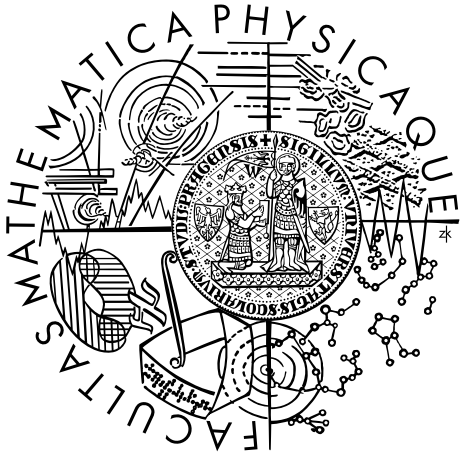
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Bc. Daniel Joščák

Hledání kolizí v hašovacích funkcích

Katedra Algebry
Vedoucí diplomové práce: **Doc. RNDr. Jiří Tůma, DrSc.**
Studijní program: **Matematika**
Studijní obor: **Matematické metody informační bezpečnosti**

Děkuji Doc. RNDr. Jiřímu Tůmovi, DrSc. za cenné rady, náměty, jazykovou úpravu a dlouhé hodiny konzultací, kterými přispěl k napsání diplomové práce. Dále děkuji svým rodičům, protože bez jejich lásky a podpory po dobu mého studia by tato práce nemohla vzniknout a také všem svým přátelům, které jsem po dobu studia poznal.

Prohlašuji, že jsem diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 21. dubna 2006 Daniel Joščák

Contents

Název práce: Hledání kolizí v hašovacích funkcích

Autor: Daniel Joščák

Katedra: Katedra algebry

Vedoucí diplomové práce: Doc. RNDr. Jiří Tůma, DrSc.

E-mail vedoucího: tuma@karlin.mff.cuni.cz

Abstrakt: Hlavním obsahem této práce je hledání kolizí v hašovací funkci MD5. Představíme náš nový algoritmus založený na metodě hledání kolizí podle Wangové a kol. V průběhu psaní této práce Stevens a Klíma publikovali dva nové algoritmy na hledání kolizí v této funkci. Přineseme popis všech tří algoritmů a také jejich výpočetní složitost.

Klíčová slova: MD5, kolize, hašovací funkce

Title: Finding Collisions in Cryptographic Hash Functions

Author: Daniel Joščák

Department: Department of Algebra

Supervisor: Doc. RNDr. Jiří Tůma, DrSc.

Supervisor's e-mail address: tuma@karlin.mff.cuni.cz

Abstract: The main interest of this paper is finding collisions in the hash function MD5. We present our new algorithm based on Wangs et al. methods of finding collisions in MD5. While writing this thesis Stevens and Klima published their fast algorithms for finding collisions. We give a description of these algorithms and the calculation of computianal complexity of all three algorithms.

Keywords: MD5, collision, cryptographic hash functions

Introduction

Hash functions belong among most important cryptographic primitives. A hash function takes as input an arbitrary long binary message and maps it to a binary output of a fixed length. The length of the output message is the *length of the hash function*. The output is called the *hash value* of the input. The hash value is also known as a"digital fingerprint" of the input message. Just like a fingerprint can be used to identify (almost) uniquely a person the hash value of a message can be used to identify (almost) uniquely the message.

There are several requirements a well designed hash function should satisfy. First of all, the algorithm for computing the hash value of a given input should be very fast even for extremely long input messages. On the other hand, the hash function should be "one-way" meaning that it is very difficult to invert it, i.e. to find any input with the prescribed hash value.

Besides various other applications, cryptographic hash functions are also used in digital signature schemes. Every digital signature scheme is based on an asymmetric cipher. The digital signature of a document is obtained by applying the decryption function of the cipher to the document. When used in this context the decryption function is also called the signature function. As asymmetric ciphers are notoriously slow the decryption function is applied only to the hash value of the document rather than to the document itself. This restricts the length of the message to which the decryption function is applied to the length of the hash function used in the digital signature scheme.

However, replacing the document by its hash value in digital signature schemes is not without danger. A digital signature of a message is simultaneously a digital signature of any other message with the same hash value as the original one. Thus it is critical that the hash functions used in digital signature schemes are *collision resistant*, meaning that it is computationally infeasible to find two different messages with the same hash value. Only in this case it is possible to assign uniquely a digital signature to the signed document.

