Building Hash Functions from Block Ciphers, Their Security and Implementation Properties

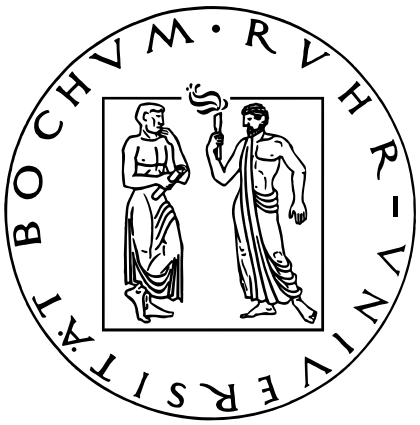
Seminararbeit

Timo Bartkewitz
Ruhr-University Bochum

February 23, 2009

Abstract

This work deals with methods to construct a hash function containing a compression function that is built from a block cipher. There are many schemes to turn a block cipher into a compression function, here the most known are presented including Merkle-Damgård Construction. Such schemes can produce either single-length-block or double-length-block hash functions according to the underlying block cipher with certain properties. At the end security considerations are outlined to convey what signifies a secure hash function that is built from a block cipher.



Chair for Embedded Security
Prof. Dr.-Ing. Christof Paar

Contents

1 Introduction

Nowadays hash functions gained a high level of importance in data traffic. There are several ways to deploy these functions. For downloads from the Internet they could serve as checksums to check the downloaded files for errors. In databases they are used to speed up table lookups, place down new items or to detect duplicated items. But in the first place and in our understanding hash functions are intended for ensuring integrity in cryptographic applications like authentication processes or distributed hashes (hash trees) which are widely used to protect file fragments and unions of those from being modified.

The purpose of a cryptographic hash function (and other hash functions) is easy to understand. It takes a string of arbitrary length as input and maps it to a fixed-length output string. Therefore the output string should be suggested as a digest of the input string. That also implies an essential property, an alteration of at least one character of the input causes a completely different output, thus every output should be unique for every input. For instance somebody obtains a very long text from somebody else and wants to determine whether a third person has altered it unauthorized, if a hash function is used one just have to compare to short strings.
At first sight the necessary functions for the above-mentioned applications seem to be equal, but they are not. We need different hash functions with special properties to meet the demands of each application.

At the moment there are intensive researches on new algorithms. In 2007 the NIST (National Institute of Standards and Technology) announced a challenge[1] to establish a candidate as new standard in cryptographic hash functions. But first of all we have to face one question, why do we need further hash functions? Today there exist many well-known algorithms like the MD and SHA Family, whereas MD5 and SHA-1 are the most used algorithms. Other mentionable functions are RIPEMD and Whirlpool. However, the past has shown that some of them, mainly MD5 (also MD2 and MD4) and SHA-1 (also SHA-0), hold weak spots such one is able to find two different inputs with the same output.

In this work we will concentrate on cryptographic hash functions and how they are built from cryptographic primitives, especially from block ciphers. Building hash functions in such a way is a typical approach due to fact that underlying block ciphers are well reviewed over years. Secondly there are many approved construction methods to use block ciphers in an efficient and simple way.
Primarily we will deal with Merkle-Damgård construction, which embeds an inner function. It formally describes how input string and block cipher are brought together at an abstract level. The inner function is based on one of the methods of *Davies-Meyer, Matyas-Meyer-Oseas, Miyaguchi-Preneel or MDC-2 and MDC-4*. It points out the relation between the input respectively the out-

[1] Cryptographic Hash Algorithm Competition, `http://www.nist.gov/hash-competition`

put of the used block cipher and the block cipher itself.
One challenge in the future will be the security level and finding a design method to derive compact hash functions from a block cipher. Currently there exist many methods, including the above-mentioned, which result in a $2n$-bit to n-bit function taking a n-bit block cipher (n-bit key, n-bit input and n-bit output). Utilizing these methods hash functions are resistant against preimage and second preimage attacks with 2^n operations and against finding collisions with $2^{\frac{n}{2}}$ operations. A newer work by Hirose showed that double-block-length constructions allow security proofs in the ideal cipher model.

2 Hash Functions

This section gives a brief outline about the functional principle, usage and examples of hash functions. In cryptographic understanding the properties of an underlying hash function design are greatly important since they provide information whether the hash function is useful in a certain application. In this work our focus lies on cryptographic hash functions, hence we exemplary introduce one of the most common hash functions (e.g. in signatures, certificates etc.) by now.

