Design principles for dedicated hash functions

Bart Preneel*

Katholieke Universiteit Leuven, Laboratorium ESAT-COSIC,
Kardinaal Mercierlaan 94, B–3001 Heverlee, Belgium
bart.preneel@esat.kuleuven.ac.be

Abstract. Dedicated hash functions are cryptographically secure compression functions which are designed specifically for hashing. They intend to form a practical alternative for hash functions based on another cryptographic primitive like a block cipher or modular squaring. About a dozen of dedicated hash functions have been proposed in the literature. This paper discusses the design principles on which these hash functions are based.

1 Introduction

Cryptographic hash functions form an important building block for providing information authentication. An efficient way to protect the authenticity of a large quantity of information consists of protecting only the short hashcode computed from that information. If the authenticity is protected with a digital signature scheme, one obtains a shorter signature, which is easier to compute.

For the time being there is no theory available to design efficient and secure hash functions. The oldest proposals for hash functions were based on block ciphers and on modular arithmetic. During the last years about a dozen of proposals for dedicated hash functions have appeared in the literature. The goal of this paper is to study the design principles of these hash functions. It is not our intention to give a description of these hash functions or to give an overview of the latest attacks. For such an overview the reader is referred to [20, 21].

In Sect. 2 we give a definition of a hash function and we establish a general model for an iterated hash function. Moreover, we discuss the two main theoretical results on the construction of iterated hash functions. In Sect. 3, we list the hash functions which are the subject of this paper. Section 4 discusses the security relevant criteria, and Sect. 5 treats some performance related aspects. In Sect. 6 it will be explained why a trapdoor can also be a problem for hash functions. The conclusions are presented in Sect. 7.

* N.F.W.O. postdoctoral researcher, sponsored by the National Fund for Scientific Research (Belgium). Part of this work was done while visiting the Department of Electrical Engineering and Computer Sciences of the University of California at Berkeley.

2 Definition and general model

In this paper we will only consider collision resistant hash functions, since all dedicated hash functions intend to be of this type. The concept of collision resistant hash functions dates back to the late seventies. We will give only an informal definition; for a formal definition the reader is referred to the work by I. Damgård [5, 6].

Definition 1 *A* **collision resistant hash function** *is a function h satisfying the following conditions:*

1. *The argument X can be of* arbitrary *length and the result $h(X)$ has a* fixed *length of n bits (with $n \geq 128$).*
2. *The hash function must be* one-way *in the sense that given a Y in the image of h, it is "hard" to find a message X such that $h(X) = Y$, and given X and $h(X)$ it is "hard" to find a message $X' \neq X$ such that $h(X') = h(X)$.*
3. *The hash function must be* collision resistant: *this means that it is "hard" to find two distinct messages that hash to the same result.*

The first part of the second condition corresponds to the intuitive concept of one-wayness, namely that it is "hard" to find a preimage of a given value in the range. Under certain conditions one can argue that this part of the one-way property follows from the collision resistance property [6]. The second part of this condition, namely that finding a second preimage should be hard, is a stronger condition, that is relevant for most applications. Finding a (second) preimage requires at most 2^n operations. Collision resistance is stronger than one-wayness, since for any hash function with a n-bit result, a collision can be found in about $2^{n/2}$ operations with a birthday or square-root attack [31].

Several options are available to specify the word "hard". In the case of "ideal security" [12], producing a (second) preimage requires 2^n operations and producing a collision requires about $2^{n/2}$ operations. For the purpose of this paper it is sufficient to assume that both operations are computationally infeasible.

Almost all known hash functions are based on a compression function with fixed size input; they process every message block in a similar way. This has been called an "iterated" hash function in [12]. The information is divided into t blocks X_1 through X_t. If the total number of bits is no multiple of the block length, the information has to be padded to the required length. The hash function can subsequently be described as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(X_i, H_{i-1}) \qquad i = 1, 2, \ldots t \\ h(X) &= H_t \,. \end{aligned}$$

The result of the hash function is denoted with $h(X)$ and IV is the abbreviation for Initial Value. The function f is called the *round* function, and the H_i's are called the *chaining variables*. Two iterations of an iterated hash function are shown in Fig. 1.

