

The State of Cryptographic Hash Functions

Bart Preneel*

Katholieke Universiteit Leuven, Dept. Electrical Engineering-ESAT
Kardinaal Mercierlaan 94, B-3001 Heverlee, Belgium

`bart.preneel@esat.kuleuven.ac.be`

Abstract. This paper describes the state of the art for cryptographic hash functions. Different definitions are compared, and the few theoretical results on hash functions are discussed. A brief overview is presented of the most important constructions, and some open problems are presented.

1 Introduction

Hash functions are well known in computer science. They compress a string of arbitrary length to a string of fixed length and are used to allocate as uniformly as possible storage for the records of a file. For cryptographic applications, additional security requirements are necessary: informally, one requires that they are hard to invert, and (in most cases) that it is hard to find colliding inputs. This is achieved by creating a mapping that associates to an input string a ‘randomly looking’ output string (note that formally this makes no sense, as hash functions are deterministic mappings). As a consequence of these properties, hash functions create a ‘unique’ relationship between the input and the hash value; there are of course many inputs corresponding to a single output, but it is hard to identify these.

Cryptographic hash functions can be used to protect the integrity of large amounts of information (such as the content of a hard disk, a set of financial transactions, a software program) by the integrity of a short string, the hash result. This protection can be achieved by digitally signing this short string [31], or by writing down the string on a piece of paper that is stored in a secure place. This is analogous to conventional message encryption, that replaces the secrecy of a large amount of information by the secrecy (and authenticity) of a secret key; typically this key is much shorter than the message.

Hash functions have been used (and sometimes abused) for many other cryptographic applications. Examples include

- the protection of pass-phrases (where the image of the pass-phrase under the hash function is stored in the computer, rather than the pass-phrase itself);
- the commitment to a string without revealing it (see Damgård *et al.* [27]).

* F.W.O. postdoctoral researcher, sponsored by the Fund for Scientific Research – Flanders (Belgium).

- the construction of Message Authentication Codes (MACs), by introducing in the hash function a second parameter that is kept secret [5, 7, 70];
- key derivation, for example to derive session keys from a transaction number and a master key.

The last two applications assume that the hash function keyed by a second parameter yields a pseudo-random function, or a mapping that is hard to predict. Hash functions have also been used to instantiate ‘random oracles’ [9]; this requires even stronger properties.

The remainder of this paper is organized as follows. Section 2 presents the definitions and Sect. 3 discusses generic constructions and security results. Specific constructions are treated in Sect. 4, while conclusions and open problems are presented in Sect. 5.

2 Definitions

One distinguishes between hash functions that have only a single input, and hash functions that have a second input. The first type are sometimes called MDCs or Manipulation Detection Codes; many authors refer to this type simply as cryptographic hash functions, or even hash functions. If the second parameter is secret, one calls these functions keyed hash functions; an important subclass are MACs or Message Authentication Codes. Finally the second parameter can also be public; examples in this class are the UOWHFs or Universal One-Way Hash Functions.

Note that one should not confuse hash functions with checksums that are used for error detection or correction (such as the well known Cyclic Redundancy Checks or CRCs).

First we define one-way hash functions (OWHF) and collision resistant hash functions (CRHF). Both classes of hash functions are studied in this paper. Then we look at some related concepts, namely message authentication codes (MACs), universal one-way hash functions (UOWHFs) and universal hash functions.

In the following the hash function will be denoted with h , and its argument, i.e., the information to be protected with x . The image of x under the hash function h will be denoted with $h(x)$. It will be assumed that the description of the hash function h is publicly known, and that it does not require any secret information (except for the optional parameter, which may be secret). A second assumption is that given the inputs, the computation of the hash result must be “easy.”

2.1 One-Way Hash Function (OWHF)

The concept of one-way hash functions was introduced by Diffie and Hellman in [31]. The first informal definition was apparently given by Merkle [59, 60] and Rabin [73]. A **one-way hash function** is a function h satisfying the following conditions:

Appeared in *Lectures on Data Security*, Lecture Notes in Computer Science 1561,
I. Damgård (ed.), pp. 158–182, 1999.
©1999 Springer-Verlag

1. The argument x can be of *arbitrary* length and the result $h(x)$ has a *fixed* length of n bits (with $n \geq 64 \dots 80$).
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 $h(x) = y$ (preimage resistant) and given x in the domain of h and $h(x)$ it is “hard” to find a message $x' \neq x$ such that $h(x') = h(x)$ (second preimage resistant).

Note that this last condition (finding a second preimage is “hard”) differs from the intuitive concept of one-wayness, namely that it is “hard” to find a preimage x given only h and the value of $h(x)$. It is clear that for permutations or injective functions only preimage resistance is relevant. The relation between preimage resistance and second preimage resistance is discussed in [58, 67].

The above definition is only informal. For example, one should specify the distribution that is used to select y respectively x , and specify what “hard” and “easy” means. The definition should also take into account that one can always invert functions with a small range (by exhaustive search) and that one can always precompute the function in a small set. There are many ways to formalize the definition, and it is a non-trivial exercise to show which of these are equivalent.

For a formal definition, we need to specify a model of computation. Rather than probabilistic Turing machines (that are used traditionally in cryptography), we will follow here Bellare and Rogaway [10] and use the RAM model including pointers (see for example [22]); execution time is measured with respect to that model. An *adversary* is a program for this model, written in some fixed programming language. The adversary has access to random bits.¹ The running time includes the actual execution time and the length of the program description.

The set of all integers will be denoted with \mathbf{N} . The alphabet considered is the binary alphabet $\Sigma = \{0, 1\}$. For $n \in \mathbf{N}$, Σ^n is the set of all binary strings of length n . The set of all strings of arbitrary length will be written as Σ^* . The size of a set S is denoted with $|S|$. Let h be a function with domain $D = \Sigma^*$ and range $R = \Sigma^n$. Note that in fact we will only consider inputs of bit length $l(n)$, with $l(n)$ a function that satisfies $l(n) > n$. This is not a real restriction, since it is not possible (in practice) to evaluate a function in inputs that are too large.

Definition 1. A **one-way hash function** H is a function with domain $D = \Sigma^{l(n)}$ and range $R = \Sigma^n$ that satisfies the following conditions:

- *preimage resistance*: let x be selected uniformly in D and let M be an adversary that on input $h(x)$ uses time $\leq t$ and outputs $M(h(x)) \in D$. For each adversary M ,

$$\Pr_{x \in D} \{h(M(h(x))) = h(x)\} < \epsilon.$$

Here the probability is also taken over the random choices of M .

¹ Returning a random integer in the interval $[1, n]$ takes time $\Theta(\log n)$.

- 2nd preimage resistance: let x be selected uniformly in $\Sigma^{l(n)}$ and let M' be an adversary that on input x uses time $\leq t$ and outputs $x' \in D$ with $x' \neq x$. For each adversary M' ,

$$\Pr_{x \in D} \{M'(x) = h(x)\} < \epsilon.$$

Here the probability is also taken over the random choices of M' .

The above definition is only meaningful if t/ϵ is large; it is clear that $t/\epsilon \leq 2^n$. One can specify a constant as required by a given application.

Note that this model does not take into account precomputation, which can be used to speed the computation of many preimages (as for example in Hellman [49]).

2.2 Collision Resistant Hash Function (CRHF)

The fact that a function is one-way does not mean that it is hard to find two colliding inputs; it will be shown in Sect. 3.2 that the effort to find a collision is only the square root of the effort to find a (2nd) preimage. This motivates the definition of a collision resistant function as a separate primitive. Note that some authors call this a collision free hash function [24–26], or a collision intractible hash function [89].

The first formal definition of a CRHF was given by Damgård [24, 25]. An informal definition was given by Merkle in [60]. 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 $n \geq 128 \dots 160$).
2. The hash function must be *one-way*, i.e., preimage resistance and second preimage resistant.
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.

Finding a second preimage cannot be easier than finding a collision: therefore the second preimage condition in this definition is redundant. However, preimage resistance is not necessarily implied by collision resistance (note that it is required for certain applications). Damgård provides some definitions and conditions under which collision resistance implies preimage resistance [26]; see also Gibson’s comment in [42].

Formalizing a collision resistant hash function is not as straightforward as formalizing a one-way hash function. One cannot specify that there should not exist an adversary that outputs a collision: since there are many colliding inputs that are very short, there will be many efficient adversaries that can output a pair of inputs that collide under x (just include two such inputs in the code). The way out of this problem is to define collision resistance as the property of a family \mathcal{H} of hash functions, that is, a set of functions indexed by a parameter S . For simplicity, it will be assumed that S is taken from the parameter space Σ^s . Thus \mathcal{H} is a mapping from $D \times \Sigma^s \rightarrow R$, and an individual function in this family is denoted with $h_S : D \rightarrow R$.

