Memoryless Near-Collisions via Coding Theory

Mario Lamberger[1], Florian Mendel[1], Vincent Rijmen[1,2], and Koen Simoens[2]

[1] Institute for Applied Information Processing and Communications
Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria.
[2] Dept. of Electrical Engineering ESAT/COSIC, K.U.Leuven,
and Interdisciplinary Institute for BroadBand Technology (IBBT),
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium.

Abstract. We investigate generic methods to find near-collisions in cryptographic hash functions. We introduce a new generic approach based on methods to find cycles in the space of codewords of a code with low covering radius. We give an analysis of our approach and demonstrate it on the SHA-3 candidate TIB3.

Keywords: Hash functions, near-collisions, cycle finding algorithms, covering codes

1 Introduction

After the publication of the attacks on MD5, SHA-1 and several other modern cryptographic hash functions by Wang *et al.* [29,30], there has been a renewed interest in the design and cryptanalysis of these important cryptographic primitives.

Cryptographic hash functions have to satisfy many requirements, among which the properties of preimage resistance, second preimage resistance and collision resistance are cited most often. While designers of practical proposals usually try to make their design satisfy some additional properties, most theoretical constructions and their accompanying proofs of security consider these three properties only.

In this paper, we are concerned with a somewhat less popular property, namely *near-collision resistance*. We want to investigate ways how to efficiently find such near-collisions and compare them with the generic approaches used to find collisions.

2 Background and Motivation

2.1 Hash and Compression Function Collisions

In general, a cryptographic hash function maps a message m of arbitrary length to a hash value of fixed length, $H : \{0,1\}^* \to \{0,1\}^n$. In this paper, we consider iterative hash functions H that split up the message into blocks of equal size.

Designs, Codes and Cryptography, 62(1), pp.1–18, 2012
The original publication is available at http://www.springerlink.com/content/h3t8181044112217/

So upon input a message m, we apply an injective padding such that the result consists of l blocks m_i, $i = 0, 1, \ldots, l-1$ of q bits each, and then process one block m_i at a time to update the n-bit internal state x_i. The *compression function* is the same function in every iteration, and is denoted by h:

$$x_{i+1} = h(x_i, m_i), \qquad i = 0, 1, \ldots, l-1 \tag{1}$$

Here x_0 is a pre-defined initial state value. The output of the hash function is defined as the final state value:

$$H(m) = x_l.$$

The strengthened Merkle-Damgård construction (further abbreviated to MD-construction) of cryptographic hash functions [6,16], which is basically (1) where the message padding also includes the bitlength of m, is very popular in practical designs like MD5, SHA-1 and the SHA-2 family [20,24]. This is mostly because of its property of *preserving the collision resistance* of the compression function:

Theorem 1. *Let H be a hash function based on the MD-construction and let $m \neq m^*$ be two messages. Then*

$$H(m) = H(m^*) \Rightarrow \exists i : h(x_i, m_i) = h(x_i^*, m_i^*) \tag{2}$$

A solution $\{(x_i, m_i), (x_i^*, m_i^*)\}$ to $h(x_i, m_i) = h(x_i^*, m_i^*)$ where $(x_i, m_i) \neq (x_i^*, m_i^*)$ is called a *collision for the compression function h.*

One early result in the field of hash function cryptanalysis was found by den Boer and Bosselaers, namely, that collisions for the compression function of the widely used MD5 hash function can be found easily [7]. Although the methods of [7] can't be used to construct collisions for MD5, this early result implied already that Theorem 1 can't be used to prove the security of MD5.

2.2 Near-Collisions

In all of the following we will work with binary values, where we identify $\{0,1\}^n$ with $\mathbb{Z}_2^n$. We denote the standard basis vectors of $\mathbb{Z}_2^n$ by e_j, $j = 1, \ldots, n$. Let "+" denote the n-bit exclusive-or operation. The Hamming weight of a vector $v \in \mathbb{Z}_2^n$ is denoted by $\mathrm{w}(v) = \#\{j \mid v_j = 1\}$ and the Hamming distance of two vectors by $\mathrm{d}(u, v) = \mathrm{w}(u + v)$. The Handbook of Applied Cryptography defines *near-collision resistance* as follows:

Definition 1 (Near-Collision Resistance [15, page 331]). *It should be hard to find any two inputs m, m^* with $m \neq m^*$ such that $H(m)$ and $H(m^*)$ differ in only a small number of bits:*

$$\mathrm{d}(H(m), H(m^*)) \leq \epsilon. \tag{3}$$

For ease of later use we also give the following definition:

Definition 2. *A message pair m, m^* with $m \neq m^*$ is called an ϵ-near-collision for H if* (3) *holds.*

Intuitively speaking, a hash function for which an efficient algorithm is known to construct near-collisions, can no longer be considered to be ideal. A practically more relevant consequence is that for several designs, near-collisions for the compression function can be converted to collisions for the hash function, see Sect. 2.3.

Let δ denote an n-bit vector, possibly of low Hamming weight. In the strict near-collision problem, we want to find two messages m, m^* such that

$$H(m^*) + H(m) = \delta.$$

If δ is fixed on beforehand, the strict near-collision problem is not necessarily significantly easier than finding collisions.

2.3 Combining a Near-Collision and a Collision for h to a Collision for H

As a motivation why we should certainly bother about near-collisions we give the following example. Although collisions for the compression function of MD5 can be constructed easily [7], these collisions require a special difference in the *state* input. To date, there is no algorithm known that can produce collisions for the compression function of MD5 without having a difference in the state input. Since there is no algorithm known to construct message blocks resulting in states with this difference, the collisions for the compression function can't be converted into collisions for MD5.

A significant contribution of [29] was the description of an efficient algorithm to find collisions for MD5, see Algorithm 1 for a simplified description. The first phase of the algorithm consists of the construction of a near-collision for h, where the output difference is such that in the second phase of the algorithm, it is feasible to construct a collision for h with this difference in the state input.

Algorithm 1 Wang et al.'s algorithm to create collisions for H (simplified).

Input: Initial value x_0 and hash function H with compression function h
Find m_0, m_0^* such that $x_1 + x_1^* = h(x_0, m_0) + h(x_0, m_0^*) = \Delta$
Find m_1, m_1^* such that $h(x_1, m_1) = h(x_1^*, m_1^*)$
Output: m_0, m_0^*, m_1, m_1^*

For many hash functions using a Davies-Meyer mode iteration function [13], like SHA-1, HAVAL, and reduced variants of SHA-256, it turns out that it is relatively easy to find collisions for the compression function (with differences in the state input). Hence, if we can also find good methods to construct near-collisions of the right form, then we can use Algorithm 1 to construct collisions. This will also be illustrated for the SHA-3 candidate TIB3 in Sect. 4.