2.1 Principle

As mentioned in section 1 the functional principle is easy to understand. The hash function takes a string of arbitrary length and maps it to a fixed-length output referred to as *hash-value* (image). More precisely, for a domain $\mathcal{D}$ and range $\mathcal{S}$ a hash function satisfies the mapping

$$h : \mathcal{D} \rightarrow \mathcal{S}, \text{ whereas } |\mathcal{D}| > |\mathcal{S}| .$$

Figure 1 illustrates that approach with the hash function as black-box. This

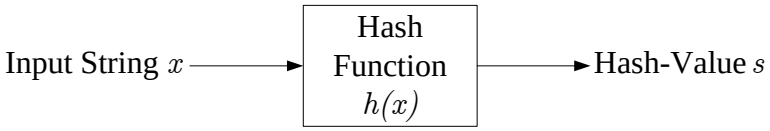


Figure 1: Hash Function Principle

implies input strings that are shorter than expected output string length are stretched and larger strings narrowed to n bits (compression), thus

$$h : x^{(*)} \rightarrow s^{(n)}.$$

Due to the restriction by value n domain $\mathcal{D}$ can be considered being much greater than range $\mathcal{S}$. As a result it is possible that two certain input strings deliver the same output string (collision) but with negligible probability, if n is

big enough.

Such a function can be used to ensure integrity of informations. Once the hash-value of an information is determined by a certain hash function the information can be confirmed as not being altered by applying the hash function again.

2.2 Requirements and Properties

A proper hash function should meet the following requirements and properties [1]:

- **Compression**. Hash function $h(x)$ produces a fixed-length output string s with bit-length n for any given input string k of any arbitrary finite length.
- **Ease of computation**. Every hash-value (output string s) of hash function $h(x)$ is efficient to compute in software and hardware.
- **Preimage resistance**. It is computationally infeasible to find an input string x' (preimage) such that $h(x') = s$ for any given output string s for which corresponding input string x is unknown.
- **Second preimage resistance**. It is computationally infeasible to find an input string x' (second preimage) for any given input string x such that $h(x') = h(x)$.
- **Collision resistance**. It is computationally infeasible to find two distinct input strings x and x' such that $h(x') = h(x)$.
- **Non-correlation**. Input string x and output string s are not correlated in any way. Every bit of input string x affects every bit of output string s.
- **Near-collision resistance**. It is computationally infeasible to find two input strings x and x' such that $h(x)$ and $h(x')$ hardly differ.
- **Partial-preimage resistance**. It is computationally infeasible to find any substring of input string x for any given output string s even for any given distinct substring of input string x.

Here, *computationally infeasible* means solving the underlying problem is not possible within polynomial time or constrained memory. This should outline the practical meaning of some properties.

2.3 Message-Digest Algorithm 5 (MD5)

The cryptographic hash function MD5 [2] was presented by R. Rivest in 1992. It is widely spread and used in signatures and certificates but it is mainly noted for its deployment in checksums to ensure data integrity. The algorithm is the successor of MD4 that has been proved to be insecure [3]. MD5 produces hash-values with a length of 128 bit.
The algorithm is shown in detail in section 3.5 to illustrate the construction of a hash function with given popular methods.
But we definitely not intend to proclaim MD5 as a secure hash function. Quite the opposite latest practical attacks[2] reveal security lacks such it is possible to create rogue Certification Authority certificate which are accepted as valid and trusted by all common web browsers. This approach is possible due to new methods for computing MD5 collisions. A scientific paper containing details is not yet released.

3 Hash Function Constructions from Block Ciphers

Due to the exemplary introduction of MD5 in section 3.5 one would agree that the complexity of latest hash functions is indeed manageable in contrast to other modern cryptographic applications. In the process of designing new hash functions it is desirable to reuse cryptographic components that are already reviewed and established but efficient to implement in hardware and software. Block ciphers could take this place since they provide some appropriate properties.
As hash functions must be able to handle inputs of arbitrary length a method is required to transform these inputs in order to use them with block ciphers whose fixed-length input is much smaller. This approach is represented by the Merkle-Damgård construction.
Additionally, a method is required to embed the block cipher within the Merkle-Damgård construction. However, a straight usage of a block cipher leads to the problem that the input can be recovered easily with aid of the decryption function of the block cipher. Methods to avoid this problem are described afterwards.

Note. Until now it is an open question what requirements of a block cipher are sufficient to construct secure hash functions. A secure block cipher that is provable secure with certain assumptions does not guarantee a secure hash function.

3.1 Merkle-Damgård Construction

The construction was described by R. Merkle in his Ph.D. thesis [4] in 1979. It represents a guidance to build hash functions from compression functions. A block cipher can be considered as a compression function since it transforms two

[2]MD5 considered harmful today, `http://www.win.tue.nl/hashclash/rogue-ca`

inputs (plain text and key) to an output (cipher text) whose length is smaller than the entire length of the two inputs.
I. Damgård proved the most important property of the construction [5].