Definition 2. A **collision-resistant hash function** \mathcal{H} is a function family with domain $D = \Sigma^{l(n)}$ and range $R = \Sigma^n$ that satisfies the following conditions:

- the functions h_S are preimage resistant and second preimage resistant (cf. Definition 1).
- collision resistance: let F be a collision string finder that on input $S \in \Sigma^s$ uses time $\leq t$ and outputs either “?” or a pair $x, x' \in \Sigma^{l(n)}$ with $x' \neq x$ such that $h_S(x') = h_S(x)$. For each F ,

$$\Pr_S \{F(\mathcal{H}) \neq \text{“?”}\} < \epsilon.$$

Here the probability is also taken over the random choices of F .

Again this definition is only meaningful if t/ϵ is large; it will follow from Sect. 3.2 that an upper bound is $2.24 \cdot 2^{n/2}$.

2.3 Universal One-Way Hash Function

The concept of a UOWHF was introduced by Naor and Yung [64]. In [10], Bellare and Rogaway give a concrete definition (rather than an asymptotic one) and study practical constructions for this primitive under the name ‘target collision resistant hash functions.’

Definition 3. A **universal one-way hash function** \mathcal{H} is a function family with domain $D = \Sigma^{l(n)}$ and range $R = \Sigma^n$, that satisfies the following condition:

- Let $F' = (F'_1, F'_2)$ be an adversary. F'_1 is run first; it is an algorithm that produces x and possibly some state information; this information is passed on to F'_2 . Algorithm F'_2 is given S , x and the state information and outputs either “?” or an $x' \neq x$ such that $h_S(x') = h_S(x)$. For each F' that runs in time $\leq t$ (that is, the sum of the running times of F'_1 and F'_2),

$$\Pr_S \{F'(\mathcal{H}) \neq \text{“?”}\} < \epsilon.$$

Here the probability is also taken over the random choices of F' .

The main difference with collision resistance is that here the input x is fixed first, and then the parameter S is chosen. This implies that finding collisions for a given value of S does not help an attacker.

Naor and Yung show that a UOWHF can be used to build a digital signature scheme [64]. Rompel [78] developed a (very inefficient) construction for a UOWHF based on any one-way function; in this way he reduces a digital signature scheme to a one-way function, which is the weakest possible assumption. A UOWHF can replace a CRHF in a digital signature scheme if the signer does not intend to repudiate her signatures (see also Sect. 3.2).

2.4 Message Authentication Code (MAC)

Message Authentication Codes have been used for a long time in the banking community. However, MAC algorithms with good cryptographic properties were only introduced in the late 1970s. The first reference to a MAC algorithm is a 1972 patent application by Simmons *et al.* (reference 10. in [81]).

Definition 4. A MAC algorithm is a function h satisfying the following conditions:

1. The argument x can be of arbitrary length and the result $h_K(x)$ has a fixed length of n bits (with $n \geq 32 \dots 64$).
2. Given h and x , it is “hard” to forge a MAC on a new message, that is, to determine $h_K(x)$ with a probability of success “significantly higher” than $1/2^n$. Even when many pairs $\{x_i, h_K(x_i)\}$ are known, where the x_i have been selected (sequentially) by the opponent, it is “hard” to compute $h_K(x')$ for any $x' \neq x_i$.

The last attack is called an adaptive chosen text attack. One distinguishes between forgery attacks and key recovery attacks: a forgery allows to determine the MAC on a new message; a key recovery attack is much more serious as it allows to forge the MAC for an arbitrary message.

The exact security of a MAC algorithm can be expressed in terms of the running time t of the adversary, the number of known and (adaptively) chosen texts an adversary has access to, and the probability of success ϵ of the forgery attack (see e.g., [7]).

A MAC algorithm can be used for message authentication between a sender and a receiver who share a secret key K . In order to protect a message, the sender applies the MAC algorithm to the message and appends the resulting string to the message. On receipt of the message, the receiver recomputes the MAC and verifies that it corresponds to the transmitted MAC value. An active eavesdropper Eve can modify the message, but as she does not know the secret key, she cannot predict the MAC value for the modified message.

2.5 Universal Hash Functions

Universal hash functions are combinatorial objects, which implies that they can be defined without using a model of computation. They were introduced by Carter and Wegman [15, 86], who show that they can be applied to efficient unconditionally secure message authentication. In this way they found practical constructions for the authentication codes introduced by Simmons in the 1970s [80]. The first published reference to authentication codes is Gilbert *et al.* [44]. Other applications of universal hash functions include interactive proof systems, pseudo-random number generation, complexity theory, and probabilistic algorithms.

A universal hash function is a mapping from a finite set A with size a to a finite set B with size b . For a given hash function h and for a pair (x, x') with

$x \neq x'$ the following function is defined:

$$\delta_h(x, x') = \begin{cases} 1 & \text{if } h(x) = h(x') \\ 0 & \text{otherwise.} \end{cases}$$

As above, a finite set of hash functions will be denoted *family* \mathcal{H} of hash functions. Now $\delta_{\mathcal{H}}(x, x')$ is defined as $\sum_{h \in \mathcal{H}} \delta_h(x, x')$, or $\delta_{\mathcal{H}}(x, x')$ counts the number of functions in \mathcal{H} for which x and x' collide. If a random choice of h is made, then for any two distinct inputs x and x' , the probability that these two inputs yield a collision equals $\delta_{\mathcal{H}}(x, x')/|\mathcal{H}|$. For a universal hash function, the goal is to minimize this probability together with the size of \mathcal{H} .

Definition 5. *Let ϵ be any positive real number. An ϵ -almost universal family (or ϵ -AU family) \mathcal{H} of hash functions from a set A to a set B is a family of functions from A to B such that for any distinct elements $x, x' \in A$*

$$|\{h \in \mathcal{H} : h(x) = h(x')\}| = \delta_{\mathcal{H}}(x, x') \leq \epsilon \cdot |\mathcal{H}| .$$

This definition states that for any two distinct inputs the probability for a collision is at most ϵ . In [15] the case $\epsilon = 1/b$ is called universal (the smallest possible value for ϵ is $(a - b)/b(a - 1)$).

Definition 6. *Let ϵ be any positive real number. An ϵ -almost strongly universal family (or ϵ -ASU family) \mathcal{H} of hash functions from a set A to a set B is a family of functions from A to B such that*

- for every $x \in A$ and for every $y \in B$, $|\{h \in \mathcal{H} : h(x) = y\}| = |\mathcal{H}|/b$,
- for every $x_1, x_2 \in A$ ($x_1 \neq x_2$) and for every $y_1, y_2 \in B$ ($y_1 \neq y_2$), $|\{h \in \mathcal{H} : h(x_1) = y_1, h(x_2) = y_2\}| \leq \epsilon \cdot |\mathcal{H}|/b$.

The first condition states that the probability that a given input x is mapped to a given output y equals $1/b$. The second condition implies that if x_1 is mapped to y_1 , then the conditional probability that x_2 (different from x_1) is mapped to y_2 is upper bounded by ϵ . The lowest possible value for ϵ equals $1/b$ and this family has been called strongly universal functions in [86]. Stinson shows that for this family the first condition in the definition follows from the second one [83].

An ϵ -almost strongly universal hash function family can be used in a similar way as a MAC algorithm. The secret key K chooses a function in the family; unlike the key for a MAC algorithm, the key can serve to authenticate a single message only. An ϵ -almost universal hash function family can also provide message authentication, but in addition its result needs to be encrypted (for example using a one-time pad).

3 Generic Constructions and Attacks

First a general model is presented for iterated hash functions. Next generic attacks are described, that is, attacks that are independent of the specific details of the hash function. This section is concluded with a discussion of generic security results for hash functions.

3.1 A General Model for Iterated Hash Functions

Most known hash functions are based on a compression function with fixed size inputs; they process every message block in a similar way. Lai and Massey call this an “iterated” hash function [55]. The information is divided into t blocks x_1 through x_t . If the total number of bits is not a multiple of the block length, the information is padded to the required length (using a so-called *padding rule*). The hash function can then be described as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(x_i, H_{i-1}) \quad i = 1, 2, \dots, t \\ h(x) &= g(H_t). \end{aligned}$$

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* or *compression function*, and the function g is called the *output transformation*. It is often omitted (that is, g is often the identity 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 do not exist two messages that can be padded to the same padded message); at the end one should append the length of the message; and the IV should be defined as part of the description of the hash function (this is called MD-strengthening after Merkle and Damgård). 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 alternative model is a tree structure, that allows for increased parallelism and may result in different security conditions for the round function (see also Sect. 3.3). For the time being it is rarely used in practice.