3 Efficiently Finding Near-Collisions

Although collision resistance and (second) preimage resistance are the properties of a hash function that have attracted the most attention in cryptanalysis, the question of near-collision resistance is also of significant importance. Examples for this are applications that truncate the hash value at the end in which case a near-collision might be sufficient to thwart a security goal. In Sect. 4, we will also show an example where near-collisions for the compression function can be used to construct collisions for the full hash function.

In this section, we present a new generic method to find near-collisions. First we give a short discussion of generic methods to find collisions. Since hash functions were our main motivation for the underlying research, we will mainly use them to formulate the results that follow. Note that hash functions could as well be replaced with compression functions or arbitrary random functions.

3.1 Generic Collision Finding

The generic method for finding collisions for a given hash function is based on the *birthday paradox*. The basic principle of this attack is that when randomly drawing elements from a set of size 2^n, with high probability a repeated element will be encountered after about $\sqrt{2^n}$ drawings [15]. Due to their complexity, these generic birthday attacks are also often called *square-root attacks*. When implementing such a square-root attack there usually are different possibilities. The simplest approach is to randomly select messages y_j, compute $H(y_j)$ and store the results in a table until a collision is detected. This approach, which is usually attributed to Yuval [31], requires approximately $2^{n/2}$ hash function computations, and a table of the same size.

If a birthday attack is implemented and run, usually the memory requirements form the bottleneck. Therefore, collision attacks are often implemented by means of cycle finding algorithms. Consider the process where we start with an arbitrary n-bit value y_0 and repeatedly apply H:

$$y_0 \xrightarrow{H} y_1 \xrightarrow{H} y_2 \xrightarrow{H} y_3 \xrightarrow{H} \dots \tag{4}$$

This can be seen as a walk on a graph with 2^n nodes, induced by a (pseudo-) random map, namely the hash function. Since the output space is finite we eventually arrive at the situation that $y_{i+1} = H(y_i) = H(y_j) = y_{j+1}$. But then due to the definition of our walk, also $y_{i+\ell} = y_{j+\ell}$ for $\ell \geq 1$, i.e., the walk runs into a cycle. Put differently, we can consider $y_0, y_1, y_2, \dots$ as an eventually periodic sequence. For this sequence there exist two unique smallest integers μ and λ such that $y_i = y_{i+\lambda}$ for all $i \geq \mu$. Here, λ is the cycle length and μ is the tail length. Under the assumption, that H behaves like a random mapping, Harris [11] could show that the expected values for λ and μ are about $\sqrt{\pi 2^{n-3}}$.

There are two well known techniques based on the above ideas to identify collisions for (pseudo)-random mappings which are due to Floyd [12, p. 7] and Brent [2]. Floyd's cycle finding algorithm is widely known and cited. It is based on

the observation that for an eventually periodic sequence $y_0, y_1, \ldots$, there exists an index i such that $y_i = y_{2i}$ and the smallest such i satisfies $\mu \leq i \leq \mu + \lambda$. Floyd's algorithm only needs a small constant amount of memory and again under the assumption that H behaves like a random mapping, it can be shown that the expected number of iterations is about $0.94 \cdot 2^{n/2}$.

Brent [2] improved on Floyd's algorithm by introducing an auxiliary variable z which stores the value $y_{\ell(i)-1}$ where i is the index of the random walk and $\ell(i)$ is the largest power of 2 less or equal to i. In other words, $\ell(i) = 2^{\lfloor \log_2(i) \rfloor}$. Brent's method is described in Algorithm 2. On average, Brent found that his algorithm needs twice as much iterations as Floyd's algorithm, that is $1.9828 \cdot 2^{n/2}$. The actual improvement of Brent is that in each iteration only one hash evaluation is necessary instead of 3 in the case of Floyd.

Algorithm 2 Brent's cycle finding algorithm

Input: Starting point y_0 and hash function H
$z \leftarrow y_0, w \leftarrow y_0, i \leftarrow 0, \ell \leftarrow 1$
while true **do**
  $w \leftarrow H(w), i \leftarrow i + 1$
  if $w = z$ **then**
    break
  end if
  if $i \geq (2\ell - 1)$ **then**
    $z \leftarrow w, \ell \leftarrow 2\ell$
  end if
end while
$j \leftarrow \ell - 1$
Output: (m, m^*) with $m = H^{i-1}(y_0)$ and $m^* = H^{j-1}(y_0)$

Apart from these classical examples of cycle finding algorithms, there are various parallelization techniques available, but all result in higher memory requirements [26,27]. Furthermore, we also want to mention a more recent technique due to Nivasch [19] which can be seen as the best technique on average in terms of (hash) function evaluations. A lot of impulses in the development of cycle finding techniques have come from the computation of discrete logarithms in finite groups (e. g. Pollard's rho-method [22]). For a nice treatise we refer to [5, Sect. 19.5.1].

3.2 Generic Near-Collision Attacks

Obviously, Definition 2 includes collisions as well, so the task of finding near-collisions is easier than finding collisions. The goal is now to find a generic method to construct near-collisions more efficiently than the generic methods to find collisions.

In all of the following, let $B_r(x) = \{y \in \mathbb{Z}_2^n \mid \mathrm{d}(x, y) \leq r\}$ denote the *Hamming sphere* around x of radius r. Furthermore, we denote by

$$V(n, r) := |B_r(x)| = \sum_{i=0}^{r} \binom{n}{i} \tag{5}$$

the cardinality of any n-dimensional Hamming sphere of radius r. With this notation, $B_\epsilon(0)$ would be the set of all vectors having Hamming weight $\leq \epsilon$.

A first approach to find ϵ-near-collisions is a simple extension of the table-based birthday attack which leads to Algorithm 3.

Algorithm 3 Birthday-like ϵ-near-collision search

Input: Hash function H
$\mathcal{T} = \{\}$
while true **do**
  Randomly select a message m and compute $H(m)$
  if $(H(m) + \delta, m^*) \in \mathcal{T}$ for some $\delta \in B_\epsilon(0)$ and arbitrary m^* **then**
    return (m, m^*);
  else
    Add $(H(m), m)$ to $\mathcal{T}$
  end if
end while
Output: m, m^* such that $\mathrm{d}(H(m), H(m^*)) \leq \epsilon$

Lemma 1 *If we assume that H acts like a random mapping, the average number of messages that we need to hash and store in Algorithm 3 before we find an ϵ-near-collisions is*

$$\approx \frac{2^{n/2}}{\sqrt{V(n, \epsilon)}}\,. \tag{6}$$

Proof. Consider a set $\mathcal{C} = \{y_0, \ldots, y_{L-1}\}$ of L independent, uniformly distributed random variables with values in $\mathbb{Z}_2^n$. We now consider the random variables $\mathrm{d}(y_i, y_j)$ and let furthermore χ be the characteristic function of the event $\mathrm{d}(y_i, y_j) \leq \epsilon$, that is,

$$\chi(\mathrm{d}(y_i, y_j) \leq \epsilon) = \begin{cases} 1 & \text{if } \mathrm{d}(y_i, y_j) \leq \epsilon \\ 0 & \text{otherwise.} \end{cases}$$

