on various assumptions see [Dam 87,Merk 89,BY 90]; in addition, [Pre 93,MOV 97] are excellent references for this topic. It is not known what is the weakest assumption under which one can construct collision-resistant hash functions.

Given a fixed-length collision-resistant *compression function* H mapping L-bit inputs to N-bit outputs ($N < L$), one can build a collision-resistant hash function G defined on arbitrary-length inputs following the construction of Merkle [Merk 90] and Damgard [Dam 89]. Assign a fixed N-bit "initial-value" string IV, and given an input $x = x_1 x_2 \cdots x_t$ (formatted, with "Merkle-Damgard strengthening," i.e. with appropriate padding to encode the length of the text, as t blocks of length $L - N$), let the value of $G(x)$ be defined as follows: $G_0 = IV$; $G_i = g(G_{i-1}, x_i)$, $1 \leq i \leq t$; $G(x) = G_t$. Thus, we will concentrate here on analyzing a fixed-length, collision-resistant compression function H.

3 New Constructions: Definition and Analysis

After describing our new construction (in §3.1), we proceed to analyze its security, first by assuming that its components are truly random functions of the appropriate class (§3.2), and then by weakening these assumptions (§3.3). This analysis treats the properties of our construction of a cryptographic hash function H with fixed-length inputs.

3.1 Definition

Our constructions first stretch the inputs and apply a compression function next. We describe the requirements on these functions after presenting some rationale for stretching.

Secure Stretch Functions We introduce the use of *secure stretch functions*, which mildly increase the input lengths, for the purposes of constructing hash functions. A *stretch function* f maps ℓ-bit inputs into $2m$ bit inputs, where $2m > \ell$. The input strings to f will be denoted by t and the output strings will be denoted by the pair $K, \bar{K}$. Informally, they satisfy:

- One-wayness: given any $y = f(x)$ it is hard to find any x' such that $f(x') = y$.
- Outputs of f behave as if f is locally random (i.e., k-wise independent, for some $k >> 1$).

Under the randomizing conditions we pose on f's outputs, f is an injective function on an overwhelming fraction of the inputs if $2m - \ell$ is large enough. Our definition of one-wayness is also known as "preimage resistance."

Compression Functions The outputs of these stretch functions (along with a $2n$-bit IV) are fed into a compression function $\mathbf{h}$ from $(2m + 2n)$ bits to $2n$ bits. We will consider the first $2m$ bits of input as a key. The remaining $2n$ bits of input will be denoted by the pair $x, \bar{x}$ and the output by $y, \bar{y}$. Our overall compression function will be denoted by $\mathbf{h}$, which compresses ℓ-bit strings to $2n$-bit strings. It is defined as follows:

$$f(t) = K, \bar{K}, \quad \mathbf{h}_{f(t)}(x, \bar{x}).$$

While there are many instantiations for $\mathbf{h}$ the one we will concentrate on is as follows. Let h denote a compression function from $m + n$ bits down to n bits. The first m bits of input of h will be considered a key. For now, we will define $\mathbf{h}_{K,\bar{K}}(x, \bar{x}) = h_K(x), h_{\bar{K}}(\bar{x})$.

In our implementations we use a modified form of DES as our function $h(\cdot)$, namely $h_K(x) = \text{MDES}(K, x) = \text{DES}_K(x) \oplus r(K, x)$, where $r(K, x)$ represents some simple function of K and x. (For example, $r(K, x) = x$ was suggested in [MMO 85,Merk 89].)

Note that in this case the hash value is 128 bits and to resist the attacks due to van Oorschot and Wiener [vOW 94] 192-bit hash values may be needed. It is easy to generalize our result to 192 bits by using three calls to the underlying cipher. This will be covered in the complete version of the paper.

Butterfly Compression: We define a variation on the compression function above, the "butterfly compression," as follows:

$$\mathbf{bh}_{K,\bar{K}}(x,\bar{x}) = h_K(x) \oplus r(\bar{K},\bar{x}), h_{\bar{K}}(\bar{x}) \oplus r(K,x),$$

where h and r are appropriately chosen and $r(\cdot)$ is very simple to compute (for example, $r(k,y) = y$). This variation appears to increase the complexity of the attacks using inversion algorithms. In the final version of this paper we present an analysis of this scheme.