The general model for MAC algorithms is similar to that of MDCs; the use of an output transformation is more common here. Bellare and Rogaway [10] show that the above model does not work for a UOWHF, and they introduce a different approach. Earlier Naor and Yung developed a different iterated construction for a UOWHF in [64].

3.2 Generic Attacks

This section gives an overview of the known general attacks methods on MDCs. A first class of attacks depend only on the size of the parameters, and not on the specific hash function. The second class depends on the properties of the round function f .

This taxonomy can be helpful to understand the security results presented in Sect. 3.3, but can also serve as a *caveat* for designers and users of hash functions.

Attacks Independent of the Algorithm These attacks depend only on the size n in bits of the hash result, and are independent of the specific details of

the algorithm. It is assumed that the MDC approximates a random function: if this is not the case this class of attacks will be even more successful. For the time being 2^{64} operations is considered to be on the edge of feasibility. In view of the fact that the speed of computers is multiplied by four every three years (this is one of the formulations of Moore's law), 2^{70} operations is sufficient for the next 5 to 10 years, but it will be only marginally secure within 15 years. For applications that require protection for 20 years, one should try to design the hash function such that an attack requires at least 2^{80} operations.

Random (2nd) Preimage Attack. The opponent selects a random message and hopes that a given hash result will be hit. If the hash function has the required "random" behavior, his probability of success equals $1/2^n$ with n the number of bits of the hash result. This attack can be carried out off-line and in parallel, which means that n should be at least 64.

If a significant number of messages can be attacked simultaneously, it is advisable to select a larger value of n . In that case it is preferable to use a UOWHF rather than a simple OWHF: as every instance will have a different parameter, this prevents an attacker from amortizing his effort over many targets.

Birthday Attack. The birthday paradox states that for a group of 23 people, the probability that at least two people have a common birthday exceeds $1/2$. Intuitively one expects that the probability is much lower. However, the number of pairs of people in such a group equals $23 \cdot 22/2 = 253$. This can be exploited to find collisions for a hash function as follows: an adversary generates r_1 variations on a bogus message and r_2 variations on a genuine message. The expected number of collisions equals $r_1 \cdot r_2/n$. The probability of finding a bogus message and a genuine message that hash to the same result is given by $1 - \exp(-r_1 \cdot r_2/2^n)$, which is about 63% when $r = r_1 = r_2 = 2^{\frac{n}{2}}$. Finding the collision does not require r^2 operations: after sorting the data, which requires $O(r \log r)$ operations, comparison is easy. This attack was first pointed out by Yuval [87].

One can reduce the memory requirements for collision search by translating the problem to the detection of a cycle in an iterated mapping. Indeed, any mapping that is iterated on a finite set will eventually repeat, i.e., it will enter a cycle. If the mapping is a random mapping (rather than a permutation), the entry point to the cycle corresponds to a collision for the function (this algorithm fails if the starting point belongs to the cycle, but this event has a negligible probability). The detection of a cycle does not require storing all the values; for example, the technique of distinguished points can be used (one only stores special points, for example those points beginning with 30 zero bits). Cycle detection techniques were first applied to collision search by Quisquater [71]. The expected number of function evaluations of his algorithm is $2\sqrt{\pi/2} \cdot 2^{\frac{n}{2}}$; the storage requirements are negligible. In [85], van Oorschot and Wiener propose an efficient parallel variant of this algorithm: the speed-up is linear with the number of processors. They estimate that with a 10 million US\$ machine, collisions for MD5 (with $n = 128$) can be found in 21 days (in 1994). In order to make a collision search infeasible, n should be at least 160 bits.

For digital signatures, hash functions need to be collision resistant since otherwise one can sign one message and later claim to have signed a different message, or be held accountable for a different message. There is no way to prevent a sender from performing this attack, although the occurrence of two messages that hash to the same value might make him suspect. Outsiders can perform the same attack if they can convince the signer to sign a message of their choice. The sender can protect himself through randomizing the message just prior to signing (or by randomizing the hash function as is done for a UOWHF, cf. Sect. 2.3).

Attacks Dependent on the Chaining This class of attacks depends on some high level properties of the compression function f .

Meet-in-the-Middle Attack. This attack is a variation on the birthday attack, but instead of comparing the hash results, one compares intermediate chaining variables. The attack enables an opponent to construct a (2nd) preimage, which is not possible for a simple birthday attack. The opponent generates r_1 variations on the first part of a bogus message and r_2 variations on the last part. Starting from the initial value and going backwards from the hash result, the probability for a matching intermediate variable is again given by $1 - \exp(-r_1 \cdot r_2 / 2^n)$. The only restriction that applies to the meeting point is that it cannot be the first or last value of the chaining variable. The cycle finding algorithm has been extended by Quisquater to perform a meet-in-the-middle attack with negligible storage [72]. The attack can be precluded by avoiding functions f that are invertible to the chaining variable H_{i-1} and to the message x_i (see also Theorem 1 in Sect. 3.3).

Further extensions of this attack have been proposed by Coppersmith [19] and Girault *et al.* [46] to break p -fold iterated schemes, i.e., weak schemes with more than one ‘pass’ over the message as proposed by Davies [28]. Other extensions take into account additional constraints on the message.

Fixed Point Attack. The idea of this attack is to look for an H_{i-1} and x_i such that $f(x_i, H_{i-1}) = H_{i-1}$. If the chaining variable is equal to H_{i-1} , it is possible to insert an arbitrary number of blocks equal to x_i without modifying the hash result. Producing collisions or a second preimage with this attack is only possible if the chaining variable can be made equal to H_{i-1} : this is the case if IV can be chosen equal to a specific value, or if a large number of fixed points can be constructed (e.g., if one can find an x_i for a significant fraction of H_{i-1} ’s). Of course this attack can be extended to fixed points that occur after more than one iteration. This attack can be made more difficult by appending a block count and by fixing IV (MD-strengthening, see Sect. 3.1).

3.3 General Security Results

Research on hash functions has focussed on the question: which properties should be imposed on f to guarantee that h satisfies certain properties? Two partial

answers have been found to this question. The first result by Lai and Massey [55] gives necessary and sufficient conditions for f in order to obtain an “*ideally secure*” hash function h , that is, a hash function for which finding a preimage takes time $\Theta(2^n)$.

Theorem 1 (Lai–Massey). *Assume that the padding contains the length of the input string, and that the message x (without padding) contains at least 2 blocks. Then finding a second preimage for h with a fixed IV requires 2^n operations if and only if finding a second preimage for f with arbitrarily chosen H_{i-1} requires 2^n operations.*

Necessity of the condition is based on the following argument: if one can find a second preimage for f in 2^s operations (with $s < n$), one can find a second preimage for h in $2^{(n+s)/2+1}$ operations with a meet-in-the-middle attack (cf. Sect. 3.2).

A second result by Damgård [26] and independently by Merkle [60] states that for h to be a CRHF it is sufficient that f is a collision resistant function.

Theorem 2 (Damgård–Merkle). *Let f be a collision resistant function mapping l to n bits (with $l - n > 1$). If an unambiguous padding rule is used, the following construction yields a CRHF:*

$$\begin{aligned} H_1 &= f(0^{n+1} \parallel x_1) \\ H_i &= f(H_{i-1} \parallel 1 \parallel x_i) \quad \text{for } i = 2, 3, \dots, t. \end{aligned}$$

Here \parallel denotes the concatenation of binary strings. The construction can be improved slightly,² and extended to the case where $l = n + 1$, at the cost of an additional assumption on f (see [26] for details and Gibson’s comment in [42]). It can also be extended to a tree construction, which allows for increased parallelism [26].

We conclude this section with two other general results on the theory of hash functions. Damgård has showed in [25] that a collision resistant hash function can be constructed if claw-free permutations exist; Russell has slightly weakened this requirement to the existence of claw-free pseudo-permutations [79] (pseudo-permutations are functions that cannot be distinguished from permutations). Recently Simon [82] has provided a motivation to treat collision resistant hash functions as independent cryptographic primitives. He showed that no provable construction of a CRHF exists based on a “black box” one-way permutation, i.e., a one-way permutation treated as an oracle.