Now, for $i \neq j$ we consider the number $N_{\mathcal{C}}(\epsilon)$ of pairs (y_i, y_j) from $\mathcal{C}$ which have $\mathrm{d}(y_i, y_j) \leq \epsilon$ (that is, the number of ϵ-near-collisions):

$$N_{\mathcal{C}}(\epsilon) = \sum_{i=0}^{L-1} \sum_{j=0}^{i-1} \chi(\mathrm{d}(y_i, y_j) \leq \epsilon)$$

The expected value of this sum of pairwise-independent random variables can be computed as

$$\mathbb{E}(N_{\mathcal{C}}(\epsilon)) = \binom{L}{2} V(n,\epsilon) 2^{-n}.$$

Therefore, if we choose L such that $L(L-1) > 2^{n+1}/V(n,\epsilon)$, the expected number of ϵ-near-collisions is at least 1. ∎

Remark 1. The proof of the previous lemma stems in part from the proof of [1, Th. 2. 1] where similar arguments apply in the context of random codes.

We see that, depending on ϵ, finding ϵ-near-collisions is clearly easier than finding collisions. The question that now arises is whether or not we can find a memoryless algorithm for the search for ϵ-near-collisions. Unfortunately, we cannot use cycle finding methods like Algorithm 2 directly, because a cycle only occurs when there is a full collision. There is no such thing as a "near-cycle".

A first approach to find near-collisions in a memoryless way is as follows. Let $I = \{j_1, \ldots, j_\epsilon\} \subseteq \{1, \ldots, n\}$ be a set of mutually distinct indices. Let p_I denote the linear projection map on the space $\mathbb{Z}_2^n$, which sets the bits of its argument to zero at the positions in I, that is, $p_I : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$ with

$$p_I(x)_j = \begin{cases} x_j & j \notin I \\ 0 & j \in I. \end{cases}$$

It follows that if $p_I(H(m)) = p_I(H(m^*))$, then $H(m)$ and $H(m^*)$ can differ only in the bits determined by I, hence they are ϵ-near-collisions.

Now we are again in the position to apply a cycle finding algorithm. Since we know that $\dim(\mathrm{Im}(p_I)) = n - \epsilon$, the expected number of iterations in Algorithm 2 is reduced to about $2^{(n-\epsilon)/2}$. This is a performance improvement factor of $2^{\epsilon/2}$ compared to the search for full collisions. On the negative side, this approach can only find a fraction of all possible ϵ-near-collisions, namely

$$\frac{2^\epsilon}{V(n,\epsilon)}. \tag{7}$$

We can generalize this approach by replacing the projection p_I by a more general map g. Ideally, we would like to have a one-to-one correspondence between ϵ-near-collisions ($\epsilon \geq 1$) for H and collisions for $g \circ H$:

$$\mathrm{d}(H(m), H(m^*)) \leq \epsilon \Leftrightarrow g(H(m)) = g(H(m^*)). \tag{8}$$

However, we can show a negative result in this direction.

Lemma 2 *Let $\epsilon \geq 1$, let H be a hash function and let g be a function such that (8) holds. Then, g is a constant map and $\mathrm{d}(H(m), H(m^*)) \leq \epsilon$ for all m, m^*.*

Proof. For an arbitrary message m, we have

$$\mathrm{d}(H(m), H(m) + e_j) = \mathrm{w}(e_j) = 1 \leq \epsilon$$

and thus

$$g(H(m)) = g(H(m) + e_j)$$

for all $j \in \{1, \ldots, n\}$. It is easy to see that g is constant on the span of the e_j, which is all of $\mathbb{Z}_2^n$. But this implies $\mathrm{d}(H(m), H(m^*)) \leq \epsilon, \forall m, m^*$, which is clearly not the case for the interesting hash functions H. ∎

3.3 An Approach Using Coding Theory

The memoryless method to find near-collisions based on the projection map p_I introduced in the previous section suffers from the fact that only a small fraction (7) of near-collisions can be detected. Our solution to improve upon this approach is to make use of the theory of covering codes. Let $\mathcal{C}$ be a binary code with length n and with K codewords. Note that in the rest of the paper, the length of a code and the output length of a hash function will both be denoted by n, because they coincide in all our applications.

Definition 3 ([21]). *The* covering radius ρ *of a binary code* $\mathcal{C}$ *is the smallest integer* ρ *such that every vector in* $\mathbb{Z}_2^n$ *is at a distance of at most* ρ *from a codeword of* $\mathcal{C}$*, i.e.,*

$$\rho(\mathcal{C}) = \max_{x \in \mathbb{Z}_2^n} \min_{c \in \mathcal{C}} \mathrm{d}(x, c)\,. \tag{9}$$

In general, if the *minimum distance* of a code is the parameter of interest, we speak of *error-correcting codes*, whereas if the emphasis is on the covering radius, we speak of *covering codes*. For a thorough introduction to covering codes we refer to the monograph [4].

Let H be a hash function of output size n. Let $\mathcal{C}$ be a code of the same length n, size K and covering radius $\rho(\mathcal{C})$ and assume there exists an efficiently computable map g satisfying

$$\begin{aligned} g :\ & \mathbb{Z}_2^n \to \mathcal{C} \\ & x \mapsto c \quad \text{with} \\ & \mathrm{d}(x, c) \leq \rho(\mathcal{C}). \end{aligned} \tag{10}$$

In other words, g maps every vector of $\mathbb{Z}_2^n$ to a codeword at distance $\rho(\mathcal{C})$ or less. For example, the map g can be the decoding map of $\mathcal{C}$, if decoding can be done efficiently. Note however that it is not necessary that g maps every vector to the closest codeword; *any* function satisfying (10) would serve our purpose. This weaker requirement may allow to replace the decoding map by a faster alternative. The approach outlined above is now summarized in Algorithm 4.

We are now in the position to state our main result:

Theorem 2. *If we assume that* $g \circ H$ *acts like a random mapping, in the sense that the expected cycle and tail lengths are the same as for the iteration of a truly random mapping on a space of size* K*, then, Algorithm 4 finds* $2\rho(\mathcal{C})$*-near-collisions for* H *with a complexity of* $\mathcal{O}(\sqrt{K})$ *and with virtually no memory requirements.*

Algorithm 4 Main algorithm to find memoryless near-collisions for H.