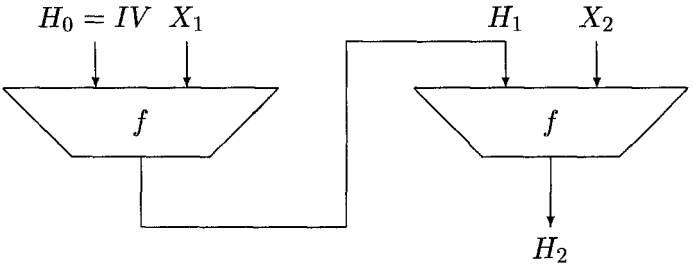

Fig. 1. Two rounds of an iterated hash function.

Two elements in this definition have an important influence on the security of a hash function: the choice of the padding rule and the choice of the IV. It is recommended that the padding rule is unambiguous (i.e., there exist no two messages that can be padded to the same message), and that it appends at the end the length of the message. The IV should be considered as part of the description of the hash function. In some cases one can deviate from this rule, but this will make the hash function less secure and may lead to trivial collisions or second preimages.

An important problem in the context of hash functions is the relation between the security of the hash function h and the security of the round function f. The goal is to find conditions on f which are necessary and/or sufficient for h to have certain properties. This can reduce the design and analysis of a hash function to the study of a function with fixed length inputs.

It can be shown that the security of h and the security of f are equivalent if one uses an unambiguous padding rule and if the attacker is allowed to modify the initial values. This implies that he will look for a *pseudo-preimage*, a (second) preimage with a different value of IV, or for a *pseudo-collision*, which means that for some IV' and IV'' he finds a pair X', X'', such that $h_{IV'}(X') = h_{IV''}(X'')$. It is clear that finding a real (second) preimage or collision cannot be easier than finding a pseudo-preimage or pseudo-collision respectively.

If the IV's are fixed, these pseudo-attacks do not form a direct threat and the situation is more complicated. I. Damgård has shown that for h to be collision resistant it is sufficient that f is collision resistant [6]. X. Lai and J. Massey showed that a similar property holds for one-wayness [12]. But, if the hash function has to be ideally one-way, they showed that is both necessary and sufficient for f to be ideally one-way. A similar result for a different type of hash functions was established in [19].

3 The dedicated hash functions

MDx (Message Digest x) is a series of hash functions designed by R. Rivest. The ones which have been published are MD2, MD4, and MD5.

MD2 is a byte oriented algorithm published in 1990; a description can be found in [11]. MD4 was designed to be fast on 32-bit machines [25, 26]. MD5 is a variant of MD4 [27]. Two other variants of MD4 are the Secure Hash Algorithm, which was published by NIST in the Secure Hash Standard - FIPS 180 [8], and RIPEMD which was developed in the framework of the EEC-RACE project RIPE (Race Integrity Primitives Evaluation) [23]. HAVAL was proposed by Y. Zheng, J. Pieprzyk, and J. Seberry [32]; it is a collection of extensions of MD5.

N-hash is a hash function with $N = 8$ rounds designed by S. Miyaguchi, M. Iwata, and K. Ohta [15, 17]. It is based on the same principles as FEAL [16]. An extended version of N-hash appeared in [18]: the roles of X_i and H_{i-1} can be interchanged, and the number of rounds is now a parameter which should be at least 8. Four rounds are claimed to be sufficient for a one-way hash function, for which the collision resistant property does not have to hold (condition 3 in Definition 1).

FFT-Hash I and II are hash functions suggested by C.P. Schnorr [28, 29] based on the Fast Fourier Transform. A contribution towards a third version is presented in these proceedings [30].

Snefru is a software oriented hash function proposed by R. Merkle [14]. It is based on large random substitution tables (2 Kbyte per pass). While initially 2 to 4 passes were recommended, it is currently advised to use at least 8 passes.