4 An Overview of Constructions

This section briefly discusses three types of MDCs: MDCs based on a block cipher, MDCs based on algebraic structures (modular arithmetic, knapsack, and lattice problems), and custom designed MDCs. For a more detailed discussion, the reader is referred to [67, 68].

² One can get rid of the extra “0” and “1” bits.

4.1 MDCs Based on a Block Cipher

Several arguments can be given for designers of hash functions to base their schemes on existing encryption algorithms. The first argument is purely historical: DES [37] was the first standard commercial cryptographic primitive that was widely available; it seemed natural to construct hash functions based on this block cipher. A second argument is the minimization of the design and implementation effort: hash functions and block ciphers that are both efficient and secure are hard to design, and many examples that support this view can be found in the literature. Moreover, existing software and hardware implementations can be reused, which will decrease the cost. The major advantage however is that the trust in existing encryption algorithms can be transferred to a hash function. The main disadvantage of this approach is that custom designed hash functions are likely to be more efficient. This is particularly true because hash functions based on block ciphers require a key change after every encryption. Finally note that block ciphers may exhibit some weaknesses that are only important if they are used in a hashing mode. One also has to take into account that in some countries export restrictions for block ciphers may be tighter than those for hash functions.

The encryption operation E will be written as $y = E_K(x)$. Here x denotes the plaintext, y the ciphertext, and K the key. The size of the plaintext and ciphertext or the block length (in bits) will be denoted with r , while the key size (in bits) will be denoted with k . In the case of the well known block cipher DES, $r = 64$ and $k = 56$ [37]. The **hash rate** of a hash function based on a block cipher is defined as the number of r -bit input blocks that can be processed with a single encryption.

A distinction will be made between the cases $n = r$, $n = 2r$, and $n > 2r$. This is motivated by the fact that most proposed block ciphers have a block length of only 64 bits, and hence an MDC with a result at least twice the block length is necessary to obtain a CRHF. Other proposals are based on a block cipher with a large key, or on a block cipher with a modified key schedule.

Size of Hash Result Equal to the Block Length. It follows from Sect. 3.2 that these hash functions can only be collision resistant if the block length r is at least 128 bits to 160 bits. Most present day block ciphers have only a block length of 64 bits, but the AES (Advanced Encryption Standard), which NIST intends to publish by 2001, will have a block length of 128 bits.

All schemes of this type proposed in the literature have rate 1. The first ‘secure’ construction for such a hash function was the ’85 scheme by Matyas *et al.* [57]:

$$H_i = E_{s(H_{i-1})}^{\oplus}(x_i) \stackrel{\text{def}}{=} E_{s(H_{i-1})}(x_i) \oplus x_i .$$

Here $s()$ is a mapping from the ciphertext space to the key space. This scheme has been included in ISO/IEC 10118-2 [51], and it forms the main building block for other hash functions based on block ciphers. This general construction is also used in several custom designed hash functions (cf. Sect. 4.3).

It is widely believed that this mapping is hard to invert, but there is no proof of this. It is not even clear which assumptions have to be imposed on the block cipher to allow for such a proof. One can apply the following intuitive reasoning: either one chooses the plaintext x , but then one has to find the key corresponding to one plaintext/ciphertext pair, which is deemed to be infeasible; alternatively, one chooses the key, but then one has to find for a given key a plaintext/ciphertext pair with a known difference, which is also believed to be difficult. Therefore it is conjectured that if the block cipher is ‘ideal’ (i.e., a keyed random one-way permutation) no shortcut attacks exist, which implies that a collision attack requires $\Theta(2^{r/2})$ operations and a (2nd) preimage attack $\Theta(2^r)$ operations.

Preneel *et al.* show that 12 variants exist with a similar security level; they can be obtained by applying an affine transformation to the inputs of two basic schemes [69]. Moreover, the security level of these hash functions is limited by $\min(k, r)$, even if the size of some internal variables is equal to $\max(k, r)$. One such variant is widely known as the Davies-Meyer scheme (the real inventors are probably Matyas and Meyer):

$$H_i = E_{x_i}^{\oplus}(H_{i-1}).$$

It has the advantage that it extends more easily to block ciphers where key size and block size are different.

Size of Hash Result Equal to Twice the Block Length. The goal of *double block length* hash functions is to achieve a higher security level against collision attacks. Ideally a collision attack on such a hash function should require 2^r operations, and a (2nd) preimage attack 2^{2r} operations.

A series of proposals attempted to double the size of the hash result, for example by iterating a OWHF; all of these succumbed to a ‘divide and conquer’ attack. A large class of proposals of rate 1 has the following form:

$$\begin{aligned} H_i^1 &= E_{A_i^1}(B_i^1) \oplus C_i^1 \\ H_i^2 &= E_{A_i^2}(B_i^2) \oplus C_i^2, \end{aligned}$$

where A_i^1 , B_i^1 , and C_i^1 are binary linear combinations of H_{i-1}^1 , H_{i-1}^2 , x_i^1 , and x_i^2 and where A_i^2 , B_i^2 , and C_i^2 are binary linear combinations of H_{i-1}^1 , H_{i-1}^2 , x_i^1 , x_i^2 , and H_i^1 . The hash result is equal to the concatenation of H_i^1 and H_i^2 . Knudsen *et al.* showed that for all hash functions in this class, a preimage attack requires at most 2^r operations, and a collision attack requires at most $2^{3r/4}$ operations (for most schemes this can be reduced to $2^{r/2}$) [53].

The few proposals that survive till today have rate less than 1. Two important examples are MDC-2 and MDC-4 with hash rate 1/2 and 1/4 respectively. They have been designed by Bracht *et al.* [13], and are also known as the Meyer-Schilling hash functions after the authors of the first paper describing these schemes [63]. MDC-2 has been included in ISO/IEC 10118-2 [51] (in a more

general form); it can be described as follows:

$$\begin{aligned} T_i^1 &= E_{u(H_{i-1}^1)}^\oplus(x_i) = LT_i^1 \parallel RT_i^1 & H_i^1 &= LT_i^1 \parallel RT_i^2 \\ T_i^2 &= E_{v(H_{i-1}^2)}^\oplus(x_i) = LT_i^2 \parallel RT_i^2 & H_i^2 &= LT_i^2 \parallel RT_i^1. \end{aligned}$$

Here the variables H_0^1 and H_0^2 are initialized with the values IV_1 and IV_2 respectively, and the hash result is equal to the concatenation of H_t^1 and H_t^2 . The ISO/IEC standard does not specify the block cipher; it also requires the specification of two mappings u, v from the ciphertext space to the key space such that $u(IV^1) \neq v(IV^2)$. The best known preimage and collision attacks on MDC-2 require 2^{3r} and 2^r operations respectively (Lai and Massey [55]). However, it is obvious that the compression function of MDC-2 is rather weak: preimage and collision attacks on the compression function require at most 2^r and $2^{r/2}$ operations (one fixes x_i and varies H_{i-1}^1 and H_{i-1}^2 independently).

One iteration of MDC-4 consists of the concatenation of two MDC-2 steps, where the plaintexts in the second step are equal to H_{2i-1}^2 and H_{1i-1}^1 . The rate of MDC-4 is equal to $1/4$. The best known attack to find a preimage for MDC-4 requires 2^{2r} operations. This shows that MDC-4 is probably more secure than MDC-2 against preimage attacks. However, a collision for the compression function of MDC-2 with a specified value for H_{i-1}^1 and H_{i-1}^2 also yields a collision for the compression function of MDC-4. Moreover, it has been demonstrated in by Preneel and Knudsen [67, 54] that collisions for the compression function of MDC-4 can be found with $2^{3r/4}$ encryptions and the storage of $2^{3r/4}$ r -bit quantities.

Merkle describes an interesting proposal in [60], for which he proves that the compression function is collision resistant based on the assumption that the underlying single block length scheme is secure. The simplest scheme (with rate $1/18.3$ for DES) can be described as follows:

$$H_i = \text{chop}_{16} \left[E_{0 \parallel H_{i-1}^1}^\oplus(H_{i-1}^2 \parallel x_i) \parallel E_{1 \parallel H_{i-1}^1}^\oplus(H_{i-1}^2 \parallel x_i) \right].$$

Here H_{i-1} is a string consisting of 112 bits, the leftmost 55 bits of which are denoted H_{i-1}^1 , and the remaining 57 are denoted H_{i-1}^2 ; x_i consists of 7 bits only. The function chop_r drops the r rightmost bits of its argument. Note that this construction is similar to MDC-2 (but much slower). The most efficient proposal is more complex and use six invocations of the block cipher in two layers. Its hash rate is equal to 0.27 for DES. Merkle's proof for this proposal only showed a security level of $2^{52.5}$ against collisions; Preneel has improved this to 2^{56} [67].