If a hash function is found not to be collision resistant then it does not mean automatically that all digital signatures that used the hash function earlier can now be repudiated. To falsify a signature of a given document requires to find a *second preimage* of the hash value of the document i.e. to find another message with the same hash value as the original document. This is much stronger requirement than just to find any two different messages with the same hash value.

Less than two years ago collision resistance of several widely used hash functions was broken. A group of researchers lead by Prof. X. Wang (of Shandong University, China) presented in [2] collision resistance attacks on MD5 and other hash functions. Since then a lot of research on various as-

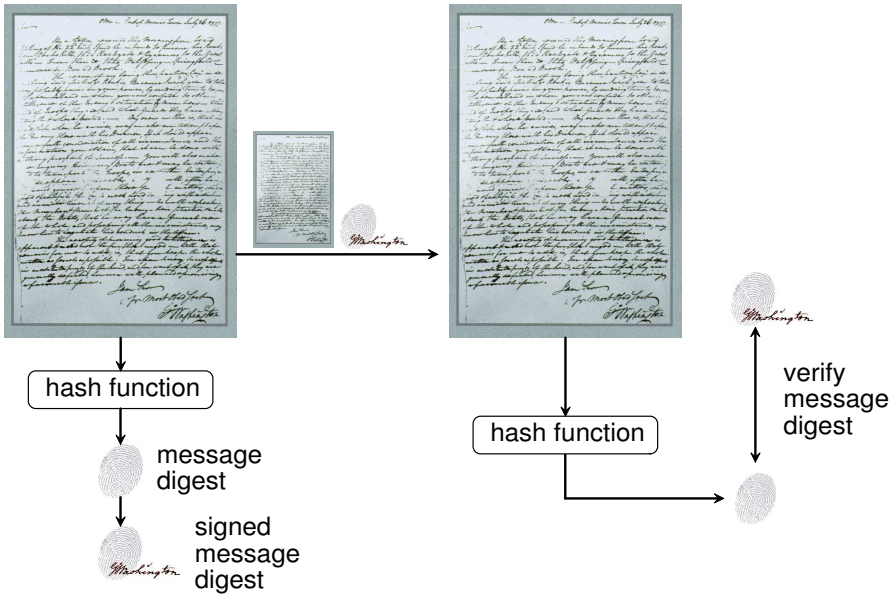


Figure 1: Hash function in a digital signature scheme

pects of the Wang et al. attacks was published, see e.g. [4], [5], [6], [7]. In this master thesis we present various improvements of multi-message modification - the method behind the original Wang et al. algorithm for finding collisions in MD5. We also present a new algorithm which is up to our knowledge the fastest known algorithm based on multi-message modifications. While writing this thesis Stevens [8] and Klima [9] published their fast algorithms for finding collisions in MD5. In their algorithms multi-message modifications are followed by so-called tunelling. The idea of tunelling leads to further improvements in running time of their algorithms. Stevens' algorithm is about twice as fast as our algorithm and the Klima's algorithm is about ten times faster than our. We also present a detailed calculation of the computional complexity of the three algorithms and verify experimentally that our results correspond very well to the actual running times of the algorithms.

1 The MD5 Algorithm

In this section we give a description of the MD5 hash algorithm. We present a slightly modified version to be more suitable for further cryptanalysis and for the description of our algorithm for finding collisions.

The hash function MD5 was designed by Ronald Rivest in 1992. It is an improved version of MD4 and a more detailed description of the algorithm can be found in its specification RFC 1321 [1]. To be consistent we present the entire description of the MD5 hash function.

First, the message is "padded" (extended) so that its length (in bits) is congruent to 448 mod 512. Then the 64-bit representation of the message's length is appended to the result. Now the message length is a multiple of 512 bits and the message can be divided into 512-bit blocks. The initialization vector IV consisting of four 32-bit registers (words) (IV_0, IV_1, IV_2, IV_3) together with the first message block form the input to the compression function, denoted h_{MD5}, of the MD5 algorithm. The output of the compression function are four 32-bit registers. These registers together with the second block of the message form the second input to the compression function. This process continues for all blocks of the message, always using the output of the previous application of the compression function as a new IV together with the next message block as the input to the next calculation of the compression function value. The output of the last computation of the compression function value is the output of the entire hash function. The hash value of MD5 has 128 bits.