I. Damgård has proposed a hash function based on a cellular automaton in [6]. After breaking this scheme, J. Daemen, J. Vandewalle, and R. Govaerts put forward Cellhash [2]; an improved version was called Subhash [3]. These schemes are hardware oriented.

Boognish is a hash function published by the same authors as Cellhash [4]; it mixes the design principles of Cellhash with those of the MD4 variants.

One of the first requirements for a hash function is that its description should be concise and clear. This will make the hash function more attractive to evaluate and implement. We will leave it up to the reader to judge whether the proposed dedicated hash functions and their descriptions satisfy this (subjective) criterion.

Additionally one would like to have a motivation for all design decisions, and a comparison with alternatives that were rejected; a particularly good example of this is the description of Snefru. This will avoid the situation where the person who is evaluating the hash function has to duplicate a large part of the effort already done by the designer. The reverse of the medal is that the evaluator will loose part of this freshness when looking at the algorithm. Unfortunately some designer seem to be reluctant to give away know-how on designing and evaluating hash functions.

4 Security Relevant Criteria

The following security relevant properties will be discussed: should the round function be collision resistant and one-way or not, should the round function be partially bijective and one-way, how to protect beginning and end of the hashing operation, the use of redundancy in the message. Subsequently, a treatment of some specific attacks will be given.

4.1 Collision resistant and one-way.

The main issue faced by the designer is whether he will base his dedicated hash function on a round function that is both collision resistant and one-way or not. A first argument in favor of a collision resistant and one-way round function has been discussed in Sect. 2: if the construction satisfies certain restrictions, the security of the hash function can be studied by analyzing the round function. The advantage of this construction is also that the choice of a specific IV is not very important: the scheme should be secure with any IV. Indeed, if a perfectly secure compression function is used, finding a pseudo-preimage or a pseudo-collision is not easier than finding a preimage or a collision. Another advantage of this approach is that the size of the chaining variables can be increased at the cost of a decreased performance. This corresponds to a trade-off between security and speed. Moreover, the round function can be used directly in applications where the size of the input is fixed. Finally hashing can be done in a parallel way using a tree construction [6, 20].

On the other hand, from analyzing the hashing process, it follows that the roles of H_i and X_i are essentially different. This means that it seems natural to impose different conditions on both inputs of the round function. E.g., if finding a pseudo-preimage is relatively easy, one can find a preimage faster than 2^n operations, but this does not mean that finding a preimage becomes feasible. One can hope that loosening the requirements will improve the performance of the hash function.

An argument against this approach is that if this construction results in a collision resistant hash function, it will also give a collision resistant function (just fix the input size), which can be used as the round function in a construction of the first type. Note that this new hash function will be slower than the first one.

If the option is taken to design a collision resistant function, one should be consistent and design the round function such that it is symmetric with respect to message and chaining variable. Unfortunately, not all proposed schemes follow this logic.

Designs along these lines are MD2, FFT-hash, Snefru, and the Damgård scheme based on a cellular automaton. Designs that treat chaining variables and message blocks differently are N-hash, Cellhash, Subhash, Boognish, and MD4 and its variants. On the other hand, the only round functions for which no collisions were found are those of MD2, Snefru with more than 6 passes, MD4, SHA, and RIPE-MD (MD5 is excluded). This is rather surprising, as the MD4 variants are designs of the second type.

4.2 Bijectivity and one-wayness.

A second related issue is the question whether the round function or part of the round function should be bijective and one-way. On the one hand, bijectivity for part of the round function seems to be a desirable property: this avoids attacks where the entropy of the internal variables or chaining variables decreases, and one can show that at certain places no collision can occur. On the other hand, one would like to have a one-way mapping: if an attacker can go backwards easily, he will be able to use meet-in-the-middle techniques. In practice however it seems hard to combine both properties: it is a well known open problem to find a very efficient one-way permutation. Proving that a scheme is bijective is in most cases done by proving that one can go backwards. Examples based on number theoretic problems do exist (e.g., RSA [24], and a scheme based on elliptic curves [10]), but are rather slow. This seems to be an intrinsic problem, which is linked to the fact that it is very difficult to find efficient public-key cryptosystems. In order to get around this dilemma, many schemes combine a bijection with a feedforward of the data input or the chaining variable input or both. Of course this destroys the bijectivity. In the study of these properties, a distinction has to be made between schemes that treat chaining variables and message blocks symmetrically and schemes that do not have this property.

