International Journal of Information Technology and Knowledge Management
January June 2009, Volume 2, No. 1, pp. 15-20

SURVEY OF HASH FUNCTION: RESISTANCE TO FINDING ATTACKS

A. Arul Lawrence Selvakumar[*] & C. Suresh Ganadhas[**]

We survey theory of cryptographic hash functions, such as MD5 and SHA-1, especially their resistance to collision-Finding attacks. We review definitions, design principles, trace genealogy of standard hash functions, discuss generic attacks, attacks on iterative hash functions, and recent attacks on specific functions.

Keywords: Finding-collision attacks, hash function, cryptographic, cryptanalyst, cryptosystem, crypto analysis.

1. Introduction

Hash functions, most notably MD5 and SHA-1, initially crafted for use in a handful of cryptographic schemes with specific security requirements, have become standard fare for many developers and protocol designers who treat them as black boxes with magic properties. This practice had not been seriously challenged until 2004, since both functions appeared to have withstood the test of time and intense scrutiny of cryptanalysts. Starting last year, we have seen an explosive growth in the number and power of attacks on the standard hash functions. In this note we discuss the extent to which the hash functions can be thought of as black boxes, review some recent attacks and, most importantly, revisit common applications of hash functions in programming practice.

2. Theory of Hash Function

In this section we introduce notation, define security properties of hash functions, describe basic design principles of modern hash functions and generic attacks.

2.1 Notation

The following notation used in this note is standard in the cryptographic literature:

$\{0,1\}^n$	The set of all binary strings of length n.
$\{0,1\}^*$	The Set of all finite binary strings
$A \times B$	The set of all pairs (v, w) where $v \in A$ and $w \in B$.
$H: A$	B–function H from set A to set B.
$\|w\|$	the length of string w.
$w\|\|v$	Concatenation of strings w and v.

[*] Department of CSE, Royal College of Engineering & Technology, Kerala, INDIA, *E-mail: aarul72@hotmail.com.*

[**] Department of CSE, Velmultitech SRS Engineering College, Tamil Nadu, INDIA, *E-mail: sureshc.me@gmail.com.*

2.2 Definitions

In practice, the hash function (sometimes called the message digest) is a fixed function that maps arbitrary strings into binary strings of fixed length. In theory, we usually consider keyed hash functions, as in the following

Definition: Let ℓ, n is positive integers. We call f a hash function with n-bit output and ℓ-bit key if f is a deterministic function that takes two inputs, the first of arbitrary length, the second of length ℓ bits, and outputs a binary string of length n. Formally,

$$H : \{0, 1\}^* \times \{0, 1\}\ell \rightarrow \{0, 1\}^n.$$

$H(x, k)$ is an expressive shorthand for $H(x, k)$. The key k is assumed to be known unless indicated otherwise (to avoid confusion with cryptographic keys, which typically represent closely guarded secrets, the hash function's key is sometimes called the "index"). We distinguish between three levels of cryptographic applications.

The definitions are framed as games that are infeasible to win by a computationally-bounded adversary. security that hash functions may satisfy to be useful in The words "infeasible" and "computationally bounded" can be formalized in several ways, neither of which we find entirely satisfying for the purpose of this note. Instead, we offer a semi-formal semantic where we say that the problem is computationally infeasible if no program performing less than T elementary operations can solve the problem with probability higher than $T/2^{80}$. It corresponds to the so-called 80-bit security level, which is currently acceptable for most applications.

Definition: Hash function H is one-way if, for random key k and an n-bit string w, it is hard for the attacker presented with k, w to find x so that $H_k(x) = w$.

Definition: Hash function H is second - preimage resistant if it is hard for the attacker presented with a random key k and random string x to find $y \neq x$ so that $H_k(x) = H_k(y)$.

Definition: Hash function H is collision resistant if it is hard for the attacker presented with a random key k to find x and $y \neq x$ so that $H_k(x) = H_k(y)$.

The last definition is noticeably harder to formalize for keyless hash functions, which explains why theorists prefer keyed hash functions. It is easy to see that collision resistance implies second-preimage resistance.

Strictly speaking, second-preimage resistance and one-wayness are incomparable (the properties do not follow from one another), although construction which are one-way but not second-preimage resistant are quite contrived. In practice, collision resistance is the strongest property of all three, hardest to satisfy and easiest to breach, and breaking it is the goal of most attacks on hash functions.

