# A FOUR-COMPONENT FRAMEWORK FOR DESIGNING AND ANALYZING CRYPTOGRAPHIC HASH ALGORITHMS

George I. Davida, Jeremy A. Hansen
*Center for Cryptography, Computer and Network Security*
*University of Wisconsin - Milwaukee, 3200 N. Cramer Street, Milwaukee, WI, 53211 USA*
*Email: davida@uwm.edu, jahansen@uwm.edu*

Keywords: collisions, compression functions, hash algorithms, HAVAL, MD5, Merkle-Damgård, pre-processing, RadioGatún, SHA-1, Tiger, WHIRLPOOL

Abstract: Cryptographic hash algorithms are important building blocks in cryptographic protocols, providing authentication and assurance of integrity. While many different hash algorithms are available including MD5, Tiger, and HAVAL, it is difficult to compare them since they do not necessarily use the same techniques to achieve their security goals. This work informally describes a framework in four parts which allows different hash algorithms to be compared based on their strengths and weaknesses. By breaking down cryptographic hash algorithms into their preprocessing, postprocessing, compression function, and internal structure components, weaknesses in existing algorithms can be mitigated and new algorithms can take advantage of strong individual components.

## 1 INTRODUCTION

Research in cryptographic hash algorithms has gone through many life-cycles while algorithms have been analyzed, created, and broken. Although techniques for block cipher construction were quickly adapted to hash algorithm design and several block-cipher-based hash algorithm structures have been developed, it is not clear that these constructions achieve the practical security goals of cryptographic hash algorithms. Even today, there is no consensus on a suitable definition of what makes a hash algorithm secure. Instead, cryptographers rely on theoretical foundations such as the random oracle model which may not be practically attainable [28] and ad hoc constructions which may or may not improve the security of these algorithms. At the Second NIST Hash Workshop in 2006, it was agreed that cryptographers do not really know what they want in a hash algorithm [33]. This paper is an attempt to partially rectify this lack of direction.

As weaknesses continue to be found in common hash algorithms, a significant step forward would be the concrete determination of what makes hash algorithms secure, or in a different light, what makes them weak. The compression function has long been considered the most important part of a hash algorithm, and while its importance should not be understated, the other parts of calculating a cryptographic hash also prove to be vital. For example, weaknesses leading to collisions have been demonstrated in the compression function of the MD5 algorithm, and subsequent suggestions for modifying the algorithm have been offered. Some of these suggestions, such as those describing preprocessing steps to be taken before any part of the message is delivered to the compression function, do not change the underlying weakness in the compressor. While these modifications certainly defeat the specific attack for which they were designed, MD5 would be better if it had "defense in depth": a strong compression function in addition to strong preprocessing techniques. This paper describes a model such that cryptographic hash algorithms can be modularly evaluated for security properties by comparing their individual components: preprocessing, postprocessing, the compression function, and the internal structure.

## 2 THE FRAMEWORK

At its most basic, this framework simply divides the operations of cryptographic hash algorithms into four distinct parts: a preprocessing step, the internal structure, the compression function(s), and a postprocessing step. An $n$-bit hash algorithm $H$: $\{0,1\}^* \rightarrow \{0,1\}^n$ that processes a message $M$ of arbitrary length made up of $L$ fixed-size blocks $M_0, M_1, M_2, ..., M_{L-1} \in \{0,1\}^b$ can also be defined in terms of four functions $P$, $Q$, $R$, and $S$, such that $H(M) = P(Q_R(S(M)))$. In this equation, $S$: $\{0,1\}^* \rightarrow \{0,1\}^*$ indicates a preprocessing step which may operate on the entire message $M$ or on each block $M_i$ of the message - shortening, lengthening, or doing otherwise according to the requirements of the internal structure. $Q_R$: $\{0,1\}^* \rightarrow \{0,1\}^s$ indicates the internal (iterative) structure of the hash algorithm which uses the compression function $R$: $\{0,1\}^b \rightarrow \{0,1\}^c$ ($b > c$) in its iterated operation on an arbitrary-length input $M$ to generate a semi-final value of length $s$. $P$: $\{0,1\}^s \rightarrow \{0,1\}^n$ indicates a postprocessing step on the semi-final hash value. Below, we describe several examples of each component category in order to offer some representative components rather than an exhaustive list.