3.2 Analysis Assuming f and h Are Random Functions

We begin our analysis of H by assuming that f and h are random functions.

Lemma 1. *When the stretching function f is a random function from ℓ bits to $2m$ bits and h is a random function from $m+n$ bits to n bits, an adversary making a total of q queries to f and h will find a collision with probability $\Theta(q^2/2^{2n})$.*

Proof: Any adversary which makes a total of q queries in sum to f and h can do no better than an adversary which makes q queries to both f and $\mathbf{h}$. We will thus analyze the latter type of adversary.

Due to the fact that both f and h are random functions, it is easy to show that the adversary maximizes its chances of finding a collision by using the outputs of its queries to f as the input to its queries to $\mathbf{h}$. Due to space limitations we omit this argument here. So, assume the adversary makes q queries to f to produce $\{(k_i,\bar{k}_i)\}$ as well as $\{(y_i,\bar{y}_i)\}$, $1 \le i \le q$, where $y_i = h_{k_i}(x)$ and $\bar{y}_i = h_{\bar{k}_i}(\bar{x})$.

We will assume that $m \ge n$. Fix a pair of queries, i and j, $1 \le i, j \le q$, and let us calculate the probability that this pair of queries yields a collision, i.e., that $(y_i,\bar{y}_i) = (y_j,\bar{y}_j)$. There are four disjoint cases.

Case 1 : $k_i = k_j$ and $\bar{k}_i = \bar{k}_j$. This event occurs with probability 2^{-2m}.
Case 2 : $k_i \neq k_j$ and $y_i = y_j$ and $\bar{k}_i = \bar{k}_j$. This event occurs with probability at most $2^{-(n+m)}$.
Case 3 : $k_i = k_j$ and $\bar{k}_i \neq \bar{k}_j$ and $\bar{y}_i = \bar{y}_j$. This event also occurs with probability at most $2^{-(m+n)}$.
Case 4 : $k_i \neq k_j$ and $\bar{k}_i \neq \bar{k}_j$ and $(y_i,\bar{y}_i) = (y_j,\bar{y}_j)$. This event occurs with probability at most 2^{-2n}.

Since we are assuming that $m \ge n$, the probability that there is a collision for this fixed pair of queries is at most $4/2^{2n}$. Hence, the probability that there is any collision is $4\binom{q}{2}/2^{2n}$.

This analysis is tight. Clearly, the probability that a collision occurs in q queries is at least $\Omega(q^2/2^{2n})$. □

3.3 Analysis with Weaker Assumptions

We now assume that one of the two functions behaves like a random function and ask what conditions must be required of the other function. It turns out that rather surprisingly weaker conditions suffice.

Assuming f is random and h is almost regular

We now assume f behaves like a random function. Our goal is to show that it is sufficient to have some assumption on the distribution of the number of inverses belonging to a point in the range of h. Define $S_x(y) = \{K \mid h_K(x) = y\}$. Let $s_x(y) = |S_x(y)|$. For any fixed x, note that $\sum_y s_x(y) = 2^m$, so that the average value of $s_x(y)$ over all the values of y is 2^{m-n}. Define $\rho_x(y) = s_x(y)/2^{m-n}$ and observe that $\sum_y \rho_x(y) = 2^n$.

Definition 2. *We say that h is α-regular if for all x, $\sum_y \rho_x(y)^2 \leq \alpha 2^n$.*

This condition is equivalent to the following: Let y be a random point in the range, chosen with probability 2^{-n}. Then, for a randomly chosen y, $\mathbf{E}(\rho_x(y)^2) < \alpha$.

For a random function, the value of α is a constant with high probability. Note that requiring a function to be α-regular is a weaker condition than requiring that each $\rho_x(y)$ be less than or equal to $\sqrt{\alpha}$ (for example, a $(1 + o(1))$-regular function might have, say, n values of y with $\rho_x(y) = n$ for some value of x).