Certificational weakness: If a good hash function must satisfy other properties not implied by one-wayness or even collision-resistance. For example, one would expect that flipping a bit of the input would change approximately half the bits of the output (avalanche property) or that no inputs bits can be reliably guessed based on the hash function's output (local one-wayness). Failure to satisfy those or similar properties, such as infeasibility of finding a pseudo-collision, free-start collisions are called a *certificational weakness.* Presence of certificational weaknesses does not amount to a break of a hash function but is enough to cast doubt on its design principles.

2.3 Generic Attacks

A generic attack is an attack that applies to all hash functions, no matter how good they are, as opposed to specific attacks that exploit flaws of a particular design. The running time of generic attacks is measured in the number of calls to the hash function, which is treated as a black box.

It is not dificult to see that in the black box model the best strategy for inverting the hash function and for finding a second preimage is the exhaustive search. Suppose the problem is to invert H_k, i.e., given w, k find x, so that $H_k(x) = w$, where k is ℓ-bit key and w is an n-bit string. The only strategy which is guaranteed to work for any hash function is to probe arbitrary chosen strings until a preimage of w is hit. For a random function H it would take on average 2^{n-1} evaluations of H. In the black-box model the problem of finding a second preimage is just as hard as inverting the hash function.

Finding collisions is a different story, the one that goes under the name of the *"birthday paradox."* The chances that among 23 randomly chosen people there are two who share the same birthday are almost 51%. The better than even odds appear to be much higher than the intuition would suggest, hence the name. Of course, there is nothing paradoxical about the fact–between 23 people there are 253 pairs, each of which has a one-to-365 odds of being a hit. The reason why the result appears counter-intuitive is because your chances of finding among 22 people somebody with the same birthday as yours to split the cost of the birthday party are strongly against you. This explains why inverting a hash function is much more dificult than finding a collision.

More careful analysis of the birthday paradox shows that in order to attain probability better than 1/2 of finding a collision in a hash function with n-bit output, it suffices to evaluate the function on approximately 1.2 $2^{n/}2$ randomly chosen inputs (notice that $\lceil 1.2 \sqrt{365} \rceil = 23$).

The running times of generic attacks on different properties of hash functions pro vide upper bounds on security of any hash function. We say that a hash function has *ideal security* if the best attacks known against it are generic. Cryptanalysts consider a primitive broken if its security is shown to be less than ideal, even though it may still be sufficient for some applications. The generic attacks are summarized in Table 1.

Table 1
Complexity of Generic Attacks on Different Properties of Hash Function

Property	*Ideal security*
One way ness	2^n-1
Second Pre image – resistance	2^n-1
Collision – resistance	$1.2.2^{n/2}$

2.4 Constructions

Provably secure constructions of cryptographic hash functions consist of two ingredients, which may be studied independently of each other. The first component is a *compression function* that maps a fixed-length input to a fixed-length output.

The second component of a construction is a *domain extender* that, given a compression function, produces a function with arbitrary-length input.

Compression function: From the theorist's point of view, a one-way function is the most basic primitive, from which many other cryptographic tools can be derived. A seminal result due to Simon [Sim98] provides strong evidence that collision-resistant hash functions cannot be constructed based on one-way functions. Instead, we design collision-resistant hash functions based on another cryptographic primitive - a block cipher.

A block cipher is a keyed permutation

$$E\colon \{0, 1\}^n \times \{0,1\}^k \rightarrow \{0, 1\}^n.$$

Technically, a block cipher already compresses its input - it maps $k+ n$ to n bits. As is, however, the block cipher is not even one-way: to invert Eon w, fix any key k_0 and decrypt w under this key. If w decrypts to x, then $E(k_0, x) = w$. Nonetheless, as many as 12 simple constructions based on a block cipher result in a collision-resistant compression function [BRS02] two schemes most often used in hash functions are the following:

Davies-Meyer : $H(x, y) = E_y(x) \in y$

Miyaguchi-Preneel : $H(x, y) = E_x(y) \in x \in y$.

Proofs of security of these and similar block cipher-based constructions assume that the underlying cipher are in distinguishable from a certain abstraction, called the ideal cipher, which goes beyond the standard security requirements for block ciphers.

2.5 Domain Extender

The domain extender is a generic construction that transforms a compression function with fixed-length input into a hash function with arbitrary input. The simplest and most commonly used domain extender is called the *Merkle-Damgard construction and it works as follows:*

Given: Compression function C: $\{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$; n-bit constant IV.