### 2.1 Preprocessors

Szydlo and Yin describe two types of preprocessing that might be performed on the input to a hash algorithm: local and global [29]. A local preprocessor is only required to keep the current input block in memory, while a global preprocessor may operate upon any part of the input. This can be thought of as analogous to the sequential access of data on a magnetic tape versus the random access of data on a disk. When the entire input to a hash algorithm is not received at once, global preprocessing is impossible. Applications with inputs of indefinite lengths are said to have a streaming requirement, which requires that cryptographic hash algorithms within these applications only use local preprocessors or global preprocessors which can process the message serially. Merkle-Damgård Strengthening is an example of such a global preprocessor which maintains a counter value of the length of the input, which it then appends to the input after the last block is reached.

Local preprocessors which use counter values have also been discussed, such as in Biham and Dunkelman's HAIFA framework [7]. In the HAIFA framework, a technique is introduced which appends a fixed-length counter containing the number of bits that the hash algorithm has so far processed to the end of every data block. This is similar to MD-strengthening, except this length field is appended to every block, not just the last, and obviates the need for MD-strengthening. Szydlo and Yin proposed whitening and message self-interleaving as specific local preprocessing techniques to defeat known collision attacks in MD5 [29]. The whitening technique inserts fixed blocks of data at intervals within the input stream. Message self-interleaving repeats every byte of data in fixed-length blocks before feeding the input stream to the internal structure. Both techniques serve to diffuse the input across more iterations of the compression function, and may operate on entire blocks. The drawback with these preprocessors is that the overall speed of the hash algorithm is reduced. Kauer et al [16] proposed a technique called local tagging which requires each input block to be hashed separately, then the results of the single-block hash appended and prepended to the data block as "tags". This effectively results in each input byte being hashed twice – once to generate the tags and again with the tags in place alongside the actual input bytes.

One global preprocessing method offered in [16] prepends the length of the input to the input itself, which makes it a global preprocessor since the length of the input is not known upon processing of the first input block. A global technique called complete striped hashing [11] reorders the input data such that the $i^{th}$ byte of the processed input is taken from the original input at block $i \cdot n \bmod |M|$, where $|M|$ is the length of the input, and $n$ is relatively prime to $|M|$. This has the benefit of disrupting certain block injection or length-extension collision attacks by spreading any injected or modified data to various places in the preprocessed input. The drawbacks are that it is a global technique and that collision attacks may still be performed in anticipation of the preprocessing step.

## 2.2 Internal Structure

In this paper, we avoid using the terms "iterative structure" or "domain extension" [4] in favor of "internal structure", as the internal structure of a hash algorithm need not be iterative, nor must the internal structure extend the domain of an internal function. It could operate on the entire message at once and never resemble the Merkle-Damgård (MD) construction, which is the structure in use by several common hash algorithms, including MD5 [34], SHA-1 [35], HAVAL [31] and Tiger [2]. While the MD iterative structure maintains the collision resistance of underlying compression functions, it does not necessarily maintain any preimage resistance properties that the compression function may possess. There has been research on the topic of security property preservation in internal structures [4], and these definitions may prove useful in the choice or design of internal structures in cryptographic hash algorithms.