Input: Starting point y_0 and hash function H of length n, a code $\mathcal{C}$ of length n, size K, covering radius ρ and decoding function g satisfying (10).
Apply Algorithm 2 to $(g \circ H)$ and y_0
Output: Pair (m, m^*) such that $m \neq m^*$ and $\mathrm{d}(H(m), H(m^*)) \leq 2\rho$

Proof. Assume that we have two inputs m, m^* to the hash function H with $m \neq m^*$. If $g(H(m)) = g(H(m^*))$ is satisfied, we can deduce that

$$\mathrm{d}(H(m), H(m^*)) \leq 2\rho(\mathcal{C}),$$

that is, every collision for $g \circ H$ corresponds to an ϵ-near-collision for H with $\epsilon = 2\rho(\mathcal{C})$.

In order to find these messages m, m^* we can apply Algorithm 2 to the function $g \circ H$ to find indices $i \neq j$ such that $(g \circ H)^i(y_0) = (g \circ H)^j(y_0)$ for some starting point y_0, or in other words, we get m, m^* with $m \neq m^*$ such that $g(H(m)) = g(H(m^*))$. Since g has an output space of size K, the expected complexity of the described method will be $\mathcal{O}(\sqrt{K})$ and there are virtually no memory requirements. ∎

Remark 2. We see that in our setting, the length of the code is determined by the size of the hash digest and the covering radius ρ is determined by the maximum weight that the near-collisions may have. The efficiency of our approach is therefore determined by the size of the code. The task is thus to find a code $\mathcal{C}$ with K as small as possible. However, also the computability of the function g defined in (10) plays a crucial role. An evaluation of g should be efficient when compared to a hash function call. The actual task is thus to find a code $\mathcal{C}$ with given length n and covering radius ρ, such that the size of $\mathcal{C}$ is as small as possible and decoding can be done efficiently.

The task pointed out in the previous remark is a central problem in the field of covering codes. In all of the following we denote by $K(n, \rho)$ the minimum size of a binary code of length n and covering radius ρ and by $k(n, \rho)$ the smallest dimension of a binary linear code of length n and covering radius ρ (that is, for binary linear codes we have $K(n, \rho) = 2^{k(n,\rho)}$).

A well known bound with respect to the size problem is the *Sphere Covering Bound*, which states that $K(n, \rho)$ satisfies

$$K(n, \rho) \geq \frac{2^n}{V(n, \rho)}, \tag{11}$$

where $V(n, \rho)$ is as in (5). Another general bound is due to van Wee (see [4, Theorem 6.4.4]) which states that for $n > \rho$

$$K(n, \rho) \geq \frac{(n - \rho + \mu)2^n}{(n - \rho)V(n, \rho) + \mu V(n, \rho - 1)}, \tag{12}$$

with

$$\mu = (\rho + 1)\lceil \tfrac{n+1}{\rho+1} \rceil - (n + 1)\,.$$

Whenever $\mu \neq 0$, (12) improves over (11). An extensive amount of work in the theory of covering codes is devoted to the improvement of upper and lower bounds on the size of covering codes and to ways to construct codes meeting these bounds (see [25,28], [4, Chapter 6] or [8]). We will discuss some possible constructions in the next section. Finally, we also want to note that covering codes have been mentioned before in the context of (keyed) hash functions in [3]. Furthermore, during the preparation of this manuscript we learned about the interesting paper [9] which treats the connection of covering codes and locality sensitive hashing.

3.4 Hamming Codes and ϵ-Near-Collisions with $\epsilon = 2$ and $\epsilon = 4$

An important class of codes are the *Hamming codes* $\mathcal{H}_r$. Hamming codes are linear codes of length $n = 2^r - 1$, dimension $k = 2^r - 1 - r = n - r$, minimum distance $d = 3$ and covering radius $\rho = 1$. They correct every 1-bit error, decoding can be done very efficiently and they are *perfect codes*. Note that for perfect codes, we have equality in (11), that is, they are optimal covering codes for their respective lengths. There is only one other non-trivial binary perfect code known, namely the Golay code which has length 23, dimension 12 and covering radius 3. In the following, we will write $[n, k]$ code for a linear code of length n and dimension k, i.e., a binary code having 2^k codewords.

Decoding a Hamming code is done via syndrome decoding. In general, syndrome decoding of a binary linear code makes use of a table of size 2^{n-k} which stores to every syndrome s the corresponding coset leader c_s, i.e., the vector in the coset of all vectors having syndrome s of smallest Hamming weight. A vector $y \in \mathbb{Z}_2^n$ is then decoded to $y + c_s$. For Hamming codes, the table of size 2^{n-k} can be omitted since the coset leaders are unique and known for a given syndrome s.

Thus, using a Hamming code in Theorem 2, we get a memoryless algorithm to find near-collisions of weight at most 2 in words of size $n = 2^r - 1$. The performance improvement factor is

$$2^{r/2} = 2^{\log_2(n-1)/2} \approx \sqrt{n},$$

compared to a generic collision search algorithm.

In practice, most common hash functions have an output size of $n = 2^r$. Unfortunately, for such lengths, no perfect codes are known. In view of (11), we therefore have to search for codes leading to the smallest possible bounds for $K(n, \rho)$. In [28] it is shown that

$$K(2^r, 1) = 2^{2^r - r} \text{ for all } r \geq 1. \tag{13}$$

The following lemma will be used frequently throughout the remainder of this section [10]:

Lemma 3 *For linear codes $\mathcal{C}_1 = [n_1, k_1]$ with $\rho(\mathcal{C}_1) = \rho_1$ and $\mathcal{C}_2 = [n_2, k_2]$ with $\rho(\mathcal{C}_2) = \rho_2$, the direct sum of $\mathcal{C}_1$ and $\mathcal{C}_2$ is defined as*

$$\mathcal{C}_1 \oplus \mathcal{C}_2 = \{(c_1, c_2) \,|\, c_1 \in \mathcal{C}_1, c_2 \in \mathcal{C}_2\}.$$

Then, $\mathcal{C}_1 \oplus \mathcal{C}_2$ is a linear code satisfying

$$[n_1 + n_2, k_1 + k_2] \quad \text{and} \quad \rho = \rho_1 + \rho_2.$$

To construct a code meeting the bound (13), one can start with a Hamming code $\mathcal{H}_r$ and extend it. In terms of direct sums of codes, this can be realized by $\mathcal{H}_r \oplus \mathcal{U}_1$, where $\mathcal{U}_\ell = \mathbb{Z}_2^\ell$ is the trivial code of length ℓ. In $\mathcal{U}_\ell$, every word is a codeword, and therefore $\rho(\mathcal{U}_\ell) = 0$. By Lemma 3 we end up with a code of length 2^r, dimension $2^r - r$ and covering radius 1. By (13) we know, that no smaller code with this property exists.