Even the schemes in this class that provide optimal security do not offer long term collision resistance when used with DES; this will change with AES, which will have a block and key length of 128 bits (key lengths of 192 and 256 bits will also be provided).

Size of Hash Result Larger Than Twice the Block Length. Knudsen and Preneel also design a collision resistant compression function, but with parallel encryptions only [54]. They show how a class of efficient constructions for

hash functions can be obtained by using non-binary error-correcting codes. Their schemes can achieve a provable security level against collisions equal to 2^r , $2^{3r/2}$ (or more) and this with rates larger than $1/2$; the security proof reduces the security of this scheme to an assumption on the single block length hash functions. The internal memory of the scheme is however much larger than 2 or 3 blocks, which implies that an output transformation is required.

Other constructions. Extending earlier work by Merkle [59], Lai and Massey [55] propose constructions for block ciphers with a key twice as long as the block length (Tandem Davies-Meyer and Abreast Davies-Meyer). Both schemes have rate equal to $1/2$; the best known attacks for a preimage and a collision requires 2^{2r} respectively 2^r encryptions. Faster schemes in this class have been developed in [54].

Aiello and Venkatesan propose in [1] a construction to double the output of a random function. In order for it to be usable for hashing, one needs to define the key schedule of this larger ‘block cipher’. The construction by Aiello, Haber, and Venkatesan [2] replaces the key schedule of DES by a function from the MDx-family (cf. Sect. 4.3); several instances are combined by choosing different (fixed) plaintexts.

4.2 MDCs Based on Algebraic Structures

First hash functions based on modular arithmetic are considered. Next hash functions based on knapsack problems and lattices are presented. This section is concluded with a short discussion of incremental hash functions.

MDCs Based on Modular Arithmetic. These hash functions are designed to use the modular arithmetic hardware that is required to produce digital signatures (for example, RSA [77]). The security of certain constructions can be based on the hardness of some number theoretic problems. Moreover these schemes are easily scalable. The disadvantage is that the underlying primitive has a rich mathematical structure; this can be exploited in attacks that use the homomorphic structure and the fixed points of modular exponentiation (trivial examples are 0 and 1); one also has to ensure that no small inputs occur.

A distinction is made between ‘ad hoc’ schemes, which have no provable reduction to the underlying hard problem, and provably secure schemes. Schemes in the first class are typically much more efficient, but many proposals have been broken; however, it seems that recently designers have been more conservative and designs survive longer.

Schemes Without Security Reduction. Most of these schemes uses a modulus N , that is, the product of two large primes. The size of N in bits (denoted with n) is typically between 512 and 1024. These hash functions can be useful in combination with RSA [77] as digital signature scheme. However, this choice poses the following practical problem: the person who has generated the modulus

knows its factorization, and hence he has a potential advantage over the other users of the hash function. One can try to design the hash function such that knowledge of the factorization does not help in attacking it (this is probably difficult to achieve). Alternatives are to use a trusted third party to generate the modulus (for example, the modulus of the Certification Authority), or to generate the modulus using a multi-party secure computation; recently practical solutions for such a computation have been developed by Boneh and Franklin [12] and Frankel *et al.* [40].

The most efficient schemes are based on modular squaring. An additional argument for squaring is that any algorithm that can extract modular square roots is reducible to a factoring algorithm (in random polynomial time). The best scheme seems to be of the following form: $H_i = (x_i \oplus H_{i-1})^2 \bmod N \oplus H_{i-1}$ [69].

It is essential to add redundancy to the message input. The first designs for this redundancy were not very successful (see for example Girault [45], Girault and Misarsky [47], and Coppersmith [20]). ISO/IEC SC27 has developed a new proposal, that is currently at the Final Draft International Standard (FDIS) level; it is called MASH-1 (for Modular Arithmetic Secure Hash) [51]:

$$H_i = ((x_i \oplus H_{i-1}) \vee A)^2 \pmod{N} \oplus H_{i-1}$$

here $A = 0xF00\dots00$, the four most significant bits in every byte of x_i are set to 1111, and the output of the squaring operation is chopped to n bits. A complex output transformation is added, which consists of a number of applications of the compression function; its goal is to destroy all the remaining mathematical structure. The final result is at most $n/2$ bits. The best known preimage and collision attacks on MASH-1 require $2^{n/2}$ and $2^{n/4}$ operations [21]; they are thus not better than brute force attacks. MASH-2 is a variant of MASH-1 which uses exponent $2^8 + 1$ [51]. This provides for an additional security margin.

Schemes With a Security Reduction. For several schemes there exists a security reduction to a number theoretic problem that is believed to be difficult. However, they are very slow: typically they hash $\log_2 \log_2 N$ bits per modular squaring (or even per modular exponentiation).

Damgård provides two hash functions for which finding a collision is provably equivalent to factoring an RSA modulus [24]. Gibson proposes a construction based on the discrete logarithm problem modulo a composite [43]. A third approach uses the discrete logarithm problem in a group of prime order p denoted with G_p (Bellare *et al.* [6], after earlier work by Chaum *et al.* [18] and Brands). Every non-trivial element of G_p is a generator. The hash function uses t random elements α_i from G_p ($\alpha_i \neq 1$). The hash result is then computed as

$$H_{t+1} = \prod_{i=1}^t \alpha_i^{x_i}.$$

Here \tilde{x}_i is obtained by considering the string x_i as the binary expansion of a number and prepending 1 to it. This avoids trivial collisions when x_i consists of all zeroes.

MDCs Based on Knapsack and Lattice Problems. The knapsack problem (which is a special case of the subset sum problem) of dimensions n and $\ell(n)$ can be defined as follows: given a set of n l -bit integers $\{a_1, a_2, \dots, a_n\}$, and an integer S , find a vector x with components x_i equal to 0 or 1 such that

$$\sum_{i=1}^n a_i \cdot x_i = S \pmod{2^{\ell(n)}}.$$

For application to hashing, one needs $n > \ell(n)$. The knapsack problem is known to be NP-hard; while this means that probably no feasible worst-case algorithms for this problem exists, this does not tell much about the hardness of a random instance. This problem was used in 1978 by Merkle and Hellman to construct the first public-key encryption system [62]. However, almost all public-key schemes based on the knapsack problem have been broken (see for example [65]), which has given the knapsack a bad reputation. The appeal of the knapsack problem (and related lattice based problems) lies in the fact that both hardware and software implementations are very fast compared to schemes based on number theoretic problems. Moreover, evaluation of a knapsack allows for significant parallelism. Finally, interesting security reductions can be proved: examples are the work for Impagliazzo and Naor [50] on knapsacks and that of Ajtai [3] for lattices; Ajtai was able to prove that if the shortest vector in a lattice problem is hard in the worst case, then the knapsack problem is hard on the average. However, some researchers believe that for realistic parameters, both these problems are relatively easy. If they are right, knapsack and lattice problems are not useful to practical cryptography.

Attacks on knapsacks often use the LLL lattice reduction algorithm [56] that finds the shortest vector in a lattice (the algorithm performs in practice much better than can be guaranteed). This reduction to the shortest vector problem only works for $\ell(n) > 1.0629 \cdot n$. Knapsack problems become more difficult when $n \approx \ell(n)$; however, the performance of the hash function decreases with the value $n - \ell(n)$. For $n = \ell(n)$, the best known attack requires time $O(2^{n/2})$ and space $O(2^{n/4})$. Impagliazzo and Naor summarize the state of the art in [50]. A different class of attacks are the algebraic attacks proposed by Camion and Patarin [14] and optimized by Patarin in [66]; these attacks tend to work better when $n \gg \ell(n)$. The scheme of Damgård [26] has been broken both using LLL [52] and using algebraic techniques [66]. It is for the time being an open problem whether a random knapsack with $n = 1024$, $l = 512$, and $\ell = 512$ is hard to solve.

Impagliazzo and Naor describe an efficient construction for a UOWHF (cf. Sect. 2.3) and provide a reduction of its security to that of the knapsack problem [50]. Ajtai introduced a function that is one-way (or preimage resistant) if the

problem of approximating the shortest vector in a lattice to polynomial factors is hard [3]. Goldreich *et al.* have proved that the function is in fact collision resistant [48].