A structure similar to Merkle-Damgård and based upon a block cipher as the underlying compression function is the Davies-Meyer (DM) iterative structure [21]. Variants of the DM structure include Matyas-Meyer-Oseas (MMO) and Miyaguchi-Preneel (MP). These structures either encrypt the output of the previous compression with the message block or encrypt the message block with the output of the previous compression. After the block cipher has returned its result, and depending on the specific structure, the output of the block cipher is then exclusive-ORed with the message block, the previous compression output, or both. Lucks' Double-Pipe Hash [20] is a stronger variant of MD that is separate from DM and uses two "levels" of compression functions such that each message block is compressed twice (starting with two different initialization vectors) such that two separate chaining values are kept. This "cascading" of compression functions results in an internal structure which is provably more resistant against (multiple) preimage attacks. The Double-Pipe Hash can be further extended to a $k$-Pipe Hash (with $k$ levels of compression functions) to further reinforce the hash algorithm against such attacks. The Double-Pipe Hash is only half as fast as a typical MD-based algorithm, and the $k$-Pipe Hash is slower by a factor of $k$.

Tree-based hash algorithms on the other hand [5], do not process input data in the same iterative manner as MD and its variants. Each input block is placed at the leaves of a balanced binary (or *n*-ary) tree. Each block is compressed, then the resulting hash value is paired with the hash value of the appropriate neighbor block, which are both then passed to the compression function again, and so on, until the hash for the entire tree is calculated. The drawbacks to this approach are that it requires $O(\log n)$ memory and twice as much time to calculate the hash as a linear, iterative solution. A benefit to this structure is that intermediate hashes may be kept to speed up recalculation of the overall hash should changes occur in only a few blocks of the input.

## 2.3 Compression Functions

One of the most straightforward ways to construct a compression function for a hash algorithm is to reuse a function that is already available, namely a block cipher [19, 26]. Since a block cipher takes two inputs whose lengths combined are longer than its output, it seems that the cipher would be an ideal candidate for a compression function. The previously-mentioned DM, MMO and MP iterative structures do exactly this, though the block cipher to be used is left as a black box. Conceivably, any block cipher known to be secure could be used in this capacity. We reiterate, though, that it is unclear whether a cryptographic hash algorithm based upon a block cipher actually achieves the practical security goals of a hash algorithm.

With this in mind, several dedicated hash algorithms (and consequently compression functions) have been proposed which do not rely on existing constructions like block ciphers. Instead, many dedicated hash algorithms like MD5, SHA-1 and Tiger use combinations of Boolean functions and substitution boxes on the input data to achieve one-way compression. These Boolean compression functions are sometimes chosen based on desirable features such as those described in [31] and include non-linearity, equal distribution of 0-bits and 1-bits, satisfaction of the Strict Avalanche Criterion [30], and independence of output bits. Other dedicated hash algorithms use compression functions which rely upon or are similar to well-known schemes, such as WHIRLPOOL's compression function which is similar to AES/Rijndael's matrix operations and Very Smooth Hash (VSH) [9] which uses modular exponentiation.

## 2.4 Postprocessors

Postprocessing, output filtering or finalization of the hash, that is, modification of the "semi-final" hash value, has received significantly less attention than the other components discussed. One reason for this is that postprocessing is of limited utility in improving the collision resistance of hash algorithms, though these techniques may help defend against certain preimage and algebraic attacks. Some postprocessing schemes which cascade the results of multiple hash algorithms have been explored, but Boneh and Boyen [8] proved that there is no more efficient way than concatenation to combine the output of two collision-resistant hash functions while retaining the collision resistance of the compression functions. This result was improved upon by Pietrzak [24] who removed the assumption that each individual hash could be computed at most once.

Nevertheless, algorithms sometimes provide a hash value which is longer than the application requires. For example, the SHA-384 algorithm is simply the SHA-512 algorithm with a different initialization value and its output truncated to 384 bits. Truncation and final compression (sometimes called finalization) are two postprocessing steps which have been proposed to cater to applications requiring a hash value of a particular length. The truncation method simply discards the most-significant or least-significant bits from a semi-final hash to yield a final hash value which is of the desired length. This method requires the compression function used in the hash algorithm to be collision resistant not only over the entire hash value, but for the truncated value as well. Truncation is not the postprocessor of choice for some other schemes. One scheme proposed by Bellare and Ristenpart called Enveloped Merkle-Damgård (EMD) [4] applies a final round of compression which uses a different initializing value (IV) to compress the semi-final hash value into the final hash value. A scheme used by the HAVAL algorithm [31] uses a final exclusive-OR-based compression of its 256-bit result depending on whether a 128-, 160-, 192- or 224-bit hash value is required. Though these techniques have been proposed and in some cases are in active use, the secure way to compress a final hash value into a smaller size without losing desirable security properties otherwise remains an open research problem.