In the first case FFT-hash and Snefru are based on a bijective mapping, where the output size is reduced by chopping part of the bits (for FFT-hash and Snefru). In Snefru the round function is made one-way by exoring the input to the output. The security of the Damgård cellular automata scheme is based on the one-wayness of the round function. With respect to this criterion, MD2 behaves a bit differently: the mapping is one-way, but part of the internal state can be recovered.

The optimal configuration in this case seems to be a number of bijective mappings, interleaved with feedforwards in order to avoid a meet-in-the-middle attack. The security can also be increased by not simply selecting the output bits but applying a simple compression.

If message blocks and chaining variables are treated in a different way, the schemes that contain a mapping that is bijective if one of the inputs is fixed can be treated in the same way as single length hash functions based on a block cipher [22]. Cellhash, Subhash, and Boognish have a round function which is bijective for a fixed message and for a fixed chaining variable, but the mapping can be inverted easily if one of the inputs is fixed. It is equivalent to a CFB mode, but it derives its strength from the specific redundancy scheme in the message. MD4 and its variants are based on a function that is bijective for a fixed message block, while the one-way property is introduced by a feedforward of the previous chaining variable. The first variant of N-hash is bijective for a fixed chaining variable, and the other one is bijective for a fixed message block. In both variants the chaining variable and the message are added modulo 2 to the output of the function. From [22] it follows that fixed points can be found easily for MD4, N-hash, and their variants. The impact of these fixed points is however limited; they pose no threat if the padding scheme adds the message length.

4.3 Beginning and end.

It is well known that the beginning and the end of the hashing operation are critical for the security of a hash function. We repeat the main recommendations of Sect. 2:

- choose a single IV or a limited set of IV's,
- use an unambiguous padding rule that comprises the addition of the length of the message.

For some schemes it can be useful to impose a maximal message length.

Many hash functions are vulnerable to attacks where one goes backwards or to attacks that concentrate on the end of the message like correcting block attacks. Their security can be increased without reducing the performance by adding at the end a limited number of blocks that are a function of all message blocks. An example is the first hashcode that is used in MD2. Other solutions are the addition of zero blocks (Subhash, Boognish) or the use of cyclic redundancy in the message (Cellhash).

Adding a first block that is dependent on IV can have as effect that finding a pseudo-preimage is not easier than finding a preimage: this is especially interesting in case of schemes where going backwards is very easy, and the use of different IV's is still necessary.

Another principle that seems to increase the security is to work with two independent chains with a limited interaction in every round: in this way the security margin caused by the additional internal memory can "wipe out" certain weaknesses in the round function. This principle has been applied in RIPE-MD.

4.4 Redundancy in the message.

It seems to be a good design principle to use every message bit as many times as possible, and in such a way that it is hard to compensate changes in one message block. From this it follows that the same bit should be used at different locations and in different functions. This calls for a structure like MD4 where the message is used as "key" to "encrypt" in a reversible way the chaining variables. The "key scheduling" should guarantee that every key bit is used a sufficient number of times.

Instead of duplicating every message bit, the idea of using an error correcting code as in SHA is very promising: in this way one can guarantee that—independently of the selection of the message– the actual input to the hash function will differ in at least d locations, with d the minimum distance of the code.

There are several arguments to avoid making more than a single "pass" over the message: the performance decreases, it might be that additional storage is necessary, if the round function is invertible a generalized meet-in-the-middle attack may be applied [9], and finally in implementations one has to verify that it is indeed the same message that enters the hashing process.

4.5 Specific attacks.

A variety of attacks have to be considered when designing a dedicated hash function. In this respect a hash function is not very different from a stream cipher or a block cipher. Attacks that have to be studied are differential attacks, linear attacks, and analytical attacks. In case of a differential attack, a single pair that yields an output exor equal to zero is sufficient to break the hash function.