Theorem 3. *If the function h is α-regular and f is a random function, then an adversary making q queries to f will find a collision of H with probability $\Theta(\alpha^2 q^2/2^{2n})$.*

Proof: Assume the adversary makes q queries. Fix i and j and consider the i-th and j-th queries. Then,

$$
\begin{aligned}
&Prob[(y_i, \bar{y}_i) = (y_j, \bar{y}_j)] \\
&= \sum_y \sum_{\bar{y}} Prob[y_i = y_j = y; \quad \bar{y}_i = \bar{y}_j = \bar{y}] \\
&= \left(\sum_y Prob[k_i \in S_x(y); k_j \in S_x(y)] \right) \left(\sum_{\bar{y}} Prob[\bar{k}_i \in S_{\bar{x}}(\bar{y}); \bar{k}_j \in S_{\bar{x}}(\bar{y})] \right) \\
&= \left(\sum_y s_x^2(y)/2^{2m} \right) \left(\sum_{\bar{y}} {s_{\bar{x}}}^2(\bar{y})/2^{2m} \right) \\
&= \left(\sum_y \rho_x^2(y)/2^{2n} \right) \left(\sum_{\bar{y}} \rho_{\bar{x}}^2(\bar{y})/2^{2n} \right) \leq \alpha^2/2^{2n}
\end{aligned}
$$

where the last inequality follows from the regularity assumption. It follows that the probability of the adversary finding a collision is at most $\binom{q}{2}\alpha^2/2^{2n}$. □

So it follows that randomness properties of f are sufficient to weaken the requirements on h considerably. Also, note that the outputs of f need not be completely independent, only 4-wise independent.

A word of caution is warranted here. If no one-wayness properties on h are imposed other than the degeneracy condition, then one must be careful about the fraction of easy points in the range of f (at which it is easy to invert f) in concrete implementations of f.

Assuming f is almost injective and h is random

In this section we consider the dual of the previous section: h will be considered a random function and we will impose some computational assumptions on f. We will first assume that f has high collision security as defined below. We stress here that the functions we consider here are not necessarily compressing.

Definition 4 (Collision Security of Hash functions). *A family of hash functions onto n bits has Collision security of $s(n)$ (example $s(n) = 2^n$) if any algorithm running in time $T(n)$ finds a collision pair with probability at most $(T(n)/s(n))^2$.*

Note that a more general version of the definition would allow the 2 to be replaced by any positive constant.

We stress that when the function is stretching and "randomizing", a large collision security is a mild and realistic assumption since the function is likely to be injective on overwhelming fraction of the range.

The collision security will help us characterize the strength of our scheme. But we will also need our stretch function f to be resistant to partial collisions. Recall that f maps l bits to $2m$ bits. Define a partial collision of size i to be when i inputs yield outputs which all have the same first m bits or which all have the same second m bits. Actually, we will need to account for all the partial collisions. Define $B_K = \{\bar{K}_i \mid (K_i, \bar{K}_i)$ is a response and $K_i = K\}$. Note that $\sum_K |B_K|$ is equal to the total number of queries to f. Define $b_K = |B_K|$ when $|B_K| \geq 2$ and 0 otherwise. $b_{\bar{K}}$ is defined similarly. Let $B^2 = \sum_K b_K^2 + \sum_{\bar{K}} b_{\bar{K}}^2$.

Definition 5 (Partial Collision Security of Hash functions). *A family of hash functions onto $2m$ bits has Partial Collision Security of $\bar{s}(m)$ if for any algorithm running in time $T(2m)$ the probability of B^2 exceeding $T(2m)$ is at most $T(2m)/\bar{s}(m)$.*

To give an example of this definition let us apply it to a random function, where $T(2m)$ just becomes the number of queries q. In such a case the partial collision security is $2^m/4$. To see this, it can first be shown that for a random function the expected value of $\sum_K b_K^2$ is at most $2^m \sum_{i \geq 2} (q/2^m)^i \leq 2 \cdot q^2/2^m$, where the last inequality follows whenever $q \leq 2^m/2$. The same holds for $\sum_{\bar{K}} b_{\bar{K}}^2$. Hence, the probability that B^2 exceeds q is at most $4 \cdot q/2^m$ which yields a partial collision security of $2^m/4$.

The following theorem shows that if the running time of an adversary is $o(2^n)$, $o(s(2m))$, and $o(\bar{s}(m))$ then it can only find a collision with probability $o(1)$.

Theorem 6. *If f has collision security $s(2m)$ and partial collision security $\bar{s}(m)$, and h is a random function then any algorithm running in time $T(2m)$ will find collisions with probability at most*

$$(T(2m)/s(2m))^2 + T(2m)/\bar{s}(m) + T(2m)/2^n + (T(2m)/2^n)^2$$

Proof: Suppose an adversary running in time T finds collisions on H. There are three cases.