Fact 1. *Any compression function f which is collision resistant can be extended to a collision resistant hash function $h(x)$.*

Therefore, the problem finding a proper hash function is reduced to finding an appropriate compression function.
Figure 2 illustrates the approach of the construction. The input string x is

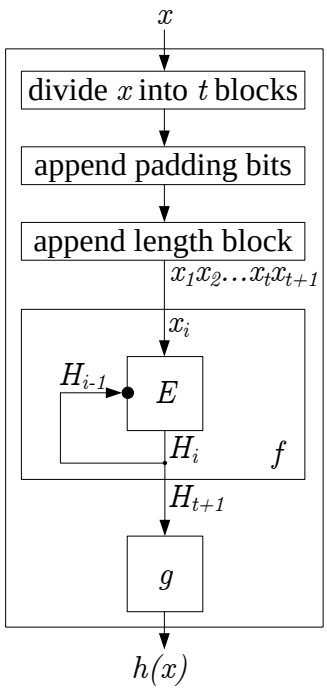


Figure 2: Merkle-Damgård Construction

divided into t equal-sized fixed-length blocks x_i with bit-length r. Bit-length r corresponds to input length of desired compression function f.
To obtain an overall bit-length of x which is a multiple of block length r a padding must be appended. The padding consists of a single 1-bit and as few 0-bits as necessary to reach bit-length r. The single 1-bit and consequently $r-1$ 0-bits are appended even if bit-length of x is already a multiple of r, thus an additional block is created. Otherwise, without including the single 1-bit trailing 0-bits from x cannot be distinguished from a padding containing 0-bits only, hence this will result in the same hash-value.
Due to the proof of *Fact 1* two distinct inputs must not have the same tail

end, hence a length block x_{t+1} with bit-length r is appended holding the right-justified binary representation of overall bit-length of x. This is called the Merkle-Damgård strengthening.
The blocks x_i serves as input to the compression function f that produces an intermediate result H_i after each iteration. H_i serves as feedback value to f and is processed with x_{i+1} in next iteration. This implies the need of an initial value (IV) H_0 for the first iteration that is often provided pre-defined with bit-length r. Choosing a proper function f is considered in section 3.3 and 3.4. As mentioned above f is not the block cipher itself but applies it.
Function g transforms the preliminary result H_{t+1} of bit-length r to the final hash-value with desired bit-length. Function g is often the identity mapping.

3.2 Rate

The rate of an iterated hash function outlines the ratio between the number of block cipher operations and the output. More precisely, if n denotes the output bit-length of the block cipher the rate represents the ratio between the number of processed bits of input x, n output bits and the necessary block cipher operations to produce these n output bits. Generally, the usage of less block cipher operations could result in a better overall performance of the entire hash function but it also leads to a smaller hash-value which is often undesirable. The rate is expressed in the formula

$$R_h = \frac{|x_i|}{s \cdot n},$$

whereas $|x_i|$ denotes the bit-length of input block x_i, s the number of block cipher operations and n the block-length.

3.3 Single-Block-Length Compression Functions

Single-length hash functions output approximately the same number of bits as processed by the underlying block cipher. Considering security concerns the following approaches should be applied if the block cipher already provides a satisfying amount of output bits.

Notation. E_k denotes a generic n-bit block cipher parameterized by a symmetric key k. E can either state the encryption or decryption function of the block cipher. H_0 represents the constant pre-specified initial value IV as mentioned above.

3.3.1 Davies-Meyer

The Davies-Meyer compression function makes a simple use of the underlying block cipher E (Fig. 3). The input blocks x_i serve as the key to E. Thus, the block size of x_i must match the excepted key size of the specific block cipher.

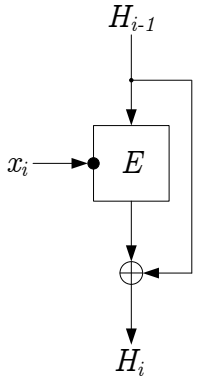


Figure 3: Davies-Meyer Scheme

The previous hash-value H_i serves as the plaintext to be handled with appropriate bit-length.
The output H_i is then concatenated with the previous output H_{i-1} with aid of the exclusive-or operator.

The final output H_t is defined by the iterated formula

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

for $1 \leq i \leq t$, $H_0 = IV$.

The rate of a hash function using Davies-Meyer compression function is

$$R_{DM} = \frac{k}{1 \cdot n} = \frac{k}{n}$$

since the input x serves as the key with bit-length k that could be different from bit-length n of input or output, respectively.

3.3.2 Matyas-Meyer-Oseas

The Matyas-Meyer-Oseas compression function is most widely identical to Davies-Meyer with the input x and the previous hash-value H_i interchanged (Fig. 4). Here, the input blocks x_i serve as plaintext to be handled and the previous hash-value H_i serve as the key to block cipher E. Due to the possible different bit-lengths k and n a function g precedes the key input of E. It maps the n-bit previous hash-value to a suitable k-bit key.