Several multiplicative knapsacks have also been proposed; multiplicative notation is used for non-Abelian groups. The earliest proposal dates back to '77 (but it was quickly shown to be insecure). A recent example are the schemes by Zémor [88] and Tillich and Zémor [84]. Their security is based on the hardness of finding short factorizations in certain groups. In some cases one can even prove a lower bound on the Hamming distance between colliding messages. Attacks on these proposals (for certain parameters) can be found in [17, 41]. Impagliazzo and Naor also extend their construction on a UOWHF to multiplication in a group [50].

Knapsack and lattice based hash functions have also the potential problem that trapdoors may be inserted when the vectors are generated. Therefore it is recommended that the instance generation is reproducible (for example, through the use of a pseudo-random string generator or a hash function).

Incremental Hash Functions. A hash function (or any cryptographic primitive) is called *incremental* if it has the following property: if the hash function has been evaluated for an input x , and a small modification is made to x , resulting in x' , then one can update $h(x)$ in time proportional to the amount of modification between x and x' , rather than having to recompute $h(x')$ from scratch. If a function is incremental, it is automatically parallelizable as well.

This concept was first introduced by Bellare *et al.* [6]. They also made a first proposal based on exponentiation in a group of prime order. Improved constructions were proposed by Bellare and Micciancio [8] that consist of two steps:

- First the message is divided into blocks; each block (together with its index) is hashed using a conventional collision resistant hash function (restricted to fixed length inputs). This is called the ‘randomization’ step as in the analysis the hash function is treated as an ‘ideal’ hash function or random oracle (which is a very demanding requirement).
- Next the different outputs are combined using a group operation. This can be a group of large prime order in which the discrete logarithm problem is hard, and modular addition. The first approach leads to a reduction to the discrete logarithm problem, while the second leads to a reduction to the ‘weighted knapsack’ problem.

The same techniques can also be used to improve the lattice based hash function. These schemes have the advantage that they are much more efficient than the other schemes studied in this section. However, this comes at a cost of requiring a collision resistant hash function, which also has to behave ‘perfectly random.’ This construction is remarkable, as it constructs a collision resistant function based on a one-way property (but with specific algebraic structure, so there is no contradiction to the result of Simon [82] discussed in Sect. 3.3).

4.3 Custom Designed MDCs

This section discusses a selection of custom designed hash functions, i.e., algorithms that were especially designed for hashing operations. Most of these algorithms use the Davies-Meyer approach (cf. Sect. 4.1): the compression function is a block cipher, ‘keyed’ by the text input x_i ; the plaintext is the value H_{i-1} , which is also added to the ciphertext (feedforward).

In 1990, R. Rivest proposed MD4 [75], a hash function with a 128-bit result based on 32-bit integer arithmetic. While this hash function proved to be not sufficiently strong, the innovative design ideas have influenced many other designs. The algorithms derived from it (with improved strength) are often called the *MDx-family*. This family contains the most popular hash functions in use today. Dobbertin has found collisions for MD4; his attack combines algebraic techniques and optimization techniques such as genetic algorithms [32, 33]. It can be extended in such a way that even ‘meaningful’ collisions are obtained: the complete message (except for a few dozen bytes) is under complete control of the attacker. His attack also applies to the compression function of ‘extended MD4’ [75], which consists of concatenating two loosely coupled instances of MD4. Later Dobbertin showed that a reduced version of MD4 (2 rounds out of 3) is not preimage resistant [35].

Following early attacks on MD4 by Merkle and den Boer and Bosselaers [29], Rivest quickly designed a strengthened version, namely MD5 [76]. It was however shown by den Boer and Bosselaers [30] that the compression function of MD5 is not collision resistant (but their collisions are of the form $f(H_{i-1}, x_i) = f(H'_{i-1}, x_i)$, which is not immediately usable in practice). Dobbertin has extended his attack on MD4 to yield collisions for the compression function of MD5, i.e., $f(H_{i-1}, x_i) = f(H_{i-1}, x'_i)$, where he has some control over H_{i-1} [34]. It is believed that it is feasible to extend this attack to collisions for MD5 itself (that is, to take into account the *IV*).

A second improved variant of MD4, the Secure Hash Algorithm, was proposed by NIST [38] in 1993. The size of the hash result is increased from 128 to 160 bits and the message words are not simply permuted but encoded with a shortened cyclic code. After a few years, NSA discovered a certification weakness in SHA; apparently collisions can be found in less than 2^{80} operations. Consequently a new release of the standard was published. The new algorithm is called SHA-1 [39]. Recently Chabaud and Joux have published an attack that finds collisions for SHA in 2^{61} operations [16]; it is probably similar to the (classified) attack developed earlier that prompted the upgrade to SHA-1.

Yet another improved version of MD4, called RIPEMD, was developed in the framework of the EEC-RACE project RIPE [74]. Due to partial attacks by Dobbertin [32], it was later upgraded by Dobbertin *et al.* to RIPEMD-128 and RIPEMD-160, that have a 128-bit and a 160-bit result respectively [36]. Variants with a 256 and 320-bit result have been introduced as well. Together with SHA-1, RIPEMD-128 and RIPEMD-160 are the three custom designed hash functions included in ISO/IEC 10118-3 [51].

Merkle suggested in 1989 a software oriented one-way hash function called Snefru [61]. It is based on large random substitution tables (2 Kbyte per pass) and allows for a variable size of the result (128 or 256 bits). Biham and Shamir have shown that Snefru-128 [11] is vulnerable to differential attacks. As a consequence it is recommended to use 6 and preferably 8 passes, preferably with a 256-bit hash result. However, these measures increase the size of the substitution tables and decrease the performance.

Two of the most recent designs are Tiger and Panama. Tiger was proposed by Anderson and Biham [4]. It is tuned towards 64-bit processors and mixes Snefru-type S-boxes (8 input bits and 64 output bits) with arithmetic operations. Panama is a design of Daemen and Clapp [23]; it has been designed to take advantage of the instruction-level parallelism in modern processors.

5 Conclusions and Open Problems

In spite of the popularity of cryptographic hash functions, very few theoretical results are known in this area; it is clear that further research is necessary to improve our understanding of these primitives. Collision resistance seems to be a condition that is particularly hard to analyze. Some open problems are whether it is possible to construct collision resistant hash functions based on weaker assumptions, and whether any theory can be developed to support the current constructions.

In the area of practical constructions, there is a need for more efficient hash functions, the security of which is better understood. For hash functions based on block ciphers, the main problem seems to be the assumptions on the block cipher. From an application viewpoint, multiplicative knapsacks seem to be very attractive (due to inherent parallelism and due to the fact that certain properties can be proved). However, further research is necessary to assess their security. Another research problem is to understand to which extent the current constructions provide other security properties such as pseudo-randomness and partial preimage resistance; both properties are related to the difficulty of inverting the hash function if part of the input is known.

References

1. W. Aiello, R. Venkatesan, "Foiling birthday attacks in length-doubling transformations. Benes: a non-reversible alternative to Feistel," *Advances in Cryptology, Proceedings Eurocrypt'96, LNCS 1070*, U. Maurer, Ed., Springer-Verlag, 1996, pp. 307–320.
2. W. Aiello, S. Haber, R. Venkatesan, "New constructions for secure hash functions," *Fast Software Encryption, LNCS 1372*, S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 150–167.
3. M. Ajtai, "Generating hard instances of lattice problems," *Proc. 28th ACM Symposium on the Theory of Computing*, 1996, pp. 99–108.
4. R. Anderson, E. Biham, "Tiger: A new fast hash function," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 89–97.

Appeared in *Lectures on Data Security*, Lecture Notes in Computer Science 1561, I. Damgård (ed.), pp. 158–182, 1999.