Input: message M

1. Break M into m-bit blocks $M_1, \ldots, M_k$, padding
2. Let $M_k + 1$ be encoding of $|M|$;
3. Let $h_0 = \text{IV}$;
4. For $i = 1$ to $k + 1$ let $hi = C(hi_{-1}, M_i)$;
5. Output h_{k+1}.

The construction iterates the compression function C: the output of C, together with the next block of the message, becomes the input to the next application of C. The hash of the last block, which contains an encoding of the length of the message, is the hash of the entire message.

The temporary storage of the compression function's output, h_i, is called the chaining variable or the internal state (see Figure 1).

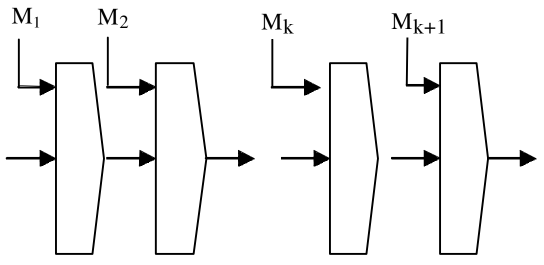

Figure 1: Merkle-Damgard Construction

There is a certain flexibility in the first two steps of the Merkle-Damgard construction. Any encoding will do as long as it satisfies the following three conditions:

– M is encoded as an integral number of m-bit blocks;
– The encoding is collision-free;
– The length of M is encapsulated in the last block.

The Merkle-Damgard construction is compatible with streaming APIs, where a message is fed one block at a time into a cryptographic search engine. Its length need not be known until the last block becomes available. On the other hand, updating even one bit of the message may trigger recomputation of the entire hash. If the compression function is collision-resistant, so is the resulting construction. However, the Merkle-Damgard construction produces a function with many structural properties, creating a number of unexpected vulnerabilities as illustrated in Section 4.

In fact, the Merkle-Damgard construction is the single most important reason why it is ***wrong (dangerous, reckless, ignorant)*** to think of hash functions as black boxes. The iterative construction was designed to meet a very modest goal that of extending the domain of a collision-resistant function, and should not be expected to provide security guarantees beyond that.

2.6 Algebraic Hash Function

Collision –resistant hash function can be based on the same hardness assumptions as public key cryptography. Such functions are largely confined to theoretical papers, being algebraic and therefore orders of magnitude slower than ad-hoc or block cipher-based hash function. The main reason why we discuss algebraic hash functions in this note is to correct the popular reference [Sch95, Section 18.12], which does not give two standard constructions and describes two incorrect ones instead. Discrete logarithm problem in group G of prime order p is to solve the equation $g^x = h$ for x given two group elements g, $h \in G$. Discrete-logarithm based hash function can be designed as follows: $H(x, y) = g^x h^y$,

Where g, h are elements of a group where the discrete logarithm problem is hard. It is easy to verify that if the inputs to H are defined modulo p, finding a collision amounts to solving the discrete logarithm problem.

The RSA - based hash function is defined for arbitrary strings. If $N = PQ$ is product of two unknown primes, and $g \neq 1$ is an element co-prime with N, then the function defined as $H(x) = g^x \bmod N$ is collision-resistant under the hardness of factoring N.

3. Practical Hash Functions

This section covers hash functions that are most likely to be used in practice: MD5, SHA-1, SHA-256, Whirlpool and their close relatives. For their detailed description we refer the reader to the documents issued by standardization bodies.

MD4 and MD5: MD4 was proposed by Ron Rivest in 1990 and MD5 [Riv92] followed shortly thereafter as its stronger version. Their design had great influence on subsequent constructions of hash function. The letters "MD" stand for "message digest" and the numerals refer to the functions being the fourth and fifth designs from the same hash-function family. MD5 follows the design principles outlined in Section 2.4: its compression function is (implicitly) based on a block-cipher and the domain extender is the Merkle-Damgard construction.

The compression function of MD5 operates on 512-bit blocks further subdivided into sixteen 32-bit sub blocks. The size of the internal state (i. e., the chaining variable) and its output are both 128 bits. One important parameter of the compression function is the number of rounds —the number of sequential updates of the internal state. The compression function of MD5 has 64 rounds organized in an unbalanced Feistel network (for comparison, DES is a Feistel cipher with 16 rounds) each using a 32-bit message sub block to update the internal state via a non-linear mix of boolean and arithmetic operations. Every 32-bit sub block is used four times by the compression function.