The final output H_t is defined by the iterated formula

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

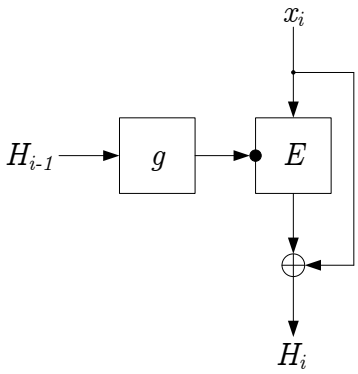


Figure 4: Matyas-Meyer-Oseas Scheme

for $1 \leq i \leq t$, $H_0 = IV$.

The rate of a hash function using Matyas-Meyer-Oseas compression function is

$$R_{MMO} = \frac{n}{1 \cdot n} = 1.$$

In contrast to Davies-Meyer this compression function takes an input bit-length equal to the output bit-length.

3.3.3 Miyaguchi-Preneel

The Miyaguchi-Preneel compression can be considered as an extension of Matyas-Meyer-Oseas. This scheme (Fig. 5) additionally involves the previous hash-value

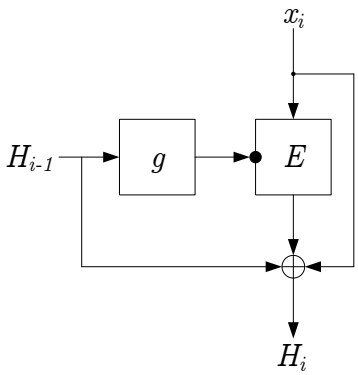


Figure 5: Miyaguchi-Preneel Scheme

H_i in the exclusive-or operator to compute the hash-value H_i.

The final output H_t is defined by the iterated formula

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

for $1 \leq i \leq t$, $H_0 = IV$.

The rate of a hash function using Miyaguchi-Preneel compression function is also

$$R_{MP} = \frac{n}{1 \cdot n} = 1$$

since there are no changes made to the ratio between input and output bit-length.

3.4 Double-Block-Length Compression Functions

Double-length hash functions output approximately twice the number of bits as processed by the underlying block cipher. Considering security concerns the following approaches should be applied if the block cipher provides an unsatisfying amount of output bits.

Fact 2. *If either h_1 or h_2 is a collision resistant hash function, then $h(x) = h_1(x) \,||\, h_2(x)$ is a collision resistant hash function.*

Note. If h_1 and h_2 are applied independently then one could hope finding a collision for $h(x)$ requires twice the effort to find a collision for one of them, if both are collision resistant [1]. Consequently, this stresses the main motivation for double-length constructions.

3.4.1 MDC-2

The MDC-2 (Manipulation Detection Code) algorithm employs two separated iterations of Matyas-Meyer-Oseas scheme. Therefore, this hash function scheme (Fig. 6) outputs a double-block length hash-value relative to the underlying block cipher. It was originally intended for use with DES but is applicable to any desired block cipher that fits this construction. For that reason the functions g and $\tilde{g}$ are meant to produce DES suitable keys. They slightly differ from each other to prevent outputting weak or semi-weak DES keys. The mapping of these functions is as follows.

For input $U = u_1 u_2 u_3 \dots u_{64}$ every eighth bit is deleted starting with u_8. Then the second and third bit is either replaced by 10 for g or 01 for $\tilde{g}$

$$g(U) = u_1 10 u_4 u_5 u_6 u_7 u_9 \dots u_{63},$$

$$\tilde{g}(U) = u_1 01 u_4 u_5 u_6 u_7 u_9 \dots u_{63}.$$

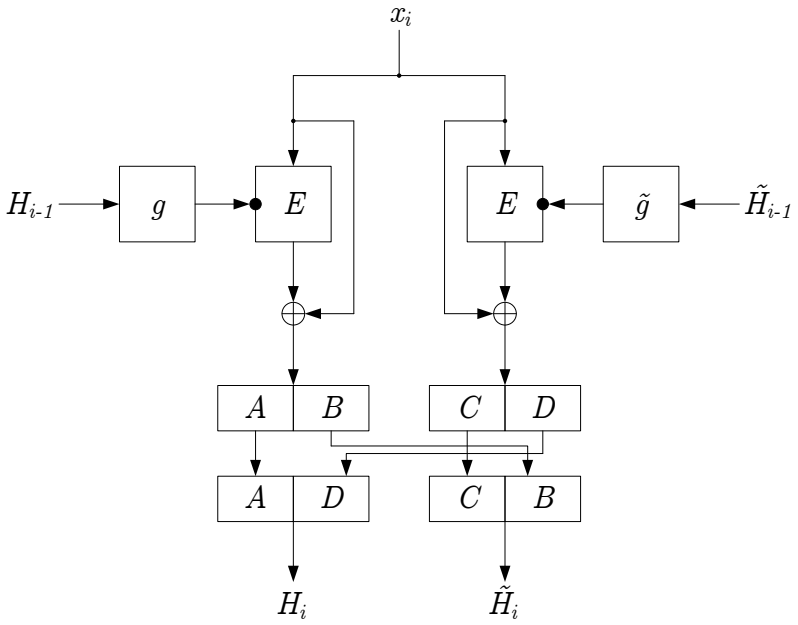


Figure 6: MDC-2 Scheme

However, these functions must be customized to use other block ciphers than DES since they probably require a different bit-length. The initial values feeding the functions g and $\tilde{g}$ are already defined for MDC-2. The IV value in hexadecimal representation is $0x5252525252525252$. The counterpart $\widetilde{IV}$ is the 4-bit shifted IV value $0x2525252525252525$. These values may also require to be customized to serve the desired block cipher.
After each Matyas-Meyer-Oseas iteration the right halves of the two outputs are interchanged. They both H_i and $\tilde{H}_i$ form the preliminary output.