©1999 Springer-Verlag

5. M. Bellare, R. Canetti, H. Krawczyk, "Pseudorandom functions revisited: The cascade construction and its concrete security," *Proc. 37th Annual Symposium on the Foundations of Computer Science, IEEE*, 1996, pp. 514–523.
Full version via <http://www-cse.ucsd.edu/users/mihir>.
6. M. Bellare, O. Goldreich, S. Goldwasser, "Incremental cryptography: the case of hashing and signing," *Advances in Cryptology, Proceedings Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 216–233.
7. M. Bellare, J. Kilian, P. Rogaway, "The security of cipher block chaining," *Advances in Cryptology, Proceedings Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 341–358.
8. M. Bellare, D. Micciancio, "A new paradigm for collision-free hashing: incrementality at reduced cost," *Advances in Cryptology, Proceedings Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 163–192.
9. M. Bellare, P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," *Proc. 1st ACM Conference on Computer and Communications Security*, ACM, 1993, pp. 62–73.
10. M. Bellare, P. Rogaway, "Collision-resistant hashing: towards making UOWHFs practical," *Advances in Cryptology, Proceedings Crypto'97, LNCS 1294*, B. Kaliski, Ed., Springer-Verlag, 1997, pp. 470–484.
11. E. Biham, A. Shamir, "*Differential Cryptanalysis of the Data Encryption Standard*," Springer-Verlag, 1993.
12. D. Boneh, M. Franklin, "Efficient generation of shared RSA keys," *Advances in Cryptology, Proceedings Crypto'97, LNCS 1294*, B. Kaliski, Ed., Springer-Verlag, 1997, pp. 425–439.
13. B.O. Brachtel, D. Coppersmith, M.M. Hyden, S.M. Matyas, C.H. Meyer, J. Oseas, S. Pilpel, M. Schilling, "*Data Authentication Using Modification Detection Codes Based on a Public One Way Encryption Function*," U.S. Patent Number 4,908,861, March 13, 1990.
14. P. Camion, J. Patarin, "The knapsack hash function proposed at Crypto'89 can be broken," *Advances in Cryptology, Proceedings Eurocrypt'91, LNCS 547*, D.W. Davies, Ed., Springer-Verlag, 1991, pp. 39–53.
15. J.L. Carter, M.N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, Vol. 18, 1979, pp. 143–154.
16. F. Chabaud, A. Joux, "Differential collisions: an explanation for SHA-1," *Advances in Cryptology, Proceedings Crypto'98, LNCS 1462*, H. Krawczyk, Ed., Springer-Verlag, 1998, pp. 56–71.
17. C. Charnes, J. Pieprzyk, "Attacking the SL_2 hashing scheme," *Advances in Cryptology, Proceedings Asiacrypt'94, LNCS 917*, J. Pieprzyk and R. Safavi-Naini, Eds., Springer-Verlag, 1995, pp. 322–330.
18. D. Chaum, E. van Heijst, B. Pfitzmann, "Cryptographically strong undeniable signatures, unconditionally secure for the signer," *Advances in Cryptology, Proceedings Crypto'91, LNCS 576*, J. Feigenbaum, Ed., Springer-Verlag, 1992, pp. 470–484.
19. D. Coppersmith, "Another birthday attack," *Advances in Cryptology, Proceedings Crypto'85, LNCS 218*, H.C. Williams, Ed., Springer-Verlag, 1985, pp. 14–17.
20. D. Coppersmith, "Analysis of ISO/CCITT Document X.509 Annex D," *IBM T.J. Watson Center, Yorktown Heights, N.Y., 10598, Internal Memo*, June 11, 1989, (also ISO/IEC JTC1/SC20/WG2/N160).
21. D. Coppersmith, B. Preneel, "Comments on MASH-1 and MASH-2," February 21, 1995, ISO/IEC JTC1/SC27/N1055.
22. T. Cormen, C. Leieron, R. Rivest, "*Introduction to Algorithms*," McGraw–Hill, 1992.

Appeared in *Lectures on Data Security*, Lecture Notes in Computer Science 1561,
I. Damgård (ed.), pp. 158–182, 1999.
©1999 Springer-Verlag

23. J. Daemen, C. Clapp, "Fast hashing and stream encryption with PANAMA," *Fast Software Encryption, LNCS 1372*, S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 60–74.
24. I.B. Damgård, "Collision free hash functions and public key signature schemes," *Advances in Cryptology, Proceedings Eurocrypt'87, LNCS 304*, D. Chaum and W.L. Price, Eds., Springer-Verlag, 1988, pp. 203–216.
25. I.B. Damgård, "The application of claw free functions in cryptography," *PhD Thesis*, Aarhus University, Mathematical Institute, 1988.
26. I.B. Damgård, "A design principle for hash functions," *Advances in Cryptology, Proceedings Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 416–427.
27. I.B. Damgård, T.P. Pedersen, B. Pfitzmann, "On the existence of statistically hiding bit commitment schemes and fail-stop signatures," *Advances in Cryptology, Proceedings Crypto'93, LNCS 773*, D. Stinson, Ed., Springer-Verlag, 1994, pp. 250–265.
28. D. Davies, W. L. Price, "The application of digital signatures based on public key cryptosystems," *NPL Report DNACS 39/80*, December 1980.
29. B. den Boer, A. Bosselaers, "An attack on the last two rounds of MD4," *Advances in Cryptology, Proceedings Crypto'91, LNCS 576*, J. Feigenbaum, Ed., Springer-Verlag, 1992, pp. 194–203.
30. B. den Boer, A. Bosselaers, "Collisions for the compression function of MD5," *Advances in Cryptology, Proceedings Eurocrypt'93, LNCS 765*, T. Helleseht, Ed., Springer-Verlag, 1994, pp. 293–304.
31. W. Diffie, M.E. Hellman, "New directions in cryptography," *IEEE Trans. on Information Theory*, Vol. IT-22, No. 6, 1976, pp. 644–654.
32. H. Dobbertin, "RIPEMD with two-round compress function is not collisionfree," *Journal of Cryptology*, Vol. 10, No. 1, 1997, pp. 51–69.
33. H. Dobbertin, "Cryptanalysis of MD4," *Journal of Cryptology*, Vol. 11, No. 4, 1998, pp. 253–271. See also *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 53–69.
34. H. Dobbertin, "The status of MD5 after a recent attack," *CryptoBytes*, Vol. 2, No. 2, Summer 1996, pp. 1–6.
35. H. Dobbertin, "The first two rounds of MD4 are not one-way," *Fast Software Encryption, LNCS 1372*, S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 284–292.
36. H. Dobbertin, A. Bosselaers, B. Preneel, "RIPEMD-160: a strengthened version of RIPEMD," *Fast Software Encryption, LNCS 1039*, D. Gollmann, Ed., Springer-Verlag, 1996, pp. 71–82.
See also <http://www.esat.kuleuven.ac.be/~bosselae/ripemd160>.
37. FIPS 46, "*Data Encryption Standard*," Federal Information Processing Standard, National Bureau of Standards, U.S. Department of Commerce, Washington D.C., January 1977 (revised as FIPS 46-1:1988; FIPS 46-2:1993).
38. FIPS 180, "*Secure Hash Standard*," Federal Information Processing Standard (FIPS), Publication 180, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., May 11, 1993.
39. FIPS 180-1, "*Secure Hash Standard*," Federal Information Processing Standard (FIPS), Publication 180-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., April 17, 1995.
40. Y. Frankel, P. D. MacKenzie, M. Yung, "Robust efficient distributed RSA-key generation," *Proc. 30th ACM Symposium on the Theory of Computing*, 1998.

Appeared in *Lectures on Data Security*, Lecture Notes in Computer Science 1561, I. Damgård (ed.), pp. 158–182, 1999.