The first evaluation that has to be performed on an algorithm is certainly a statistical evaluation: it should be checked whether any irregularities can be found in the distribution of the output, and whether input differences affect the complete output. Special attention can be paid here to weaknesses occurring in most and least significant bits, to the relation between parity of inputs and outputs, and to other linearities.

Differential and linear cryptanalysis appear to be very powerful techniques, especially against schemes based on S-boxes [1, 13]. In case of hash functions, one will look for two inputs for which the output difference is zero, or for which the output difference is equal to the input difference (if a feedforward exists of the input to the output).

Fixed points are certainly not usable in practice, but in absence of better criteria, they can be used to discriminate between several schemes.

5 Efficiency

The discussion in this section is not specific for hash functions; the same criteria apply to block ciphers and stream ciphers.

In general a choice is made between software and hardware implementations. In software implementations, one will try to update variables sequentially, i.e., use the output of a calculation immediately in the next step. An important constraint is the limitation of the memory access, which exists at two levels. In the first place, one will try to keep as many variables as possible in the registers. Secondly, one will try to optimize the use of the cache memory. These considerations become more and more important as the access time to the memory seems to decrease more slowly than the cycle time of the processors. This suggests that faster hash functions will rather make use of logic and arithmetic operations available on a standard processor, than of S-boxes. However, the advantage of S-boxes is that they yield a strong nonlinear relation between input and output. Other less important aspects are word size, byte ordering, and problems with carries. Finally it should be remarked that one should not try to optimize the design towards a single processor: designing and reviewing a dedicated hash function will take several years, and by that time the processor will probably be outdated. On the other hand, it is acceptable to tune an algorithm, which uses common instructions, to be very fast on a recent processor; it is very likely to achieve a high speed on most other processors as well. The MD4 family is clearly optimized to be fast in software: these algorithms run at more than 10 Mbit/sec on present day computers, which makes them about one order of magnitude faster than other schemes that are not yet broken.

For hardware oriented hash functions, one will try to make use of parallelism. Nonlinearity in hardware can be generated efficiently by S-boxes. The diffusion can be increased by bit permutations, that simply correspond to wire crossings. Ideally, such a dedicated hash function should also take a very limited area: this will decrease the cost of the IC, and will make it possible in the near future to integrate the hash function as a basic component in other IC's. A design that consumes too much area will very likely go the same way as most ASIC (Application Specific IC) designs for RSA during the eighties: they were never built for economical reasons. The only dedicated hardware hash function seem to be Cellhash and its variant Subhash. The expected speed is about 1 Gbit/sec with a 33 MHz clock, but the area will probably be much larger than the area for a DES implementation.

In order to limit the evaluation effort, one might ask the question whether a single hash function can be designed that would offer a good compromise between hardware and software implementation criteria. In that case the design rule should be a compromise between software and hardware. A limited degree of parallelism is necessary in order to make hardware implementations efficient, and this will be advantageous as well on computer architectures with several processors. In order to make software implementations fast, permutations at bit level should be designed in a structured way (byte or word structure), or it should be possible to combine them together with the S-boxes, as the permutation P of the DES. Moreover S-boxes seem to be a promising component, as they are acceptable in both hardware and software. These S-boxes should be optimal rather than large and random, and should have a size of at most a few Kilobytes. They could also degenerate to parallel Boolean functions, which reduces the memory access; this corresponds to simple arithmetic (addition) and logic operations. In this way, one can achieve a speed of more than 10 Mbit/sec in software and more than 100 Mbit/sec in hardware (both estimates are for current standard technology), which is about twice as fast as the current DES implementations. So far, the only proposal for a hash function in this class is Boognish.