MD5 allocates 64 bits in the last block to encode the message's length and it pads the message so that its length is congruent to 448 modulo 512. The padding procedure expands the message by at least one bit, so the largest message fitting into one block is 447 bits.

SHA-0 and SHA-1 The Secure Hash Algorithm (SHA) was initially approved for use with the Digital Signature Standard (DSS) in 1993 [NIST00]. Two years later the standard was updated to become what is currently known as SHA-1 [NIST95]. The first version of SHA is referred in the cryptographic literature as SHA-0, although it has never been its official designation. SHA-1 differs from SHA-0 by exactly one additional instruction, which is nonetheless extremely important from the cryptanalytic perspective, Since there were no reasons to prefer the initial version of the standard, SHA-1 replaced SHA-0 in all but most antiquated applications.

SHA-1 is closely modeled after MD4, taking some cues from MD5. It uses the same padding algorithm, breaking the message into 512-bit blocks and encoding the length as a 64-bit number. The size of its internal state and its output length are 160 bits, which is substantially longer than MD5's 128 bits. Although its round functions are simpler and less varied than those of MD5, there are more of them—80 instead of 64. SHA-1 uses a more complex procedure for deriving 32-bit sub blocks from the 512-bit message. If one bit of the message is fliipped, more than a half of the sub blocks get changed It is interesting to note that the cipher, which operates inside the compression function, has never been given any officially recognition. It did not stop the cryptanalytic community, which dubbed the cipher SHACAL, from isolating and studying it.**SHA-224, 256, 384, 512**. The new standard issued by NIST in August 2002 adds three members to the SHA family of functions [NIST02], followed by one more in 2004.

The connections between the NIST-approved functions are following: SHA-256 and SHA-512 have similar designs, with SHA-256 operating on 32-bit words and SHA-512 operating on 64-bit words. Both designs bear strong resemblance to SHA-1, although they are much closer to each other than to their common predecessor. SHA-384 is a trivial modification of SHA-512, which consists of trimming the output to 384 bits and changing the initial value of the chaining variable. A change notice issued in February 2004 defined SHA-224 as a truncated version of SHA-256 with a different initial value. The new hash function is to provide 112-bit level of security, on par with triple DES.

The most important difference between the three new functions and SHA-1 is the new message schedule (procedure for deriving sub blocks from one block of the message).

Whirlpool: Whirlpool was designed by Paulo Barreto and Vincent Rijmen (the latter is of AES's fame) and submitted in response to the call for cryptographic primitives issued by NESSIE (New European Schemes for Signature, Integrity, and Encryption) in 2000. Whirlpool was selected together with SHA-256,384,512 as part of NESSIE's portfolio.

Whirlpool's design combines the Merkle-Damgard domain extender with a blockcipher based compression function. The blockcipher is a variant of AES, which is radically different from SHACAL, and it is converted into a compression function using the Miyaguchi-Preneel construction.

Whirlpool does not target any particular architecture, although 32- or 64-bit processors permit some optimizations impossible in 8-bit implementations.

We summarize in Table 2 parameters of the standard hash functions. Table 3 presents performance of some of the hash functions on selected processors compiled from two studies [P+03, NM02]. The first report considered portable C implementations of the hash functions limiting optimization to a choice between different combinations of the compiler's options. The second study undertook a painstaking optimization of the assembly language code for Pentium III, taking advantage of the MMX registers and carefully orchestrated pipeline scheduling.

Table 2
Standard Hash Functions in a Glance

Name	*Block Size*	*Word Size*	*Output Size*	*Rounds*	*Year of the standard*
MD4	512	32	128	48	1990
MD5	512	32	128	64	1992
SHA-0	512	32	160	80	1993
SHA-1	512	32	160	80	1995
SHA-224	512	32	224	64	2004
SHA-256	512	32	224	64	2002
SHA-384	1024	64	384	80	2002
SHA-512	1024	64	512	80	2002
Whirlpool	512	–	512	10	2003

Table 3
Performance in Cycles/Byte

Name	*PIII C, Visual C 6.0*	*XeonC, gcc3.2*	*PIIIAssembly, MASM*
MD4	4.8	6.4	–
MD5	7.2	9.4	3.7
SHA-1	19	25	8.3
SHA-256	56	39	20.6
SHA-512	80	135	40.2
Whirlpool	82	112	36.5

4. GENERIC ATTACKS

In this section we discuss generic attacks against schemes that use iterative hash functions based on the Merkle-Damgard construction, highlighting pitfalls of using such functions as black boxes, and review specific attacks against standard hash functions.