The final output $H_t \,||\, \tilde{H}_t$ is defined by the iterated formula

$$H_i = C_i^L \,||\, \tilde{C}_i^R;\ C_i = E_{g(H_{i-1})}(x_i) \oplus x_i$$

$$\tilde{H}_i = \tilde{C}_i^L \,||\, C_i^R;\ \tilde{C}_i = E_{\tilde{g}(\tilde{H}_{i-1})}(x_i) \oplus x_i$$

for $1 \leq i \leq t$, $H_0 = IV$ and $\tilde{H}_0 = \widetilde{IV}$.

The rate of a hash function using MDC-2 compression function is

$$R_{MDC2} = \frac{n}{2 \cdot n} = \frac{1}{2}$$

since it needs two block cipher operations for one output regardless of the double-length hash-value.

3.4.2 MDC-4

The MDC-4 compression function (Fig. 7) can be considered as an extension of MDC-2. It employs four separated iterations of Matyas-Meyer-Oseas scheme

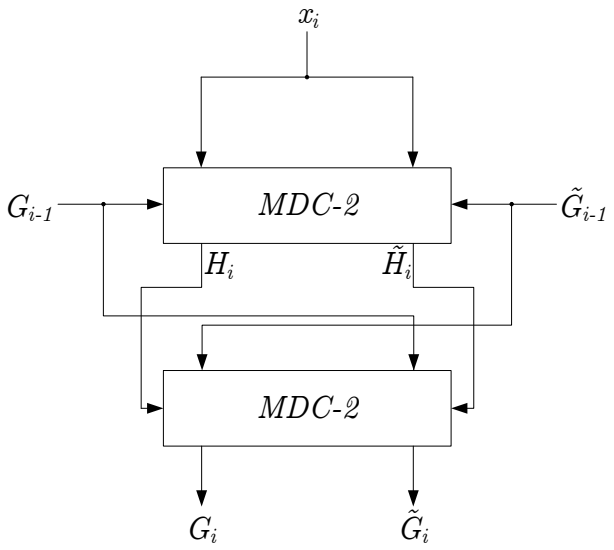


Figure 7: MDC-4 Scheme

respectively two iterations of MDC-2. The two MDC-2 schemes in MDC-4 are connected in a similar way like the two Matyas-Meyer-Oseas schemes in MDC-2. Thereby, the functions g, $\tilde{g}$ and the initial values IV, $\widetilde{IV}$ within MDC-2 remain unchanged.
The upper MDC-2 compression is exactly the same as mentioned in section above but with G_{i-1} and $\tilde{g}_{i-1}$ as feedback values. The resulting preliminary outputs H_i and $\tilde{H}_i$ are put through to the lower MDC-2 compression. Instead of x this time the preliminary output G_{i-1} and $\tilde{G}_{i-1}$ of the overall scheme serve as input.

The final output $G_t \,||\, \tilde{G}_t$ is defined by the iterated formula

$$H_i = C_i^L \,||\, \tilde{C}_i^R;\ C_i = E_{g(G_{i-1})(x_i)} \oplus x_i$$

$$\tilde{H}_i = \tilde{C}_i^L \,||\, C_i^R;\ \tilde{C}_i = E_{\tilde{g}(\tilde{G}_{i-1})}(x_i) \oplus x_i$$

for $1 \leq i \leq t$, $H_0 = IV$ and $\tilde{H}_0 = \widetilde{IV}$ and

$$G_i = D_i^L \,||\, \tilde{D}_i^R;\ D_i = E_{g(\tilde{G}_{i-1})(x_i)} \oplus \tilde{G}_{i-1}$$

$$\tilde{G}_i = \tilde{D}_i^L \,||\, D_i^R;\ \tilde{D}_i = E_{\tilde{g}(G_{i-1})(x_i)} \oplus G_{i-1}$$

for $1 \leq i \leq t$, $G_0 = IV$ and $\tilde{G}_0 = \widetilde{IV}$.

The rate of a hash function using MDC-4 compression function is

$$R_{MDC4} = \frac{n}{4 \cdot n} = \frac{1}{4}$$

since it needs four block cipher operations for one output regardless of the double-length hash-value.