Furthermore, the code $\mathcal{C} = \mathcal{H}_r \oplus \mathcal{U}_1$ from above is as easy to decode as $\mathcal{H}_r$ since decoding can be done for each component of the direct sum. We can summarize:

Proposition 1 *Let H be a hash function of output length $n = 2^r$ for $r \geq 1$. Then, for the approach outlined in Theorem 2, i.e., a generic method capable of finding 2-near-collisions for H, the choice of the code $\mathcal{C} = \mathcal{H}_r \oplus \mathcal{U}_1$ is optimal.*

We can also aim for near-collisions of weight ≤ 4, i.e. we take $\rho = 2$. The sphere covering bound (11) in the case $n = 2^r$ then implies

$$K(2^r, 2) \geq \frac{2^{2^r+1}}{2 + 2^r + 2^{2r}} \geq 2^{2^r - 2r}.$$

We do not have a strict formula like (13) but we can use a result from [10] about linear codes with covering radius 2.

Lemma 4 *Let $k(n, 2)$ be the smallest dimension k such that a binary linear $[n, k]$ code with $\rho = 2$ exists. Then for $n \geq 28$,*

1. *if $2^{j+1} - 4 \leq n < 3 \cdot 2^j - 4$,*

$$k(n, 2) \in \{n - 2j - 2, n - 2j - 1, n - 2j\};$$

2. *if $3 \cdot 2^j - 4 \leq n < 2^{j+2} - 4$,*

$$k(n, 2) \in \{n - 2j - 2, n - 2j - 1\}.$$

When applying Lemma 4 to the special case $n = 2^r$, we end up with

$$k(2^r, 2) \in \{2^r - 2r, 2^r - 2r + 1, 2^r - 2r + 2\}.$$

In order to come as close as possible to the above bounds for $\rho = 2$, we again use the direct sum construction to build the following code:

$$\mathcal{C} = \mathcal{H}_{r-1} \oplus \mathcal{H}_{r-1} \oplus \mathcal{U}_2, \tag{14}$$

that is, the direct sum of two Hamming codes of length $2^{r-1}-1$ and the trivial code of length 2. From Lemma 3 we get that this code has covering radius 2, length $n = 2^r$ and dimension $k = 2^r - 2r + 2$. We have no construction reaching one of the two lower possible dimensions $k = 2^r - 2r + 1$ or $k = 2^r - 2r$ (for $n \geq 128$).

Remark 3. Note that for special cases, better constructions than the direct sum might be available. One example is the *amalgamated direct sum* construction, also introduced in [10].

3.5 Near-Collisions for General n and Higher Weight

The last construction can be further generalized. One way to go is to consider a larger covering radius ρ and another is to consider output lengths n that are not a power of 2. The construction principle we propose does not claim to be optimal, our objective is to use only simple, well understood codes with nice properties. We can safely assume $\rho < \lfloor \frac{n}{4} \rfloor$ when talking about "near"-collisions.

Let ρ be a given covering radius and we consider a hash function H with arbitrary output length n. The idea is now to construct a code of length n by using the direct sum construction with ρ suitable Hamming codes and filling the remaining space with $\mathcal{U}_\ell$ codes. This renders a code, which can be decoded efficiently and we will now prove a result on the size of codes based on the above construction.

For this, we define for $i = 1, 2, \ldots$ the numbers $N_i = 2^i - 1$ to be the possible lengths of the Hamming codes $\mathcal{H}_i$. Let $\mathcal{D} = \{0, 1, \ldots, \rho\}$ be the set of digits. We are interested in digital expansions $x = \sum_{i \geq 1} d_i N_i$ with $d_i \in \mathcal{D}$ and $d_i \neq 0$ for finitely many i. For ease of notation, we will denote by $d \cdot \mathcal{H}_i$ the direct sum of d copies of $\mathcal{H}_i$. Now we can prove the following:

Construction 1 *Let n be given and let $\rho < \lfloor \frac{n}{4} \rfloor$. We now consider digital expansions $\sum_{i \geq 1} d_i N_i$ with $d_i \in \mathcal{D}$ which are smaller or equal to n. For a given expansion $(d_i)_{i \geq 1}$, let $s((d_i)_{i \geq 1})$ denote the difference $n - \sum_{i \geq 1} d_i N_i$. We assume that additionally the following holds:*

$$\begin{aligned} \textstyle\sum_{i \geq 1} d_i &= \rho, \\ \textstyle\sum_{i \geq 1} d_i \cdot i &\text{ is maximal}. \end{aligned} \tag{15}$$

Then, the code

$$\mathcal{C} = \bigoplus_{i \geq 1} d_i \cdot \mathcal{H}_i \oplus \mathcal{U}_{s((d_i)_{i \geq 1})} \tag{16}$$

has length n, covering radius ρ and the dimension of the code is

$$k = n - \sum_{i \geq 1} d_i \cdot i \,.$$

Properties of Construction 1:
Since the construction is based on Hamming codes and the trivial codes $\mathcal{U}_\ell = \mathbb{Z}_2^\ell$, we certainly need ρ Hamming codes in the direct sum construction since the covering radius $\rho(\mathcal{U}_\ell) = 0$. Also multiple $\mathcal{H}_i$'s are allowed, so we choose a combination where the lengths $N_i = 2^i - 1$ of the Hamming codes satisfy $\sum_{i\geq 1} d_i N_i \leq n$. In general, we cannot reach n exactly because of the requirement $\sum_{i\geq 1} d_i = \rho$, that is, the sum of the digits must be exactly ρ. Let $s((d_i)_{i\geq 1})$ be the error we make when approximating n. From Lemma 3 we know that a code that is constructed as in (16) has length

$$\sum_{i\geq 1} d_i N_i + s((d_i)_{i\geq 1}) = n,$$

and that the covering radius is exactly ρ. For the dimension of the code we get

$$\begin{aligned} k &= \sum\nolimits_{i\geq 1} d_i(N_i - i) + s((d_i)_{i\geq 1}) \\ &= \sum\nolimits_{i\geq 1} d_i(N_i - i) + n - \sum\nolimits_{i\geq 1} d_i N_i = n - \sum\nolimits_{i\geq 1} d_i \cdot i, \end{aligned}$$

which is smallest possible if $\sum_{i\geq 1} d_i \cdot i$ is maximal.

Table 1. For given $\rho = 1, \ldots, 5$, the table compares the base-2 logarithms of the complexity of the standard table-based approach (with $\epsilon = 2\rho$) (6), the complexity induced by the van Wee bound (12) and the complexity of our construction (16) for $n = 128, 160$ and 512.

	$n = 128$			$n = 160$			$n = 512$		
ρ	(6)	(12)	(16)	(6)	(12)	(16)	(6)	(12)	(16)
1	57.5	60.5	60.5	73.2	76.3	76.5	247.5	251.5	251.5
2	52.3	57.5	58.0	67.7	73.2	74.0	240.3	247.5	248.0
3	47.8	54.8	56.0	62.8	70.3	71.5	233.8	243.8	245.0
4	43.8	52.3	54.0	58.5	67.7	69.5	227.7	240.3	242.0
5	40.1	50.0	52.5	54.4	65.2	67.5	221.9	237.0	239.5

Remark 4. Since the direct sum of perfect codes is no longer perfect, we obviously get further away from the sphere covering bound (11). Nevertheless, since our construction also has the requirement of being easy to decode, we see no better construction that is more efficient when subject to the same restrictions. When restricting Theorem 2 to codes constructed as direct sums of Hamming codes $\mathcal{H}_r$ and $\mathbb{Z}_2^\ell$, then, Construction 1 provides the optimal solution. Of course, we also have to note the drawback that our construction so far only allows to search for ϵ-near-collisions with even ϵ (since $\epsilon = 2\rho$). This issue will be addressed in the following section.