Black box abstraction: Software engineers and researchers alike are trained to re duce complexity of systems by thinking of the systems' components as "black boxes" modules with well-defined interface, separating functionality from Implementation details. It is also common to treat programming objects as real-life models of mathematical abstractions. For instance, we use the double type to represent real numbers, or the **rand()** function as a source of randomness. This pragmatic approach is productive as long as its limits are well understood.

To continue our example, double represents reals with a certain machine-dependent precision and rand ( ) from the standard library has a relatively short cycle—both facts are well documented and known to most C/C++ developers.

It is convenient to think of a concrete hash function such as SHA-1 as a real-world instantiation of a random function. One important characteristic of a random function is that the only way to learn its value on some input is to evaluate the function on precisely this input. As we will show in this section, this is not the case for iterative hash functions. We stress that the generic attacks in this sections are attacks against schemes that uncritically use iterative hash functions, not against hash functions themselves.

5. SPECIFIC ATTACKS

Table 4 summarizes attacks on standard hash functions that appeared in the public literature. Some of the attacks are trivial to interpret, as they expose collisions in the hash function, some call for more definitions.

Definition: We say that (h, x) and (h', x') is a pseudo-collision for compression function

$$C : \{0, 1\}n \times \{0, 1\}m \rightarrow \{0, 1\}n$$

if $\quad C(h, x) = C(h', x')$ and $(h, x) \neq (h', x')$.

A pseudo-collision for a compression function C invalidates the proof of the Merkle Damgard construction (Section 2.4) that states that if the compression function C is collision-resistant, so is the hash function H based on C. It does not, however, translate into an attack on H, since the attacker does not directly control the value of the chaining variable

Definition**:** We say that (h, x) and (h', x') is a free-start collision for compression function

$$C : \{0, 1\}^n \times \{0, 1\}^m \rightarrow \{0, 1\}^n$$

if $\quad C(h, x) = C(h', x')$ and $x \neq x'$.

Notice that a free-start attack is stronger than a pseudo-collision attack, since the attacker is forced to use the same value of the chaining variable for both colliding arguments in the free-start attack. This attack still falls short of a collision-finding attack on C-based hash function H for the same reason as before—the Merkle-Damgard paradigm fixes the initial value of the chaining variable, which is unlikely to coincide with the value anticipated by the adversary, and subsequent values of the chaining variables cannot be easily chosen either.

Definition We say that x and x' is a near-collision for the hash function H if the Hamming distance between $H(x)$ and $H(x')$ is small.

Once again, this attack is not as strong as a collision finder, although it sometimes serves as a precursor to a full attack (as was the case of SHA-0). Reduced-round hash functions are weaker primitives that share important characteristics with their full-round variants. Cryptanalysts often attack reduced-round hash functions first since such attacks may provide insight into potential attack vectors and can be tested using modest computational resources.

Table 4
Attack on Standard Hash Function

Hash	*Attack*			
	Author	*Type*	*Complexity*	*Year*
MD4	Dobbertin	Collision	2^{22}	1996
	Wangetal	Collision	2^{8}	2005
MD5	Dan boer	Pseudo collision	2^{16}	1993
	Dobbertin	Free start	2^{34}	1996
	Wangetal	Collision	2^{39}	2005
SHA-0	Chadaud	Collision	2^{61}	1998
	Biham	Near collision	2^{40}	2004
	Biham	Collision	2^{51}	2005
	Wangetal	Collision	2^{39}	2005
Sha - 1	Biham	Collision 40 round	Very low	2005
	Biham	Collision 58 round	2^{25}	2005
	Wangetal	Collision 58 round	2^{33}	
	Wangetal	Collision	2^{63}	2005

6. CONCLUSION

Research in cryptographic hash function has been active and, in recent years, explosive. We will undoubtedly see new proposals, tweaks of existing design, and attack in years to come.