Case 1: The adversary finds a collision on f. By assumption this happens with probability at most $(T(2m)/s(2m))^2$.

Case 2: The adversary finds partial collisions of f. For each K with $b_K > 1$, the adversary will get a collision on y and have a probability of a collision on $\bar{y}$ at most $2^{-n}b_K^2$. An analogous statement holds for each $\bar{K}$ with $b_{\bar{K}} > 1$. So, in this case the probability of a collision on the output is $2^{-n}B^2$. By assumption B^2 exceeds T with probability at most $T/\bar{s}$. Hence, this case occurs with probability at most $T/\bar{s} + T/2^n$.

Case 3: The adversary finds no collisions or partial collisions on f. Since h is random function, the probability of a collision is $q^2/2^{2n}$, where q is the number of input output pairs of f computed by the adversary. Since q is always bounded from above by T, this yields an upper bound on the probability for this case of $(T/2^n)^2$. □

4 Constructions Based on Imperfect Random-Function Models

Here we suggest simple constructions and heuristics to construct new hash functions using the old ones, so that the new one may be hard to break even if one or more of the old ones come under attack.

4.1 Composition Construction

Let f and g be two functions from finite binary strings to n bits. Then define

$$H(x) = [g(x), f(g(x), x)].$$

By *inversion security* of a one-way function f, we mean a lower bound $s_f(L)$ on the time to find an L-bit inverse of the function on all but a negligible fraction of the instances $y = f(x)$, where x is a randomly chosen L-bit string. The *collision security* of a hash function f is a lower bound $s'_f(n)$ on the time required to find two inputs x_1 and x_2 such that $f(x_1) = f(x_2)$. We first claim that the above construction at least preserves security. Note the symmetry: we need not know which of the functions is more secure, either with respect to collisions or with respect to inversion.

Lemma 7. *If f and g are collision-resistant hash functions, then H is collision-resistant. Moreover its inversion security and collision security are not lower than that of either f or g. I.e.,*

$$s'_H(n) \geq Max(s'_f(n), s'_g(n)), \qquad s_H(n) \geq Max(s_f(n), s_g(n))$$

Proof: Indeed, if $H(x_1) = H(x_2)$ is a collision for H, then we get a g-collision at $y = g(x_1) = g(x_2)$ and an f-collision at $z = f(y, x_1) = f(y, x_2)$. Similarly for the inversion security. □

It seems to be essential to use x twice in this construction. The inversion security of H is never more than twice the maximum of the two. In the random-function model the composition of two functions usually causes more collisions, which makes it easy to distinguish a composition from a truly random function; however, here we are interested in the difficulty of finding collisions. Since the number of rounds in H is the sum of the rounds in f and g, one would heuristically expect the resultant function to be stronger.

Now we would like to obtain f and g that behave as if they were "independent." If the functions behaved almost like "random functions" then of course the construction would be secure. We want to provide a formal background for analyzing this. For this we use the model of *imperfect random functions* for the primitives. Any existing primitive with an estimated security (at least currently) can be thought of as an imperfect random function with appropriate parameters. For this we define two measures. First we consider a simpler (but more restrictive) bit-level parameter. We define the *bias* of a Boolean-valued random variable β to be $\Pr[\beta = 1]$.

Definition 8. *A function is called p-random if its output bits have bias p and are independent* ($0 \le p \le 1$).

Needless to say, considering the individual bits to be independent is less realistic, but it gives us a more reasonable heuristic than the perfect random function model. These functions are easily distinguished from truly random functions (for $p \neq \frac{1}{2}$) merely by observing the fraction of 1's in the output strings. Considering the output as a whole we make the following definition.

Definition 9. *A function (with n-bit values) is called h-imperfect if its output has min-entropy equal to hn.*

Here, the *min-entropy* of a source that outputs strings $x_1, \ldots, x_N$ with respective probabilities $p_1, \ldots, p_N$ is $\min\{-\lg p_i\}$. Note that the individual bits of the values of an h-imperfect function can be correlated, and they may not look at all random. However, when it comes to collision security, h-imperfect functions are good enough: High min-entropy is a necessary condition for a secure hash function; for example, if the min-entropy is low, then much of the probability may be concentrated in a small set, and it would to be easy to find a collision. But the condition is also sufficient, as shown by Lemma 11 below.