3.6 Additional Thoughts and Probabilistic Considerations

We want to conclude this section with two observations. First, we want to note that the projection based approach described in Sect. 3.2 can also be seen in the context of our coding based solution. To be more precise, in the projection case, we cover $\mathbb{Z}_2^n$ with $2^{n-\epsilon}$ sets which all have size 2^ϵ. The representative of each set is the vector having zeros in the ϵ positions of the predefined set I and p_I then maps a given vector to its representative. Basically, we have a "code" that does not take into account its parity bits. On the other hand, when we look at Hamming spheres around these representatives of radius ϵ, we clearly observe significant overlap.

There is however a simple idea to improve the projection based approach, namely, by just enlarging the set of indices I that are then set to zero. Assume we want to find ϵ-near-collisions. It is not forbidden to take a set of indices I with $|I| > \epsilon$. A cycle finding method applied to $p_I \circ H$ has an expected complexity of $2^{(n-|I|)/2}$ and clearly finds two messages m, m^* such that $\mathrm{d}(H(m), H(m^*)) \leq |I|$. The probability that these two messages m, m^* satisfy $\mathrm{d}(H(m), H(m^*)) \leq \epsilon$ can be computed to be

$$2^{-|I|} \sum_{i=0}^{\epsilon} \binom{|I|}{i} . \tag{17}$$

If we set for example $|I| = 2\epsilon + 1$, (17) implies that a collision for $p_I \circ H$ is an ϵ-near-collision with probability $1/2$.

For a truly memoryless approach, we can treat multiple runs of the cycle-finding algorithm as independent events. Then, the expected complexity to find an ϵ-near-collision is obtained by multiplying the expected complexity to find a cycle by the expected number of times that we have to run the cycle-finding algorithm, i. e. one over the probability that a single run finds an ϵ-near-collision. In other words, we end up with an expected complexity of

$$2^{\frac{n+|I|}{2}} \left(\sum_{i=0}^{\epsilon} \binom{|I|}{i} \right)^{-1} . \tag{18}$$

Finding the best trade-off for this probabilistic approach corresponds to finding the minimum value of (18) with respect to $|I|$ and given ϵ. In Table 2 we give some optimal combinations of ϵ and $|I|$.

Table 2. Optimal values of $|I|$ to minimize (18) for small values of ϵ.

ϵ	1	2	3	4	5	6	7	8	9	10
$\lvert I\rvert$	2	5	8	11	15	18	21	25	28	32

A similar approach can also be taken for the coding based method to find near-collisions. Assume that Algorithm 4 produces two messages m, m^* such

that $g(y) = g(y')$ with $y = H(m)$ and $y' = H(m^*)$. Now we know for sure that $\mathrm{d}(y, y') \leq 2\rho(\mathcal{C})$ but what do we know about the distribution of $\mathrm{d}(y, y')$?

For general codes, this question is difficult to answer, but for a Hamming code $\mathcal{H}_r$ of length $n = 2^r - 1$ it is fairly easy. Since $\rho(\mathcal{H}_r) = 1$, and $g(y) = g(y')$ implies that both y and y' lie in the same Hamming sphere of radius 1 around some code word, the distance $\mathrm{d}(y, y')$ must be either $0, 1$ or 2 by the triangle inequality. For an ideal hash function, we consider y, y' to be uniformly distributed in $\mathbb{Z}_2^n$. The Hamming sphere $B_1(c)$ contains $n + 1$ elements, namely the codeword c and $c + e_i$ for $i \in \{1, \ldots, n\}$.

Proposition 2 *Let y, y' be taken independently and uniformly from $B_1(c)$ for some codeword $c \in \mathcal{H}_r$ of length $n = 2^r - 1$, then*

$$\mathrm{d}(y, y') = \begin{cases} 0 & \textit{with prob.} \quad \frac{n+1}{(n+1)^2} \\ 1 & \textit{with prob.} \quad \frac{2n}{(n+1)^2} \\ 2 & \textit{with prob.} \quad \frac{n(n-1)}{(n+1)^2} \end{cases}. \tag{19}$$

Proof. There are $(n+1)^2$ possibilities to choose a pair (y, y') in $B_1(c)$. In $n+1$ cases $y = y'$ and thus $\mathrm{d}(y, y') = 0$. In order to have $\mathrm{d}(y, y') = 1$ we need to have either $y = c$ and $y' \neq c$ or vice versa. This explains the second probability. The probability for $\mathrm{d}(y, y') = 2$ results from $y \neq c, y' \neq c$ and $y \neq y'$. ∎

These probabilities can also be used for the codes coming from Construction 1. The distribution of $\mathrm{d}(y, y')$ can then be described as the convolution of ρ distribution functions of the form (19).

For example, the probability that a $2\rho(\mathcal{C})$-near-collision found by a code $\mathcal{C}$ as in (16) is in fact a $(2\rho(\mathcal{C}) - 1)$-near-collision, can now be computed as follows. The probability that $\mathrm{d}(y, y') = 2\rho(\mathcal{C})$ is the product of the probabilities that the restriction of y, y' to every independent Hamming code contributing to the direct sum $\mathcal{C}$, has distance 2. Hence, the probability that $\mathrm{d}(y, y') \leq 2\rho(\mathcal{C}) - 1$ is:

$$1 - \prod_{i \geq 1} \left(\tfrac{(2^i - 1)(2^i - 2)}{2^{2i}} \right)^{d_i} \tag{20}$$

More generally, we can play the same game as with the projection approach and fix the near-collision parameter ϵ. Afterwards, we successively increase the covering radius ρ, follow Construction 1, and compute again the product of the complexity of the cycle finding algorithm in the constructed code and the reciprocal value of the probability, that a resulting $2\rho(\mathcal{C})$-near-collison is in fact an ϵ-near-collision.

Determining a closed expression for the complexity like (18) in the projection case seems out of reach, since the dependence on the tuneable covering radius is too involved. Numerical experiments for relevant values of n and ϵ indicate however, that increasing the covering radius does rarely bring an advantage. We refer to Sect. 4 for a concrete example (Table 3).

4 Illustration: The SHA-3 Candidate TIB3

In this section, we want to demonstrate our novel approach introduced above on a practical example, in this case, a recently proposed hash function. In Sect. 2, we have discussed Theorem 1 about the MD-construction which stated that collisions for the hash function H imply collisions for the compression function h. Sometimes we are also able to convert collisions for h into collisions for H.