©1999 Springer-Verlag

41. W. Geiselmann, "A note on the hash function of Tillich and Zémor," *Cryptography and Coding. 5th IMA Conference*, C. Boyd, Ed., Springer-Verlag, 1995, pp. 257–263.
42. J.K. Gibson, "Some comments on Damgård's hashing principle," *Electronics Letters*, Vol. 26, No. 15, 1990, pp. 1178–1179.
43. J.K. Gibson, "Discrete logarithm hash function that is collision free and one way," *IEE Proceedings-E*, Vol. 138, No. 6, November 1991, pp. 407–410.
44. E. Gilbert, F. MacWilliams, N. Sloane, "Codes which detect deception," *Bell System Technical Journal*, Vol. 53, No. 3, 1974, pp. 405–424.
45. M. Girault, "Hash-functions using modulo-n operations," *Advances in Cryptology, Proceedings Eurocrypt'87, LNCS 304*, D. Chaum and W.L. Price, Eds., Springer-Verlag, 1988, pp. 217–226.
46. M. Girault, R. Cohen, M. Campana, "A generalized birthday attack," *Advances in Cryptology, Proceedings Eurocrypt'88, LNCS 330*, C.G. Günther, Ed., Springer-Verlag, 1988, pp. 129–156.
47. M. Girault, J.-F. Misarsky, "Selective forgery of RSA signatures using redundancy," *Advances in Cryptology, Proceedings Eurocrypt'97, LNCS 1233*, W. Fumy, Ed., Springer-Verlag, 1997, pp. 495–507.
48. O. Goldreich, S. Goldwasser, S. Halevi, "Collision-free hashing from lattice problems," *Theory of Cryptography Library*, <http://philby.ucsd.edu/CRYPTOLIB.html>, 96–09, July 1996.
49. M. Hellman, "A cryptanalytic time-memory tradeoff," *IEEE Trans. on Information Theory*, Vol. IT-26, 1980, pp. 401–406.
50. R. Impagliazzo, M. Naor, "Efficient cryptographic schemes provably as secure as subset sum," *Journal of Cryptology*, Vol. 9, No. 4, 1996, pp. 199–216.
51. ISO/IEC 10118, "*Information technology – Security techniques – Hash-functions, Part 1: General*", 1994, "*Part 2: Hash-functions using an n-bit block cipher algorithm*", 1994, "*Part 3: Dedicated hash-functions*", 1998, "*Part 4: Hash-functions using modular arithmetic*," (FDIS) 1998.
52. A. Joux, L. Granboulan, "A practical attack against knapsack based hash functions," *Advances in Cryptology, Proceedings Eurocrypt'94, LNCS 950*, A. De Santis, Ed., Springer-Verlag, 1995, pp. 58–66.
53. L.R. Knudsen, X. Lai, B. Preneel, "Attacks on fast double block length hash functions," *Journal of Cryptology*, Vol. 11, No. 1, Winter 1998, pp. 59–72.
54. L.R. Knudsen, B. Preneel, "Fast and secure hashing based on codes," *Advances in Cryptology, Proceedings Crypto'97, LNCS 1294*, B. Kaliski, Ed., Springer-Verlag, 1997, pp. 485–498.
55. X. Lai, J.L. Massey, "Hash functions based on block ciphers," *Advances in Cryptology, Proceedings Eurocrypt'92, LNCS 658*, R.A. Rueppel, Ed., Springer-Verlag, 1993, pp. 55–70.
56. A. Lenstra, H. Lenstra, L. Lovász, "Factoring polynomials with rational coefficients," *Mathematischen Annalen*, Vol. 261, pp. 515–534, 1982.
57. S.M. Matyas, C.H. Meyer, J. Oseas, "Generating strong one-way functions with cryptographic algorithm," *IBM Techn. Disclosure Bull.*, Vol. 27, No. 10A, 1985, pp. 5658–5659.
58. A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
59. R. Merkle, "*Secrecy, Authentication, and Public Key Systems*," UMI Research Press, 1979.
60. R. Merkle, "One way hash functions and DES," *Advances in Cryptology, Proceedings Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 428–446.

Appeared in *Lectures on Data Security*, Lecture Notes in Computer Science 1561,
I. Damgård (ed.), pp. 158–182, 1999.

©1999 Springer-Verlag

61. R. Merkle, "A fast software one-way hash function," *Journal of Cryptology*, Vol. 3, No. 1, 1990, pp. 43–58.
62. R. Merkle, M. Hellman, "Hiding information and signatures in trapdoor knapsacks," *IEEE Trans. on Information Theory*, Vol. IT-24, No. 5, 1978, pp. 525–530.
63. C.H. Meyer, M. Schilling, "Secure program load with Manipulation Detection Code," *Proc. Securicom 1988*, pp. 111–130.
64. M. Naor, M. Yung, "Universal one-way hash functions and their cryptographic applications," *Proc. 21st ACM Symposium on the Theory of Computing*, 1990, pp. 387–394.
65. A.M. Odlyzko, "The rise and fall of knapsack cryptosystems," *Cryptology and Computational Number Theory*, C. Pomerance, Ed., Proc. Sympos. Appl. Math., Vol. 42, American Mathematical Society, 1990, pp. 75–88.
66. J. Patarin, "Collisions and inversions for Damgård's whole hash function," *Advances in Cryptology, Proceedings Asiacrypt'94, LNCS 917*, J. Pieprzyk and R. Safavi-Naini, Eds., Springer-Verlag, 1995, pp. 307–321.
67. B. Preneel, "Analysis and design of cryptographic hash functions," *Doctoral Dissertation*, Katholieke Universiteit Leuven, 1993.
68. B. Preneel, "Cryptographic primitives for information authentication — State of the art," *State of the Art in Applied Cryptography, LNCS 1528*, B. Preneel and V. Rijmen, Eds., Springer-Verlag, 1998, pp. 50–105.
69. B. Preneel, R. Govaerts, J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach," *Advances in Cryptology, Proceedings Crypto'93, LNCS 773*, D. Stinson, Ed., Springer-Verlag, 1994, pp. 368–378.
70. B. Preneel, P.C. van Oorschot, "MDx-MAC and building fast MACs from hash functions," *Advances in Cryptology, Proceedings Crypto'95, LNCS 963*, D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 1–14.
71. J.-J. Quisquater, J.-P. Delescaille, "How easy is collision search ? Application to DES," *Advances in Cryptology, Proceedings Eurocrypt'89, LNCS 434*, J.-J. Quisquater and J. Vandewalle, Eds., Springer-Verlag, 1990, pp. 429–434.
72. J.-J. Quisquater, J.-P. Delescaille, "How easy is collision search. New results and applications to DES," *Advances in Cryptology, Proceedings Crypto'89, LNCS 435*, G. Brassard, Ed., Springer-Verlag, 1990, pp. 408–413.
73. M.O. Rabin, "Digitalized signatures," in *Foundations of Secure Computation*, R. Lipton, R. DeMillo, Eds., Academic Press, New York, 1978, pp. 155–166.
74. RIPE, "Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040)," *LNCS 1007*, A. Bosselaers and B. Preneel, Eds., Springer-Verlag, 1995.
75. R.L. Rivest, "The MD4 message digest algorithm," *Advances in Cryptology, Proceedings Crypto'90, LNCS 537*, S. Vanstone, Ed., Springer-Verlag, 1991, pp. 303–311.
76. R.L. Rivest, "The MD5 message-digest algorithm," *Request for Comments (RFC) 1321*, Internet Activities Board, Internet Privacy Task Force, April 1992.
77. R.L. Rivest, A. Shamir, L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications ACM*, Vol. 21, February 1978, pp. 120–126.
78. J. Rompel, "One-way functions are necessary and sufficient for secure signatures," *Proc. 22nd ACM Symposium on the Theory of Computing*, 1990, pp. 387–394.
79. A. Russell, "Necessary and sufficient conditions for collision-free hashing," *Journal of Cryptology*, Vol. 8, No. 2, 1995, pp. 87–99.

Appeared in *Lectures on Data Security*, Lecture Notes in Computer Science 1561, I. Damgård (ed.), pp. 158–182, 1999.

©1999 Springer-Verlag

80. G.J. Simmons, "A survey of information authentication," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 381–419.
81. G.J. Simmons, "How to insure that data acquired to verify treat compliance are trustworthy," in *Contemporary Cryptology: The Science of Information Integrity*, G.J. Simmons, Ed., IEEE Press, 1991, pp. 615–630.
82. D. Simon, "Finding collisions on a one-way street: Can secure hash functions be based on general assumptions?" *Advances in Cryptology, Proceedings Eurocrypt'98, LNCS 1403*, K. Nyberg, Ed., Springer-Verlag, 1998, pp. 334–345.
83. D.R. Stinson, "Universal hashing and authentication codes," *Designs, Codes, and Cryptography*, Vol. 4, No. 4, 1994, pp. 369–380. See also *Advances in Cryptology, Proceedings Crypto'91, LNCS 576*, J. Feigenbaum, Ed., Springer-Verlag, 1992, pp. 74–85.
84. J.-P. Tillich, G. Zémor, "Hashing with SL_2 ," *Advances in Cryptology, Proceedings Crypto'94, LNCS 839*, Y. Desmedt, Ed., Springer-Verlag, 1994, pp. 40–49.
85. P.C. van Oorschot, M.J. Wiener, "Parallel collision search with application to hash functions and discrete logarithms," *Proc. 2nd ACM Conference on Computer and Communications Security*, ACM, 1994, pp. 210–218 (final version to appear in *Journal of Cryptology*).
86. M.N. Wegman, J.L. Carter, "New hash functions and their use in authentication and set equality," *Journal of Computer and System Sciences*, Vol. 22, No. 3, 1981, pp. 265–279.
87. G. Yuval, "How to swindle Rabin," *Cryptologia*, Vol. 3, 1979, pp. 187–189.
88. G. Zémor, "Hash functions and Cayley graphs," *Designs, Codes, and Cryptography*, Vol. 4, No. 4, 1994, pp. 381–394.
89. Y. Zheng, T. Matsumoto, H. Imai, "Connections between several versions of one-way hash functions," *Proc. SCIS90, The 1990 Symposium on Cryptography and Information Security*, Nihondaira, Japan, Jan. 31–Feb. 2, 1990.