## 2.5 Application

This modular building-block approach allows for a certain amount of interchangeability. Using this framework, hash algorithm designers can choose the components that most closely satisfy their security and performance requirements a la carte. It also allows cryptographers to analyze both the strength of components individually and interactions between existing components without having to analyze the entire algorithm monolithically. The framework offers the opportunity to treat components as black boxes or to ignore them altogether while evaluating the security of another component. Determining the strength of a compression function, for example, does not require one to include the preprocessing component in the evaluation. However, evaluation of the security of a particular iterative internal structure requires consideration of the compression function. For example, one compression function component may not suit a particular iterative structure due to block size differences, known vulnerabilities with the use of the two, or poor performance.

One might even construct algorithms by combining components of well-known hash algorithms in novel ways. The framework also allows for parameterizable design, so that a hash algorithm need not define only one method of operation. For example, Lucks proposes that "...the size *w* of the internal hash values is a security parameter of its own right" [20] and HAVAL has its final hash length and number of rounds of the compression function as parameters. These parameters are easily supported by this framework. Taking this to a logical extreme, an application using a cryptographic hash algorithm might have all of the four components as runtime parameters. An application with the streaming requirement might use a local preprocessor, but later might choose a global preprocessor if the entire input is available. In a smart card environment where processing power is limited, the algorithm might choose a compression function that is easier to compute when a "full strength" version is not required. Applications may also make decisions based on how parallelizable the calculation of the hash value must be.

Having described each part of the four-part framework of hash algorithm design, we now apply this framework to decompose the hash algorithms MD5, SHA-1, Tiger, WHIRLPOOL [3], HAVAL and RadioGatún [6], the results of which can be found in Table 1.

| Algorithm | Hash Length (bits) | Preprocessing | Structure | Compression Function(s) | Postprocessing |
|---|---|---|---|---|---|
| **MD5** | 128 | MD-Strengthening | Merkle-Damgård Iterative | Non-linear boolean | None |
| **SHA-1** | 160 | MD-Strengthening | Merkle-Damgård Iterative | Non-linear boolean | None |
| **Tiger** | 192/160/128 | MD-Strengthening | Merkle-Damgård Iterative | Non-linear boolean & S-boxes | Truncation to desired hash length |
| **WHIRLPOOL** | 512 | MD-Strengthening | Miyaguchi-Preneel Iterative | Block cipher similar to Rijndael | None |
| **HAVAL** | 256/224/192/160/128 | Length, version, parameter padding | Merkle-Damgård Iterative | Boolean functions with specific security criteria | Compression to desired hash length |
| **RadioGatún** | Any multiple of 2 words | MD-Strengthening | "Belt-and-Mill" custom two-function iterative | 1 linear, 1 non-linear with feedback | Repeated "blank" iteration until desired hash length is reached |

Table 1: Existing Hash Algorithms in the Four-Component Framework

## 3 ATTACKS

Each component of a hash algorithm plays a role in the strength of the overall algorithm. Weaknesses in a single component can expose the algorithm to various attacks which may compromise its overall security. The specific security features of the hash algorithm as a whole can be described by explicitly enumerating the attacks which are infeasible by virtue of its components. With this in mind, we will describe the attacks against which a strong hash algorithm should be resistant, then discuss the components that are responsible for protecting the algorithm against each.