Obviously, a p-random function is $H(p)$-imperfect.

Lemma 10. *A collision adversary making a total of q queries to a p-random function will find a collision with probability at most* $q^2/2^{\lambda H(p)/2}$, *where* $p = 1/2 + \epsilon$ *and* $\lambda = 1 - 2\epsilon^2 \lg_2 e$.

Proof: Omitted due to space constraints.

Lemma 11. *A collision adversary making a total of q queries to an h-imperfect function (with n-bit values) will find a collision with probability at most $q^2/2^{hn/2}$.*

Proof: Let f be an h-imperfect function. If a string y occurs as an output of f with probability $p > 0$, then we have $-\lg p \geq hn$, or $p \leq 2^{-hn}$. And now consider a collision adversary that makes q queries to f, which we model as the random variables $Y_1, \ldots, Y_q$, taking respective values $y_1, \ldots, y_q$ in a particular run. The expected number of colliding pairs is

$$\sum_{i,j} \Pr[Y_i = Y_j] \;=\; \sum_{i,j} \Pr[Y_j = y_i] \;\leq\; \sum_{i,j} 2^{-hn} \;=\; \binom{q}{2} 2^{-hn} \;\leq\; q^2 2^{-hn}.$$

If we take $q = 2^{hn/2}$, then the expected number of collisions is at most $q^2 2^{-hn} = 1$. Thus the collision security of f is at least $2^{hn/2}$, as claimed. □

4.2 Construction of "Independent" Primitives

One way to view the above construction is that it takes two imperfectly random functions and yields a function that is closer to being a truly random function. Towards this end, consider the following construction:

$$\bar{H}(x) = f(x) \oplus g(x)$$

It is relatively easy to analyze the construction at a bit level in the random-function model.

Lemma 12. *If f and g are p-random and independent, then the function $f \oplus g$ is q-random, with $|1 - 2q| = (1 - 2p)^2$.*

Proof: If α and β are two independent Boolean variables, both with bias p, then $\alpha \oplus \beta$ has bias $q = 2p(1-p)$, and q satisfies $|1 - 2q| = (1 - 2p)^2$. □

A Boolean variable with bias p has a gap between the probabilities of occurrence of 1 and 0 of $|p - (1-p)| = |1 - 2p|$. The significance of this lemma is that the gap narrows quadratically as we pass from f or g to $f \oplus g$. Thus, for any p, after k iterations of this process with independent functions narrows the gap to $|1 - 2p|^{2^k}$.

However, this construction of $\bar{H}$ does not allow us to make a claim as in the above lemma, when we move from the idealized random-function world to a complexity-theoretic world where the functions involved are specific presumably collision-secure functions. That is, you can not show that $\bar{H}$ is collision-secure if f or g is. However in an imperfect random function model it is easy to show the following.

Given two imperfect random functions f and g, the expected number of queries to find collisions for H and $\bar{H}$ is the same in both cases. Thus the construction for H is "better" in that it allows us to analyze its security both in the complexity and in the random-function worlds. But we now show that the $\bar{H}$ construction has the surprising result of creating independence.

Intuitively, we say that f is independent of g if the provision of an oracle for finding collisions for f does not help in finding collisions for g. Note the asymmetry in the definition; for technical reasons, we must allow for the possibility that a collision-finder for g may help to find collisions for f.

Definition 13. *A collision oracle for f is a function from i to the set of finite binary strings. On input (i, k) it outputs a set $S_i = \{x_1, \ldots, x_j\} \subseteq \{0,1\}^L$ $(2 \leq j \leq k)$ such that f has the same value on all points from the set. The set are distinct: for $i \neq j$, the sets S_i and S_j are unequal. The charge for a sequence of queries is the sum of $|S_i|$.*

We say that f is T-independent of g if, given a collision oracle for g, one incurs a charge of at least T for the calls to the oracle to find a collision for f. We say that f is independent of g if g is h-imperfect and f is T-independent of g with $T = \Omega(2^{nh/2})$

Now we claim the following. Let f and g be a-random and b-random respectively. Then $\bar{H}$ is T-independent of f, where T is the collision security of g. Similar comments apply for independence with respect to f. To see this, assume we are given an oracle for g to find collisions. $\bar{H}$ collides on the outputs of the oracle if and only if f collides on them as well. We can give similar constructions and results for imperfect random functions.