3.4.3 Hirose

The Hirose compression function consists of a $(n, n + |x_i|)$ block cipher plus a permutation p. Figure 8 illustrates this scheme. The bock cipher is instantiated

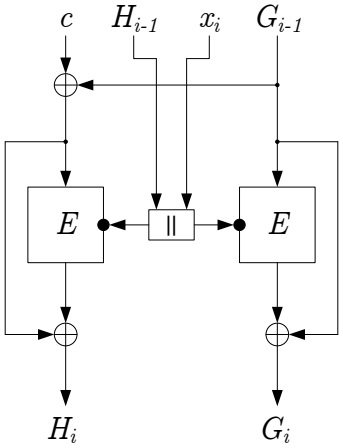


Figure 8: Hirose Scheme

twice with two feedback values H_{i-1}, G_{i-1} and x_i serving as input. The fixed-point free permutation p precedes one of the block cipher instances which most widely follow the Davies-Meyer scheme but with the difference that the key input is a concatenation of feedback value H_{i-1} and input x_i. Permutation p is specified as

$$p(G_{i-1}, H_{i-1}, x_i) = (G_{i-1} \oplus c, H_{i-1}, x_i),$$

whereas c denotes a non-zero constant of bit-length n.

The final output $H_t \,||\, G_t$ is defined by the iterated formula

$$H_i = E_{H_{i-1}||x_i}(p(G_{i-1}, H_{i-1}, x_i)) \oplus H_{i-1}$$

$$G_i = E_{H_{i-1}||x_i}(G_{i-1}) \oplus G_{i-1} \oplus c$$

for $1 \leq i \leq t$, $H_0 = IV$ and $G_0 = \widetilde{IV}$.

The rate of a hash function using Hirose compression function is

$$R_H = \frac{k-n}{2 \cdot n}$$

since the input x_i concatenated with feedback H_{i-1} serves as the key of bit-length $k = n + |x_i|$, thus $|x_i| = k - n$.

3.5 Construction in Detail: Message-Digest Algorithm 5

The MD5 algorithm exactly follows the approach of Merkle-Damgård Construction (Fig. 2). The input x is divided into t 512-bit bocks. The appending of the padding and the length block is combined in one step. First a single 1-bit is appended and as few 0-bits as necessary resulting in a overall bit-length 64 less than a multiple of 512. These last 64 bits are represented by the right-justified 64 bits of the initial overall bit-length of input x.
The MD5 compression function does not exactly follow one of those note above but utilizes a varied version of Davies-Meyer scheme. Hereby, the exclusive-or operation involving the previous preliminary output H_i is replaced by an addition modulo 2^32 (preliminary output is divided into 4 32-bit words).
The block cipher is a dedicated cipher solely designed for use within MD5. It divides the 512 bit block x_i into 16 32-bit words. In four rounds each containing 16 iterations the preliminary output H_i is updated in the following manner schematically shown in Figure 9. The preliminary output is divided into 4 32-

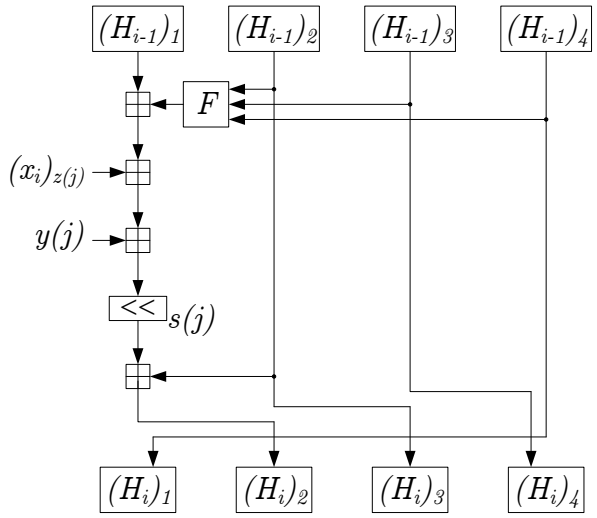


Figure 9: MD5 Block Cipher

bit word. The function F denotes one of four round functions that takes the

second, third and fourth word of H_i. The result is then added to H_i modulo 2^{32}. The lookup table z determines which word of input block x_i is added in the next step. The lookup table y consists of constant 32-bit words derived from the sine function. One of those constants is added prior a leftshift is applied determined by the lookup table s. Finally, the second word of H_i is added and the words are right shifted by one. Further details are describe in [2].

4 Security Consideration

As mentioned before secure block ciphers does not guarantee secure hash functions since even a good block cipher exhibit unobserved weaknesses that could make the hash function insecure.
However, in the matter of hash functions secure signifies that there is no algorithm that perform better on finding collisions than a birthday attack. Hence, a proof of security solely states that the birthday attack is the most efficient attack on the examined hash function.