4.1 Overview on the SHA-3 Candidate TIB3

TIB3 [17] is an iterated hash function based on the Merkle-Damgård design principle and was proposed as a candidate for the NIST SHA-3 competition [18]. TIB3 comes in two flavors: TIB3-256 and TIB3-512. TIB3-256 processes message blocks of 512 bits and a 256-bit state in order to produce 224 or 256 bits of hash output, whereas TIB3-512 operates on message blocks of 1024 bits, a state of size 512 bits and produces hash values of 384 or 512 bits. Let $m = M_1 \| M_2 \| \cdots \| M_t$ be a t-block message (after padding). Then, the hash value $h = H(m)$ is computed as follows:

$$\begin{aligned} H_0 &= IV_H,\ M_0 = IV_M \\ H_i &= h_{\mathrm{T}}(H_{i-1}, M_i \| M_{i-1}) \quad \text{for } 1 \le i \le t \\ H_{t+1} &= h_{\mathrm{T}}(H_t, 0 \| H_t \| M_t) = h \end{aligned}$$

where IV_H and IV_M are predefined initial values. Note that each message block is used in two compression function calls. The compression function h_{T} is used in Davies-Meyer mode [13] and consists of 2 parts: the key schedule and the state update transformation. The state update of the compression function has 16 rounds, consisting of additions modulo 2^{64}, bitwise exclusive-ORs, bitwise parallel nonlinear functions and fixed rotations. We refer to [17] for a complete description of the hash function.

In the following, we want to focus on TIB3-512 in order to demonstrate our ideas. Basically, our illustration will rely on the attacks presented by Mendel and Schläffer in [14]. We note that the 512 bit and the 256 bit version of TIB3 are closely related since TIB3-512 is more or less a parallel invocation of two TIB3-256 instances. In all of the following, the 512-bit state and hash value are considered as values in $(\mathbb{Z}_2^{128})^4$.

In [14] it was shown that the compression function of TIB3-512 exhibits a similar weakness as the MD5 compression function, see Sect. 2.3. Namely, it is relatively easy to find internal state values and message blocks such that

$$h_{\mathrm{T}}(x_i, m_i) = h_{\mathrm{T}}(x_i + \Delta, m_i), \tag{21}$$

where Δ is one of a set of special difference vectors of the form

$$\Delta = (0, \delta, \delta, \delta) \in (\mathbb{Z}_2^{128})^4. \tag{22}$$

Take one application of the compression function as unit operation. Then, according to [14], the complexity Q to find a solution m_i for (21) is

$$Q = \begin{cases} 2^{24} & \text{if } \delta = e_j \text{ and } j \in \{64, 128\}, \\ 2^{48} & \text{if } \delta = e_j \text{ and } j \in \{1, \ldots, 63, 65, \ldots, 127\} \end{cases}$$

and we have that $Q \leq 2^{240}$ if $\delta = e_{j_1} + e_{j_2} + e_{j_3} + e_{j_4} + e_{j_5}$ and $j_1, j_2, j_3, j_4, j_5 \in \{1, \ldots, 128\}$. For δ vectors with a higher Hamming weight, the complexity becomes larger than 2^{256}, hence worse than the generic complexity of finding a collision in a 512-bit hash function. Note that by this construction we get $\sum_{i=0}^{5} \binom{128}{i} \approx 2^{28}$ different Δ vectors. In [14], near-collisions having a difference vector Δ from this set are constructed by means of a classical birthday attack, hence with a complexity of

$$2^{\frac{512-28}{2}} = 2^{242}$$

compression function evaluations. The memory requirements were reduced to about 2^{100} using the method of distinguished points [23].

4.2 Memoryless Near-Collisions for h_{T}

We will now show how we can construct near-collisions for h_{T} with suitable output difference Δ in a memoryless way. The combination of these near-collisions with the collisions for h_{T} will then result in full collisions.

We first observe that the difference vectors Δ in (22) are of a special form: the nonzero bits occur three times in the same positions. The probability that a randomly found near-collision with weight 15 shows this structure, is negligible.

Therefore, we introduce an additional linear output transformation. We define

$$h_{\mathrm{RT}}(x_i, m_i) = h_{\mathrm{T}}(x_i, m_i) \times \begin{pmatrix} 0\,0\,1\,0 \\ 1\,1\,0\,0 \\ 0\,1\,1\,1 \\ 0\,0\,1\,1 \end{pmatrix}.$$

It can easily be verified that

$$h_{\mathrm{T}}(x_i, m_i) + h_{\mathrm{T}}(x_i, m_i^*) = (0, \delta, \delta, \delta) \tag{23}$$

if and only if

$$h_{\mathrm{RT}}(x_i, m_i) + h_{\mathrm{RT}}(x_i, m_i^*) = (\delta, 0, 0, 0). \tag{24}$$

Now we want to show how to use the approach of Sect. 3.3 to efficiently find near-collisions of the form (24) for the function h_{RT}.

Since $\mathrm{w}(\delta) = 5$, and we have found an almost optimal code of length 2^r with covering radius 2, we will choose the code from (14)

$$\mathcal{C} = \mathcal{H}_6 \oplus \mathcal{H}_6 \oplus \mathcal{U}_2,$$

with length $n = 128$ and $k = 116$. Let g be the decoding function of $\mathcal{C}$ satisfying (10), i.e., g efficiently maps a 128-bit vector to a codeword at distance at most 4. With g, we define the function

$$\begin{aligned} f:\ (\mathbb{Z}_2^{128})^4 &\to (\mathbb{Z}_2^{128})^4 \\ (A,B,C,D) &\mapsto (g(A),B,C,D), \end{aligned} \tag{25}$$

and finally, we can apply a cycle finding method to the function $f \circ h_{\mathrm{RT}}$. Theorem 2 together with the generic collision complexity in the last three 128-bit words lead to a complexity of about

$$2^{\frac{116+3\cdot 128}{2}} = 2^{250} \tag{26}$$

but with virtually no memory requirements. Compared to the attack of [14], at the cost of a higher complexity by a factor 2^8, we can eliminate the memory requirements of 2^{100}. Although an attack with a complexity of 2^{250} is not feasible, NIST stated explicitly in its requirements that a result like this should not exist for SHA-3 [18].

Remark 5. Obviously, by taking $\rho = 2$ we cannot find near-collisions with Hamming weight equal to 5. We can now make use of the discussion in Sect. 3.6. When using $\rho = 3$, we can choose the code $\mathcal{C} = \mathcal{H}_5 \oplus \mathcal{H}_5 \oplus \mathcal{H}_6 \oplus \mathcal{U}_3$ having length 128, dimension $k = 112$ and $\rho = 3$. Then, following from (20) the probability of finding a near-collision of weight ≤ 5 is

$$p = 1 - \left(\frac{(2^5-1)(2^5-2)}{2^{10}}\right)^2 \frac{(2^6-1)(2^6-2)}{2^{12}} \approx 0.2134\,.$$

Replacing $k = 116$ in (26) with $k = 112$ leads to a complexity of 2^{248}, however due to the probability p, we have to repeat the attack p^{-1} times and thus end up again with $\approx 2^{250}$ compression function computations. However, now we can also find near-collisions of weight 5.