### 3.1 Collision attacks

A hash function $h$ is collision resistant if it is computationally infeasible to determine inputs $a$ and $b$ such that $h(a) = h(b)$. All four components play a role in the collision resistance of a cryptographic hash algorithm. Clearly, a compression function plays a large role in maintaining the collision resistance of a hash algorithm, as a weak compressor may expose the algorithm to single-block collisions for which the other components can not compensate. An internal structure component can maintain the collision resistance properties of a compression function, one example of which is the MD iterative structure which provably maintains the collision resistance of the underlying compression function. Coron et al suggest strengthening iterative algorithms with a local preprocessor which appends a 0-bit to each non-final block and appends a 1-bit to the final block [10]. In the same paper, they explain that postprocessing by truncation can defeat collision attacks based on length extension by hiding a portion of the final hash. It has also been determined that finding multicollisions – collisions in iterated hash algorithms beyond the first – does not require much more additional work [14, 15, 22]. Fortunately, alternate internal structures such as those described in [20] can increase the work required to find these multicollisions to a much safer level.

### 3.2 Preimage & second preimage attacks

A hash function $h$ is preimage resistant if, given $y$, it is computationally infeasible to determine an input $a$ such that $h(a) = y$ and second preimage resistant if, given $b$ and $y$, it is computationally infeasible to determine an input $a$ such that $h(a) = h(b) = y$. If one can prove the collision resistance of a hash algorithm, then one can also prove the second preimage resistance of an algorithm [21]. The same cannot be said for preimage resistance. While the MD structure preserves the collision resistance of a compression function, it does not necessarily maintain its preimage resistance. Structures such as the aforementioned Double-Pipe Hash, on the other hand, do help to enforce the algorithm's preimage resistance. Length extension attacks such as those described in [13] and a generalization in [18] can be prevented by trivial message preprocessing such as HAIFA's Bits So Far [7].

### 3.3 Other freedom properties & security requirements

Applications with additional specific security requirements should be expected, and candidates for the four components can be evaluated individually for how well they satisfy those requirements. In [1], Anderson described several freedom properties for cryptographic hash algorithms other than collision, preimage and second preimage resistance which may be quite important in certain applications. He first describes complementation freedom in a function $h$ which means it is infeasible to find inputs $a$ and $b$ such that $h(a) = {\sim}h(b)$. He also offers several others, mostly involving arithmetic operations, such as addition freedom, where it is infeasible to find $a$, $b$, and $c$ such that $h(a) + h(b) = h(c)$. If the compression function does not provide the desired freedom properties, postprocessing or preprocessing steps can be introduced that explicitly destroy any algebraic structure within the hash algorithm which might violate the freedom properties [12]. We posit without proof that a careful choice of components can guarantee any explicit finite set of arithmetic freedom properties.

## 4 CONCLUSIONS & FUTURE WORK

Unfortunately, this framework does not account for the interdependencies between components, and determining a generalized way for components to interconnect is an interesting future research topic. The RadioGatún [6] hash algorithm, for example, tightly couples its internal structure to its compression functions. Other simpler interdependencies like the block length or input size are easier to decouple and fit within the framework. The fact that the components are distinct and logically separate in this paper should not be understood to mean that we believe that all hash algorithms must follow this pattern. We do do believe, though, that this divide-and-conquer approach will allow cryptographers to construct strong cryptographic hash algorithms from vetted and well-known building blocks. When a component is found to be weak or unsuitable in a particular application, it can be replaced by one which is known to be strong. That is not to say that there are any guarantees that creating a new hash algorithm from existing strong components will yield a strong overall algorithm, but best practices should be assembled and researched to determine the best methods for constructing algorithms piecewise. We would suggest the following very loose sequence for designing iterated cryptographic hash algorithms:

1. Determine the security and usability properties desired
2. Determine the hash length(s) and length of the internal chaining value based on the desired properties of the algorithm
3. Choose or design compression functions which satisfy the relevant security properties and output chaining values of appropriate length
4. Choose or design an internal structure which interoperates with the chosen compression function and satisfies the relevant security properties
5. Choose or design preprocessing steps which interoperate with the internal structure and satisfy the relevant security properties, if necessary
6. Choose or design postprocessing steps which interoperate with the internal structure and provide the required hash length, if necessary
7. Confirm that the construction interoperates correctly and provides the desired security properties.