4.1 Birthday Paradox and Attack

The birthday paradox originally deals with probability that two distinct persons from a group of persons share their birthday. Against all expectations the small number of 23 persons is needed to obtain a probability greater than 50 percent that at least two persons share their birthday.
In the sense of hash functions this fact can be interpreted as the probability that two distinct inputs of a certain hash function share the same hash-value. A hash function whose range is constrained by n holds 2^n distinct hash-values. With birthday paradox in mind a collision can be found after $1.1774 \cdot \sqrt{2^n} = 2^{\frac{n}{2}-0.23}$ trials with probability $\frac{1}{2}$. Since this statement is applicable to any hash function it dictates a lower bound on collision resistance.
One of the first attacks based on the birthday paradox is accordingly called the birthday attack by G. Yuval [7].

4.2 Merkle-Damgård - Structural Weaknesses

The Merkle-Damgård construction suffers from structural weaknesses.

- Given a hash-value $h(x)$ of an unknown input x it is possible to produce a hash-value $h(x \,||\, x')$ of the original input concatenated with any desired extension x'. The extension x' can be considered as additional blocks $x_{t+1}x_{t+2}\ldots$ of original input x. The hash-value $h(x \,||\, x')$ can be produced with feeding the underlying compression function with $h(x)$ and the blocks of x'.

- Following the approach above it is easy to find any number of collisions once a single collision is found. This succeeds since any extension attached to each of the colliding inputs will result in the same hash-value.

- Moreover, A. Joux has shown a multicollision attack in [8] finding 2^r collisions. The attack has complexity slightly increased by factor r as necessary for a birthday attack.

To avoid these weaknesses S. Lucks proposes in [9] a modified structure called the wide-pipe hash. It has a larger internal state and two compression functions. The first compression

$$C' : \{0,1\}^w \times \{0,1\}^m \to \{0,1\}^w$$

is iteratively used in the same manner as in Merkle-Damgård construction but with bit-length w greater than hash-value bit-length n. The second compression

$$C'' : \{0,1\}^w \to \{0,1\}^n$$

produces the final output.
However, this is not a new idea. SHA-384, for instance, uses internal 512-bit states but output a 384-bit hash-value. The SHA-384 is derived from SHA-512.

4.3 Attacks Based on Underlying Block Cipher

Block ciphers may have certain properties with no practical concern when the block cipher is used for encryption but allow manipulation of compression function inputs or prediction and greater control of relations between outputs within the hash function. Properties facilitating such concerns are

- **Complementation property**. $\overline{E_k(x)} = E_{\overline{k}}(\overline{x})$, whereas $\overline{x}$ denotes bitwise complement. This facilitates finding collisions trivially. A slightly modified Matyas-Meyer-Oseas compression function $E_{H_{i-1} \oplus x_i}(x_i) \oplus x_i$, for instance, produces the same output for x and $\overline{x}$.

- **Weak keys**. $E_k(E_k(x)) = x$, for any x. For example, a simplified Davies-Meyer compression function $E_{x_i}(H_{i-1})$ allows creating a two-step fixed point since inserting two blocks x_i containing a weak key leads to the relation $H_i = H_{i-2}$.

- **Semi-weak keys**. $E_{k'}(E_k(x)) = x$, for any x. Here, the concern is similar to that of weak keys.

- **Fixed points**. $E_k(x) = x$, for certain k. For example, the Davies-Meyer compression function $E_{x_i}(H_{i-1}) \oplus H_{i-1}$ produces the output 0^n, if H_{i-1} is a fixed point.

- **Key collisions**. $E_k(x) = E_{k'}(x)$, for certain x. Obviously, this results in hash function collisions.

4.4 Black-Box Analysis

The black-box analysis applies the ideal cipher model where an encryption function $E : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ is assumed to be randomly selected from a set $\mathcal{B}_{n,k}$ containing all such block ciphers. The encryption, respectively, the decryption is simulated by two oracles.
The encryption oracle e receives a pair of a key and a plaintext and, if the plaintext is received for the first time, returns a randomly selected ciphertext. Whereas the decryption oracle e^{-1} receives a pair of a key and a ciphertext and, if the ciphertext is received for the first time, returns a randomly selected plaintext.
They both share a table for these triplets, a pair of the query and corresponding reply, and return the record if a query is received for the second time.

Collision resistance of Hirose scheme. Assuming the ideal cipher model it can be shown that any collision-finding attack on a hash function using Hirose compression function is at most as efficient as the birthday attack.
Two queries are required to compute the output of the compression function but a query to either of them determines to the other due to the permutation p. Thus, they are considered to be a single query.
`FindColHF`$(\mathcal{A}, H)$ is an experiment to measure the collision resistance. $\mathcal{A}$ denotes a collision-finding algorithm.