As suggested in Sect. 3.6, increasing the covering radius further while still looking for 5-near-collisions would have the effect, that the dimension of the code, and thus, the complexity of the cycle-finding part, decreases. The probability that the found 2ρ-near-collision has weight ≤ 5 however decreases even faster and therefore, the overall complexity increases. Table 3 displays the numerical values of the complexities when using a code of length $n = 128$ and covering radius $\rho \in \{2, 3, \ldots, 9\}$ in the same manner as above:

Table 3. Base-2 logarithm of the complexity of finding 5-near-collisions with codes (16) of length $n = 128$ with increasing covering radii ρ.

ρ	2	3	4	5	6	7	8	9
Compl.	250	250.23	251.24	252.20	253.24	254.34	255.46	256.48

5 Conclusion

In this paper, we have proposed a new memoryless method to search for near-collisions. Our approach is based on the decoding operation of covering codes. The efficiency of our algorithm depends on the size of the underlying code and we gave constructions to find near-collisions of small weight which are optimal, or close to optimal. One merit of our approach is that we do not have to impose any conditions on how the near-collisions look like. We demonstrated our approach on the SHA-3 candidate TIB3, where we showed how to completely eliminate the memory requirements of 2^{100} by a small loss in efficiency. Our method marks the first general approach which makes cycle finding algorithms applicable to the search for near-collisions.

Acknowledgements

The authors wish to thank the anonymous referees, Gaëtan Leurent and Kazumaro Aoki for valuable comments and discussions. The work in this paper has been supported in part by the Research Fund K. U. Leuven, project OT/08/027, in part by the European Commission under contract ICT-2007-216646 (ECRYPT II), in part by the Austrian Science Fund (FWF), project P21936 and in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

References

1. A. Barg and G. D. Forney, Jr. Random codes: minimum distances and error exponents. *IEEE Trans. Inf. Theory*, 48(9):2568–2573, 2002.
2. R. P. Brent. An improved Monte Carlo factorization algorithm. *BIT*, 20(2):176–184, 1980.
3. R. Canetti, R. L. Rivest, M. Sudan, L. Trevisan, S. P. Vadhan, and H. Wee. Amplifying collision resistance: A complexity-theoretic treatment. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Comput. Sci.*, pages 264–283. Springer, 2007.
4. G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein. *Covering codes*, volume 54 of *North-Holland Mathematical Library*. North-Holland Publishing Co., Amsterdam, 1997.
5. H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, editors. *Handbook of elliptic and hyperelliptic curve cryptography*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2006.
6. I. Damgård. A Design Principle for Hash Functions. In G. Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Comput. Sci.*, pages 416–427. Springer, 1989.
7. B. den Boer and A. Bosselaers. Collisions for the Compression Function of MD5. In G. Goos and J. Hartmanis, editors, *EUROCRYPT*, volume 765 of *Lecture Notes in Comput. Sci.* pages 293–304. Springer, 1993.

8. G. Kéri. Tables for bounds on covering codes. `http://www.sztaki.hu/~keri/codes/`. accessed 2010/05/17.
9. D. Gordon, V. Miller, and P. Ostapenko. Optimal hash functions for approximate matches on the n-cube. *IEEE Trans. Inform. Theory*, 56(3):984 –991, 2010.
10. R. L. Graham and N. J. A. Sloane. On the covering radius of codes. *IEEE Trans. Inform. Theory*, 31(3):385–401, 1985.
11. B. Harris. Probability distributions related to random mappings. *Ann. Math. Statist.*, 31:1045–1062, 1960.
12. D. E. Knuth. *The art of computer programming. Vol. 2*. Addison-Wesley Publishing Co., Reading, Mass., third edition, 1997. Seminumerical algorithms, Addison-Wesley Series in Computer Science and Information Processing.
13. S. M. Matyas, C. H. Meyer, and J. Oseas. Generating strong one-way functions with crypographic algorithm. *IBM Technical Disclosure Bulletin*, 27(10A):5658–5659, 1985.
14. F. Mendel and M. Schläffer. On Free-Start Collisions and Collisions for TIB3. In P. Samarati, M. Yung, F. Martinelli, and C. A. Ardagna, editors, *ISC*, volume 5735 of *Lecture Notes in Comput. Sci.*, pages 95–106. Springer, 2009.
15. A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
16. R. C. Merkle. One Way Hash Functions and DES. In G. Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Comput. Sci.*, pages 428–446. Springer, 1989.
17. M. Montes and D. Penazzi. The TIB3 Hash. Submission to NIST, 2008.
18. National Institute of Standards and Technology (NIST). Cryptographic Hash Project, 2007. `http://www.nist.gov/hash-competition`.
19. G. Nivasch. Cycle detection using a stack. *Inf. Process. Lett.*, 90(3):135–140, 2004.
20. National Institute of Standards and Technology (NIST). FIPS-180-2: Secure Hash Standard, August 2002. Available online at `http://www.itl.nist.gov/fipspubs/`.
21. V. Pless. *Introduction to the theory of error-correcting codes*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., New York, third edition, 1998. A Wiley-Interscience Publication.
22. J. M. Pollard. Monte Carlo methods for index computation (mod p). *Math. Comp.*, 32(143):918–924, 1978.
23. J.-J. Quisquater and J.-P. Delescaille. How Easy is Collision Search. New Results and Applications to DES. In G. Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Comput. Sci.*, pages 408–413. Springer, 1989.
24. R. Rivest. RFC1321 - The MD5 Message-Digest Algorithm, 1992.
25. R. Struik. An improvement of the Van Wee bound for binary linear covering codes. *IEEE Trans. Inform. Theory*, 40(4):1280–1284, 1994.
26. P. C. van Oorschot and M. J. Wiener. Improving Implementable Meet-in-the-Middle Attacks by Orders of Magnitude. In N. Koblitz, editor, *CRYPTO*, volume 1109 of *Lecture Notes in Comput. Sci.*, pages 229–236. Springer, 1996.
27. P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999.
28. G. J. M. van Wee. Improved sphere bounds on the covering radius of codes. *IEEE Trans. Inform. Theory*, 34(2):237–245, 1988.
29. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Comput. Sci.*, pages 19–35. Springer, 2005.

30. X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Comput. Sci.*, pages 17–36. Springer, 2005.
31. G. Yuval. How to swindle Rabin? *Cryptologia*, 3(3):187–191, 1979.