Reference

[1] B+05 John R. Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, Phillip Rogaway, UMAC—Message Authentication Webpage,*www.cs.ucdavis.edu/~rogaway/umac/*

[2] BRS02 John Black, Phillip Rogaway, and Tom Shrimpton, "Black Box Analysis of the Block-cipher-based Hash-Functions Constructions from PGV," Proc. of CRYPTO'02, Lecture Notes in Computer Science 2442, Springer, (2002), 320–335. *www.cs.ucdavis.edu/~rogaway/papers/hash.htm*

[3] CC02 Douglas Century and Rick Cowan, Takedown: The *Fall of the Last Mafia Empire*, Putnam Publishing Group, (2002).

[4] CLRS01 Thomas H. Cormen, Charles E. Leiserson, Ronald L.Rivest, and Clifford Stein, Introduction to Algorithms, Second Edition, MIT Press, (2001).

[5] CS00 Ronald Cramer and Victor Shoup, "Signature Schemes based on the Strong RSA Assumption," *ACM Transactions on Information and System Security*, **3**(3), (2000), 161–185, *www.zurich.ibm.com/security/ace/*

[6] DL05 Magnus Daum and Stefan Lucks, "Attacking Hash Functions by Poisoned Messages," EUROCRYPT Rump Session, 2005. *www.cits.rub.de/MD5Collisions/*

[7] DA99 T. Dierks and C. Allen, "RFC 2246–The TLS Protocol Version 1.0," IETF RFC 2246, (1999). *www.ietf.org/rfc/rfc2246.txt*

[8] HC98 D. Harkins and D. Carrel, "RFC 2409–The Internet Key Exchange (IKE)," *IETF RFC 2409*, (1998). *www.ietf.org/rfc/rfc2409.txt*

[9]. H+02 R. Houseley, W. Polk, W. Ford, and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," *IETF RFC* 3280, (2002) *www.ietf.org*/rfc/rfc3280.txt

[10] Jou04 Antoine Joux, "Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions," *Proc. of CRYPTO 2004, Lecture Notes in Computer Science* 3152, Springer, (2004), 306–316.

[11] MOV96 Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, Hand Book of Applied Cryptography, CRC Press, 1996. Available from *www.cacr.math.uwaterloo.ca/hac/*

[12] LMS05 P. Leach, M. Mealling, and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace," IETF RFC 4122, 2005. Available from *www.ietf.org/rfc/rfc4122.txt*

[13] LV01 Arjen K.Lenstra and Eric R. Verheul, "Selecting Cryptographic Key Sizes," *Journal of Cryptology*, **14**(4), (2001), 255–293. Available from *www.win.tue.nl/~klenstra/key.pdf*

[14]. LWW05 Arjen K. Lenstra, Xiaoyun Wang, and Benne de Weger, "Colliding X.509 Certificates," Cryptology ePrint Archive, Report 2005/067. Available from eprint.iacr.org/2005/067

[15]. MT79 Robert Morris and Ken Thompson, "Password Security: A Case History," *Communications of ACM*, **22**(11), (1979), 594–597.

[16] NM02 Junko Nakajima and Mitsuru Matsui, "Performance Analysis and Parallel Implementation of Dedicated Hash Functions," Proc. of EUROCRYPT 2002, *Lecture Notes in Computer Science* 2332, Springer, (2002), 165–180.

[17] P+03 Bart Preneel *et al.*, "Performance of Optimized Implementations of the NESSIE Primitives "NES/DOC/TEC/WP6/D21/2, *www.cosic.esat.kuleuven.be/nessie/deliverables/D21-v2.pdf*

[18] Riv 92 Ronald L. Rivest, "The MD5 Message-Digest Algorithm," *IETF RFC* 1321, (1992). Available from *www.ietf.org/rfc/rfc1321.txt*

[19] Sch95 Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, Second Edition, Wiley, (1995).

[20] NIST95 "Secure Hash Standard," FIPS PUB 180-1, 1995. *www.itl.nist.gov/fipspubs/fip180-1.htm*

[21] NIST00"Digital Signature Standard," FIPS PUB 186-2, (2000).

[22] NIST02"Secure Hash Standard," FIPS PUB 180-2, 2002 *www.csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf*

[23] NIST05 "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication," NIST *www.csrc.nist.gov/publications/nistpubs/800-38B/SP_800-38B.pdf.*

[24] Sim98 Daniel R. Simon, "Finding Collisions on a One-way Street: Can Secure Hash Functions be based on General Assumptions?" Proc. of Eurocrypt'98, *Lecture Notes in Computer Science* 1403.