## 5 REFERENCES

[1] Anderson, Ross, "The Classification of Hash Functions" (1993) *Proc. IMA Conf. Crypto. & Coding*. http://citeseer.ist.psu.edu/anderson93classification.html. (Accessed November 28, 2006)

[2] Anderson, Ross and Eli Biham, "Tiger: A Fast New Hash Function," (1995) http://www.cs.technion.ac.il/~biham/Reports/Tiger/tiger.html (Accessed August 25, 2006)

[3] Barreto, Paulo S.L.M. and Vincent Rijmen, "The WHIRLPOOL Hashing Function," (2003) http://planeta.terra.com.br/informatica/paulobarreto/whirlpool.zip (Accessed January 23, 2007)

[4] Bellare, Mihir and Thomas Ristenpart, "Multi-Property-Preserving Hash Domain Extension: The EMD Transform," (2006), NIST's Second Cryptographic Hash Workshop. http://www.csrc.nist.gov/pki/HashWorkshop/2006/Papers/RISTENPARThashdomext1.pdf (Accessed January 22, 2007)

[5] Bellare, Mihir and Phillip Rogaway, "Collision-Resistant Hashing: Towards Making UOWHFs Practical," *Proc. CRYPTO 1997*. (1997): 470-484.

[6] Bertoni, Guido, Joan Daemen, Gilles Van Assche and Michaël Peeters, "RadioGatún, a belt-and-mill hash function," (2006) NIST's Second Cryptographic Hash Workshop. http://csrc.nist.gov/pki/HashWorkshop/2006/Papers/VANASSCHE_RadioGatun_0720.pdf (Accessed November 28, 2006)
[7] Biham, Eli and Orr Dunkelman, "A Framework for Iterative Hash Functions: HAIFA," (2006) NIST's Second Cryptographic Hash Workshop. http://csrc.nist.gov/pki/HashWorkshop/2006/Papers/DUNKELMAN_NIST3.pdf (Accessed August 30, 2006)
[8] Boneh, Dan and Xavier Boyen, "On the Impossibility of Efficiently Combining Collision Resistant Hash Functions," *Proc. CRYPTO 2006.* (2006): 570-583.
[9] Contini, Scott, Arjen Lenstra, and Ron Steinfeld, "VSH, an Efficient and Provable Collision Resistant Hash Function," (2005) NIST's First Cryptographic Hash Workshop. http://www.csrc.nist.gov/pki/HashWorkshop/2005/Nov1_Presentations/LENSTRA_vsh.pdf (Accessed June 14, 2006)
[10] Coron, Jean-Sébastien, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya, "A New Design Criteria for Hash-functions" (2005) NIST's First Cryptographic Hash Workshop. http://www.csrc.nist.gov/pki/HashWorkshop/2005/Nov1_Presentations/Puniya_hashDesign.pdf (Accessed February 12, 2007)
[11] Davida, George I. and Jeremy A. Hansen, "A Preliminary Exploration of Striped Hashing: A probabilistic scheme to speed up existing hash algorithms," *Proc. ICETE 2005*. (2005): 364-367.
[12] Davida, George I. And Jeremy A. Hansen, "Preprocessing and Functional Freedom Properties in Cryptographic Hash Algorithms," Submitted to CCS 2007.
[13] Dean, Richard Drews, "Formal Aspects of Mobile Code Security," (1999) Unpublished PhD Dissertation. http://www.cs.princeton.edu/sip/pub/ddean-thesis.pdf (Accessed January 22, 2007)
[14] Hoch, Jonathan J. and Adi Shamir, "Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions," *Proc. Fast Software Encryption 2006*. (2006): 179-194.
[15] Joux, Antoine, "Multicollisions in Iterated Hash Functions: Application to Cascaded Constructions," (2004) *Proc. CRYPTO 2004*. (2004): 306-31.
[16] Kauer, Neil, Tony Suarez, and Yuliang Zheng, "Enhancing the MD-Strengthening and Designing Scalable Families of One-Way Hash Algorithms," (2005) NIST's First Cryptographic Hash Workshop. http://www.csrc.nist.gov/pki/HashWorkshop/2005/Nov1_Presentations/Zheng-designing-hash-family-sbm2NIST.pdf (Accessed January 26, 2007)
[17] Kelsey, John and Tadayoshi Kohno, "Herding Hash Functions and the Nostradamus Attack," (2006), NIST's First Cryptographic Hash Workshop. http://eprint.iacr.org/2005/281.pdf (Accessed January 22, 2007)
[18] Kelsey, John and Bruce Schneier, "Second Preimages on n-Bit Hash Functions for Much Less than $2^n$," *Proc. EUROCRYPT 2005*. (2005): 474-490.
[19] Lai, Xuejia and James Massey, "Hash Functions Based on Block Ciphers," *Proc. EuroCrypt 1992*. (1993): 55-70.
[20] Lucks, Stefan, "Design Principles for Iterated Hash Functions," (2004) IACR ePrint Archive, http://eprint.iacr.org/2004/253.pdf (Accessed January 22, 2007)