To conclude this section, Table 1 gives an overview of the speed of some hash functions in software. All timings were performed on a 16 MHz IBM PS/2 Model 80 with a 80386 processor. On a more recent PC with a 66 MHz 80486, these figures will be improved with almost one order of magnitude. The implementations were written by A. Bosselaers. Most of them use additional memory to improve the speed. The C-code was compiled with a 32-bit compiler in protected mode. In order to allow for a comparison with hash functions based on the DES, speed of a software implementation of the DES [7] is indicated, as well as the timings for a modular squaring and exponentiation with a short exponent. In this case a 512-bit modulus was chosen, and no use was made of the Chinese remainder theorem to speed up the computations. Some algorithms like Snefru and SHA would perform relatively better on a RISC processor, where the complete internal state can be stored in the registers. On this type of processor, SHA is only about 15% slower than MD5.

Table 1. Performance of several hash functions on an IBM PS/2 (16 MHz 80386).

type	hash function	C language (Kbit/sec)	Assembly language (Kbit/sec)
dedicated MDC	MD2	78	78
	MD4	2669	6273
	MD5	1849	4401
	SHA	710	1370
	RIPEMD	1334	3104
	N-hash	266	477
	FFT-hash I	212	304
	Snefru-6	358	358
	Snefru-8	270	270
block cipher	DES (+ key schedule)	130	200
	DES (fixed key)	512	660
modular arithmetic	squaring	50	273
	exponentiation $(2^{16}+1)$	1.8	14

6 Trapdoors in Hash Functions

It is very common to check proposed encryption functions for trapdoors. These are weaknesses that are built in the system on purpose, and that allow the person who knows of these weaknesses to break the algorithm much faster. This risk is certainly larger in case of proprietary algorithms, where the algorithm can only be reviewed by a limited number of persons. Although one would not expect that this issue comes up in the design of hash functions, it is clear that here also this problem arises. An obvious weakness would be the choice of the IV: one could select it in such a way that a particular plaintext block yields a fixed point.

The most obvious place to look for trapdoors are S-boxes. In order to avoid any allegations, designers tend to generate these S-boxes in a random way: the S-boxes from Snefru are derived from random numbers that were published in 1955, and the S-box from MD2 is derived from the digits of the number π. However, this solution is not completely effective: breaking the hash function might be more easy if a certain property is present. If this event has probability 10^{-9}, the designer can easily come up with 10^9 "straightforward" ways to generate the S-box from the public string, while the "random" permutation still has the required property. In some cases, the security of the scheme might be increased if certain properties are built into the S-boxes. It is then even harder to show that no other criteria were applied.

7 Conclusion

For the time being there is no agreement on the design principles for a dedicated hash function. Although some theory exists, several designs deviate from the

basic principles in order to improve the performance. In addition, using the available theory is no guarantee for a secure scheme. This paper attempted to describe the different approaches and to assess their merits. Hopefully it can give some guidance to future designers of hash functions.