`FindColHF`$(\mathcal{A}, H)$
$E \leftarrow_R \mathcal{B}_{n,n+|x_i|}$;
$(c, c') \leftarrow_R \mathcal{A}^{e,e^{-1}}$;
`if` $c \neq c' \wedge h_H(c) = h_H(c')$ `return 1; else 0;`

First the block cipher is randomly chosen. $\mathcal{A}$ then makes two randomly chosen queries which results in two replies c and c' by the oracles. `FindColHF` verifies whether a collision has occurred or not.

$\mathbf{Adv}_H^{coll}(q)$ denotes the probability that `FindColHF`$(\mathcal{A}, H)$ returns 1 where $\mathcal{A}$ makes q pairs of queries in total presumed that every pair is made once.
The following theorem depicts the collision resistance.

Theorem 1. *For every query* $1 \leq q \leq 2^{n-2}$

$$\mathbf{Adv}_H^{coll}(q) \leq 3(\frac{q}{2n-1})^2.$$

The proof can be found in [6].

$\mathbf{Adv}_H^{coll}(q) = \frac{1}{2}$ gives

$$q \geq 2^{n-2.3}.$$

Collision resistance of single-block-length schemes. In [10] Black, Rogaway and Shrimpton provide upper and lower bounds on collision resistance, in

the above depicted black-box analysis, of the 64 most basic ways (single-block-length) to construct a hash function considered by B. Preneel, R. Govaerts and J. Vandewalle in [11]. The single-block-length schemes presented in section 3.2 are also considered.

5 Conclusion

The methods to construct hash functions from block ciphers presented in this work are well studied and represent a proper way to develop new hash algorithms. However, in the real world most of common hash functions are partially built from scratch (e.g. MD5, SHA-1) which means following a dedicated design. As illustrated in section 3.5 these hash functions indeed use the Merkle-Damgård construction as a structural frame but virtually none of them contains a compression function based on a well-known block cipher. Whirlpool[3] for example uses the Merkle-Damgård construction combined with the Miyaguchi-Preneel compression function scheme but uses its own dedicated block cipher.
Utilizing the Merkle-Damgård construction is advantageous since it reduces the problem of finding a secure hash function to finding a secure compression function where secure means any collision-finding attack is at most as efficient as the birthday attack. Section 4 outlined that some compression functions satisfy this meaning of being secure with certain assumptions but it is still an open question whether a secure block cipher leads to a secure hash function or not. There still could be unknown weaknesses in a block cipher that makes the hash function vulnerable to collision-finding attacks. Moreover, in the matter of performance block ciphers are handicapped related to dedicated block ciphers since dedicated algorithms are less complex (MD5, section 3.5).
In embedded environments it is important to save resources (program memory, computing power), hence it is useful to reuse already implemented block ciphers for the hash function.

[3] Whirlpool, `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html`

Acronyms

IV Inital Value

DES Data Encryption Standard

MD Message-Digest algorithm

MDC Manipulation Detection Code

NIST National Institute of Standards and Technology

RIPEMD Race Integrity Primitives Evaluation Message-Digest

SHA Secure Hash Algorithm

List of Figures

References

[1] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, Chapter 9, CRC Press, 1997.

[2] R. Rivest, *RFC 1321: The MD5 Message-Digest Algorithm*, 1992.

[3] Y. Naito, Y. Sasaki, N. Kunihiro and K. Ohta, *Improved Collision Attack on MD4*, In: LNCS, vol. 3935, pp. 129-145, Springer, Heidelberg 2006.

[4] R. Merkle, *Secrecy, authentication, and public key systems*, UMI Research Press, 1982.

[5] I. Damgård, *A Design Principle for Hash Functions*. In: Brassard, G. (ed.) CRYPTO 1989, LNCS, vol. 435, pp. 416-427, Springer, Heidelberg 1989.

[6] S. Hirose, *Some Plausible Constructions of Double-Block-Length Hash Functions*. In: Robshaw, M.J.B. (ed.) FSE 2006, LNCS, vol. 4047, pp. 210-225, Springer, Heidelberg 2006.

[7] G. Yuval, *How to Swindle Rabin*, In: Cryptologia, vol. 3, pp. 187-191, 1979.

[8] A. Joux, *Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions*. In: Franklin, M. (ed.) CRYPTO 2004, LNCS, vol. 3152, pp. 306-316, Springer, Heidelberg 2004.

[9] S. Lucks, *Design Principles for Iterated Hash Functions*, In: Cryptology ePrint Archive, Report 2004/253, 2004.

[10] J. Black, P. Rogaway and T. Shrimpton, *Black-Box Analysis of the Block-Cipher Based Hash-Function Constructions from PGV*. In: Yung, M. (ed.) CRYPTO 2002, LNCS, vol. 2442, pp. 320-335, Springer, Heidelberg 2002.

[11] B. Preneel, R. Govaerts and J. Vandewalle, *Hash Functions Based on Block Ciphers: A Synthetic Approach*, In: CRYPTO 1993, LNCS, vol. 773, pp. 368-378, Springer, London 1994.