Thus in the imperfect oracle model, finding collisions for one of the functions does not help in finding collisions for the composite function at all, when the collision finder is used as a black box. Using this as a heuristic, if one takes f, g as SHA-1, MD5, and both are broken, then $\bar{H}$ would still need many calls to the collision finders, if the outputs of these functions behave approximately randomly even to a mild degree. This is helpful even if one of the hash functions is weak, for example, breakable in 2^{35} steps. In this case, finding collisions for the combination may still be infeasible.

For more detailed discussion and schemes based on these considerations, see the full version of this paper.

Acknowledgments: We thank Bart Preneel for helpful discussions. The third author thanks Yacov Yacobi, whose questions regarding a smart-card application provided the initial interest in this problem.

References

AV 96. W. Aiello and R. Venkatesan. Foiling birthday attacks in length-doubling transformations. In *Advances in Cryptology—Eurocrypt '96*, Lecture Notes in Computer Science, Vol. 1070, ed. U.M. Maurer, pp. 307–320 (Springer-Verlag, 1996).

AB 96. R. Anderson and E. Biham. Tiger: A Fast New Hash Function, In *Fast Software Encryption 3*, Lecture Notes in Computer Science, Vol. 1039 (Springer-Verlag, 1996).

BGV 96. A. Bosselaers, R. Govaerts, J. Vandewalle. Fast hashing on the Pentium. In *Advances in Cryptology—Crypto '96*, ed. N. Koblitz, Lecture Notes in Computer Science, Vol. 1109, pp. 298–312 (Springer-Verlag, 1996).

BP 95. A. Bosselaers and B. Preneel (eds.). *Integrity Primitives for secure information systems: Final report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040)*, Chapter 3: RIPEMD. Lecture Notes in Computer Science, Vol. 1007 (Springer-Verlag, 1995).

BC$^+$ 88. B. O. Brachtl, D. Coppersmith, M. M. Hyden, S. M. Matyas, Jr., C. H. W. Meyer, J. Oseas, Sh. Pilpel, and M. Shilling. Data authentication using modification detection codes based on a public one way encryption function. U.S. Patent No. 4,908,861, issued March 13, 1990. (Described in: C. H. Meyer and M. Shilling, Secure program load with modification detection code, In *Securicom 88: 6ème Congrès mondial de la protection et de la sécurité informatique et des communications*, pp. 111–130 (Paris, 1988).)

BY 90. G. Brassard and M. Yung. One-way group actions. In *Advances in Cryptology—Crypto '90*, Lecture Notes in Computer Science, Vol. 537, pp. 94-107, (Springer-Verlag, 1991).

C 85. D. Coppersmith. Another birthday attack. In *Advances in Cryptology—Crypto '85*, Lecture Notes in Computer Science, Vol. 218, pp. 14–17, (Springer-Verlag, 1986).

Dam 87. I. Damgard. Collision-free hash functions and public-key signature schemes. In *Advances in Cryptology—Eurocrypt '87*, Lecture Notes in Computer Science, Vol. 304, pp. 203–217, Springer-Verlag (1988).

Dam 89. I. Damgard. A design principle for hash functions. In *Advances in Cryptology—Crypto '89*, Lecture Notes in Computer Science, Vol. 435, pp. 416–427, Springer-Verlag (1988).

Dob 96a. H. Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption*, Lecture Notes in Computer Science, Vol. 1039, ed. D. Gollman, pp. 53–69, Springer-Verlag (1996).

Dob 96b. H. Dobbertin. Cryptanalysis of MD5 compress. Rump Session of Eurocrypt '96, presented by B. Preneel (May 1996). (Available at `http://www.iacr.org/conferences/ec96/rump/`.)

Dob 96c. H. Dobbertin. The status of MD5 after a recent attack. *CrytoBytes*, Vol. 2, No. 2 (Summer 1996). (Available at `http://www.rsa.com/rsalabs/-pubs/cryptobytes/`.)

Dob 97. H. Dobbertin. RIPEMD with two-round compress function is not collision-free. *Journal of Cryptology*, Vol. 10, No. 1, pp. 51–69 (1997).

Dob 98. H. Dobbertin. The first two rounds of MD4 are not one-way. In *Fast Software Encryption*, Lecture Notes in Computer Science, Springer-Verlag (to appear).