References

1. E. Biham and A. Shamir, *"Differential Cryptanalysis of the Data Encryption Standard,"* Springer-Verlag, 1993.
2. J. Daemen, R. Govaerts, and J. Vandewalle, "A framework for the design of one-way hash functions including cryptanalysis of Damgård's one-way function based on a cellular automaton," *Advances in Cryptology, Proc. Asiacrypt'91, LNCS 739*, H. Imai, R.L. Rivest, and T. Matsumoto, Eds., Springer-Verlag, 1993, pp. 82–96.
3. J. Daemen, R. Govaerts, and J. Vandewalle, "A hardware design model for cryptographic algorithms," *Computer Security – ESORICS 92, Proc. Second European Symposium on Research in Computer Security, LNCS 648*, Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, Eds., Springer-Verlag, 1992, pp. 419–434.
4. J. Daemen, R. Govaerts, and J. Vandewalle, "Fast hashing both in hard- and software," *Presented at the Rump Session of Eurocrypt'93.*
5. I.B. Damgård, "Collision free hash functions and public key signature schemes," *Advances in Cryptology, Proc. Eurocrypt'87, LNCS 304*, D. Chaum and W.L. Price, Eds., Springer-Verlag, 1988, pp. 203–216.
6. I.B. Damgård, "A design principle for hash functions," *Advances in Cryptology, Proc. Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 416–427.
7. *"Data Encryption Standard,"* Federal Information Processing Standard (FIPS), Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977.
8. *"Secure Hash Standard,"* Federal Information Processing Standard (FIPS), Publication 180, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., 1993.
9. M. Girault, R. Cohen, and M. Campana, "A generalized birthday attack," *Advances in Cryptology, Proc. Eurocrypt'88, LNCS 330*, C.G. Günther, Ed., Springer-Verlag, 1988, pp. 129–156.
10. B.S. Kaliski, "One-way permutations on elliptic curves," *Journal of Cryptology*, Vol. 3, No. 1, 1991, pp. 187–199.
11. B.S. Kaliski, "The MD2 Message-Digest algorithm," *Request for Comments (RFC) 1319*, Internet Activities Board, Internet Privacy Task Force, April 1992.
12. X. Lai and J.L. Massey, "Hash functions based on block ciphers," *Advances in Cryptology, Proc. Eurocrypt'92, LNCS 658*, R.A. Rueppel, Ed., Springer-Verlag, 1993, pp. 55–70.
13. M. Matsui, "Linear cryptanalysis method for DES cipher," *Advances in Cryptology, Proc. Eurocrypt'93, LNCS*, Springer-Verlag, to appear.
14. R. Merkle, "A fast software one-way hash function," *Journal of Cryptology*, Vol. 3, No. 1, 1990, pp. 43 58.
15. S. Miyaguchi, M. Iwata, and K. Ohta, "New 128-bit hash function," *Proc. 4th International Joint Workshop on Computer Communications*, Tokyo, Japan, July 13–15, 1989, pp. 279–288.
16. S. Miyaguchi, "The FEAL cipher family," *Advances in Cryptology, Proc. Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 627–638.

17. S. Miyaguchi, K. Ohta, and M. Iwata, "128-bit hash function (N-hash)," *Proc. Securicom 1990*, pp. 127–137.
18. S. Miyaguchi, K. Ohta, and M. Iwata, "128-bit hash function (N-hash)," *NTT Review*, Vol. 2, No. 6, 1990, pp. 128–132.
19. M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," *Proc. 21st ACM Symposium on the Theory of Computing*, 1990, pp. 387–394.
20. B. Preneel, "Analysis and design of cryptographic hash functions," *Doctoral Dissertation*, Katholieke Universiteit Leuven, 1993.
21. B. Preneel, "Cryptographic hash functions," *Kluwer Academic Publishers*, 1994.
22. B. Preneel, R. Govaerts, and J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach," *Advances in Cryptology, Proc. Crypto'93, LNCS*, Springer-Verlag, to appear.
23. "Race Integrity Primitives Evaluation (RIPE): final report," RACE 1040, 1993.
24. R.L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications ACM*, Vol. 21, February 1978, pp. 120–126.
25. R.L. Rivest, "The MD4 message digest algorithm," *Advances in Cryptology, Proc. Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 303–311.
26. R.L. Rivest, "The MD4 message-digest algorithm," *Request for Comments (RFC) 1320*, Internet Activities Board, Internet Privacy Task Force, April 1992.
27. R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC) 1321*, Internet Activities Board, Internet Privacy Task Force, April 1992.
28. C.P. Schnorr, "An efficient cryptographic hash function," *Presented at the Rump Session of Crypto'91.*
29. C.P. Schnorr, "FFT-Hash II, efficient cryptographic hashing," *Advances in Cryptology, Proc. Eurocrypt'92, LNCS 658*, R.A. Rueppel, Ed., Springer-Verlag, 1993, pp. 45–54.
30. C.P. Schnorr and S. Vaudenay, "Parallel FFT-Hashing," *These Proceedings.*
31. G. Yuval, "How to swindle Rabin," *Cryptologia*, Vol. 3, 1979, pp. 187–189.
32. Y. Zheng, J. Pieprzyk, and J. Seberry, "HAVAL — a one-way hashing algorithm with variable length output," *Advances in Cryptology, Proc. Auscrypt'92, LNCS 718*, J. Seberry and Y. Zheng, Eds., Springer-Verlag, 1993, pp. 83–104.