[21] Menezes, Alfred J., Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography.* http://www.cacr.math.uwaterloo.ca/hac/ (Accessed January 22, 2007).

[22] Nandi, M. and D. R. Stinson, "Multicollision Attacks on Some Generalized Sequential Hash Functions," (2005) http://csrc.nist.gov/pki/HashWorkshop/2006/UnacceptedPapers/NANDI_hashworkshop_nandistinson.pdf (Accessed June 14, 2006)

[23] Nechvatal, James and Shu-jen Chang "Workshop Report: The Second Cryptographic Hash Workshop" http://www.csrc.nist.gov/pki/HashWorkshop/2006/SecondHashWshop 2006 Report.pdf (Accessed January 19, 2007)

[24] Pietrzak, Krzysztof, "Non-Trivial Black-Box Combiners for Collision-Resistant Hash Functions don't Exist," IACR ePrint Archive, http://eprint.iacr.org/2006/348.pdf (Accessed June 8, 2007)

[25] Preneel, Bart, "Analysis and Design of Cryptographic Hash Functions," (1993) PhD Thesis. http://www.esat.kuleuven.ac.be/~preneel/phd_preneel_feb1993.pdf (Accessed January 22, 2007)

[26] Preneel, Bart, René Govaerts, and Joos Vandewalle, "Hash functions based on block ciphers: a synthetic approach," *Proc. CRYPTO 1993*. (1994): 368-378.

[27] Rogaway, P. and T. Shrimpton. "Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance." *Proc. FSE 2004*. (2004): 371-388.

[28] Rogaway, Philip, "Formalizing Human Ignorance: Collision-Resistant Hashing without the Keys," *Proc. Vietcrypt 2006.* (2006): 221-228.

[29] Szydlo, Michael and Yiqun Lisa Yin, "Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing," IACR ePrint Archive, http://eprint.iacr.org/2005/248.pdf (Accessed January 22, 2007)

[30] Webster, A. F. and S. E. Tavares, "On the Design of S-Boxes," *Proc. CRYPTO 1985*. (1986): 523-534.

[31] Zheng, Yuliang, Josef Pieprzyk, and Jennifer Seberry, "HAVAL – A One-Way Hashing Algorithm with Variable Length of Output," *Proc. AUSCRYPT 1992*. (1993): 83-104.

[32] "Hash Function Workshop" http://www.csrc.nist.gov/pki/HashWorkshop/index.html (Accessed January 4, 2007)

[33] "Notes from the Hash Futures Panel" http://www.proper.com/lookit/hash-futures-panel-notes.html (Accessed February 19, 2007)

[34] "The MD5 Message-Digest Algorithm" http://www.ietf.org/rfc/rfc1321.txt (Accessed February 19, 2007)

[35] "US Secure Hash Algorithm 1 (SHA1)" http://www.ietf.org/rfc/rfc3174.txt (Accessed Feburary 19, 2007)