DBP 96. H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. In *Fast Software Encryption*, Lecture Notes in Computer Science, Vol. 1039, pp. 71–82, Springer-Verlag (1996).

GGH 96. O. Goldreich, S. Goldwasser, and S. Halevi. Collision-free hashing from lattice problems. Theory of Cryptography Library, Record 96-09. (Available at `http://theory.lcs.mit.edu/~tcryptol/`.)

KP 97. L. Knudsen, B. Preneel. Fast and secure hashing based on codes. In *Advances in Cryptology—Crypto '97*, Lecture Notes in Computer Science, Vol. 1294, pp. 485–498, Springer-Verlag (1997).

MMO 85. S.M. Matyas, C.H. Meyer, and J. Oseas. Generating strong one-way functions with cryptographic algorithm. *IBM Technical Disclosure Bulletin*, vol. 27, pp. 5658–5659 (1985).

MOV 97. A. Menezes, P. van Oorschot, S. Vanstone. *Handbook of Applied Cryptography* (CRC Press, 1997).

Merk 80. R.C. Merkle. Protocols for public key cryptosystems. In *Proc. 1980 Symposium on Security and Privacy*, IEEE Computer Society, pp. 122–133 (April 1980).

Merk 89. R.C. Merkle. One-way hash functions and DES. In *Advances in Cryptology—Crypto '89*, Lecture Notes in Computer Science, Vol. 435, pp. 428–446 (Springer-Verlag, 1990).

Merk 90. R.C. Merkle. A fast software one-way hash function. *Journal of Cryptology*, Vol. 3, pp. 43–58 (1990).

MH 81. R.C. Merkle and M. Hellman. On the security of multiple encryption. *Communications of the ACM*, Vol. 24, No. 7, pp. 465–467 (July 1981).

MOI 90. S. Miyaguchi, K. Ohta, and M. Iwata. 128-bit hash function (N-hash). *NTT Review*, vol. 2, pp. 128–132 (1990).

NY 89. M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Symposium on Theory of Computing*, pp. 33–43 (ACM, 1989).

NIST 94. National Institute of Standards and Technology. Secure Hash Standard. NIST Federal Information Processing Standard Publication 180-1 (May 1994).

PV 96. M. Peinado, R. Venkatesan. Highly parallel cryptographic attacks. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM-MPI'97)*, Lecture Notes in Computer Science (Springer-Verlag, 1997).

Pre 93. B. Preneel. *Analysis and Design of Cryptographic Hash Functions*. Ph.D. dissertation, Katholieke Universiteit Leuven (January 1993).

Pre 97. B. Preneel, private communication (1997).

PGV 93a. B. Preneel, R. Govaerts, J. Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *Advances in Cryptology—Crypto '93*, Lecture Notes in Computer Science, Vol. 773, pp. 368–378 (Springer-Verlag, 1991).

PGV 93b. B. Preneel, R. Govaerts, J. Vandewalle. Differential cryptanalysis of hash functions based on block ciphers. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pp. 183–188 (ACM, 1993).

R 78. M.O. Rabin. Digitalized signatures. In *Foundations of Secure Computation*, eds. R. Lipton, R. DeMillo, pp. 155–166 (Academic Press, 1978).

RP 95. V. Rijmen, B. Preneel. Improved characteristics for differential cryptanalysis of hash functions based on block ciphers. In *Fast Software Encryption*, Lecture Notes in Computer Science, Vol. 1008, pp. 242–248 (Springer-Verlag, 1995).

Riv 90. R. Rivest. The MD4 message digest algorithm. In *Advances in Cryptology—Crypto '90*, Lecture Notes in Computer Science, Vol. 537, pp. 303–311, (Springer-Verlag, 1991).

Riv 92. R. Rivest. The MD5 Message-Digest Algorithm. Internet Network Working Group Request for Comments 1321 (April 1992).

Sur 95. Surety Technologies, Inc. Answers to Frequently Asked Questions about the Digital Notary™ System. `http://www.surety.com` (since January 1995).

vOW 94. P. van Oorschot and M. Wiener. Parallel collision search with applications to hash functions and discrete logarithms. In *Proceedings of the 2nd ACM Conference on Computer and Communication Security*, pp. 210–218 (ACM Press, 1994).