Mathematically we can summarize the process of calculating the MD5-hash value of a message m as follows

$$\begin{aligned} MD5(m) &= MD5(m') \\ &= MD5(M^0|M^1|\dots|M^k) \\ &= h_{MD5}(h_{MD5}(\dots h_{MD5}(h_{MD5}(IV, M^0), M^1)\dots), M^{k-1}), M^k), \end{aligned}$$

where m is the input message, m' its extended (padded) version, and $M^0|\dots|M^k$ is the decompositions of m' into blocks of 512 bits.

1.1 The Compression Function

The most important part of the MD5 hash function is its compression function h_{MD5}. It has two inputs, a 128-bit initialization vector $IV = (IV_0, IV_1, IV_2, IV_3)$ and a 512-bit message block M. We will use notation $h_{MD5}(IV, M)$, to denote the value of the compression function h_{MD5} at a given initialization vector IV and a message block M.

First, four registers $R_{-4}, \ldots, R_{-1}$ are initialized by IV in the following way

$$\begin{aligned} R_{-4} &= IV_0 \\ R_{-3} &= IV_3 \\ R_{-2} &= IV_2 \\ R_{-1} &= IV_1 \end{aligned}$$

Then 64 similar steps are executed, in each step the new value of register R_i is computed. This computation is described in the following definition.

Definition 1.1. *An* **MD5 step** *is a computation of the next register R_i in the compression function h_{MD5} from known values of the previous four registers R_{i-4}, R_{i-3}, R_{i-2}, R_{i-1}:*

$$\begin{aligned} Z_i &= f_i(R_{i-1}, R_{i-2}, R_{i-3}) + R_{i-4} + K_i + W_i; \\ Z_i^{\lhd} &= Z_i^{\lll s_i}; \\ R_i &= R_{i-1} + Z_i^{\lhd}, \quad \text{for } 0 \leq i \leq 63. \end{aligned}$$

or equivalently,

$$R_i = R_{i-1} + (f_i(R_{i-1}, R_{i-2}, R_{i-3}) + R_{i-4} + K_i + W_i)^{\lll s_i}$$

The symbols used in this formula have the following meaning:

- The message block M is divided into sixteen 32-bit words, $M_0, ..., M_{15}$. (We use the upper index for indexing the 512-bit message blocks and the lower index for indexing the 32-bit registers.) The message block M consisting of 16 words $M_0, M_1, \ldots, M15$ is expanded into the sequence of 64 words W_i as follows

$$W_i = \begin{cases} M_i, & \text{for } 0 \leq i \leq 15, \\ M_{1+5i \ mod \ 16}, & \text{for } 16 \leq i \leq 31, \\ M_{5+3i \ mod \ 16}, & \text{for } 32 \leq i \leq 47, \\ M_{7i \ mod \ 16}, & \text{for } 48 \leq i \leq 63. \end{cases}$$

- K_i is a pseudorandom constant of length of 32 bits. The values of K_i are given in [1].

- $+$ means the addition modulo 2^{32}, where each word of length 32 bits is interpreted as a binary expression of a natural number $< 2^{32}$.

- $Z_i^{\lll s_i}$ is a left circular shift by s_i bits of the register Z_i, the values of s_i are given in [1].

- f_i is a bitwise nonlinear Boolean function defined as

$$f_i = \begin{cases} F(x,y,z) & = (x \wedge y) \vee (\neg x \wedge z), \quad \text{for } 0 \le i \le 15, \\ G(x,y,z) & = (x \wedge z) \vee (y \wedge \neg z), \quad \text{for } 16 \le i \le 31, \\ H(x,y,z) & = x \oplus y \oplus z, \quad \text{for } 32 \le i \le 47, \\ I(x,y,z) & = y \oplus (x \wedge \neg z), \quad \text{for } 48 \le i \le 63. \end{cases}$$

Remark 1.2. Notice that the calculation of the value of the register R_i depends only on the values of four previous registers $R_{i-4}, \ldots, R_{i-1}$ and one word W_i of the message block M. So it is necessary to keep the values of only four registers R_{i-3}, R_{i-2}, R_{i-1}, R_i at every step of computation. This view of the MD5 calculation is used in its specification RFC 1321[1].

Finally, the new initialization vector is computed as:

$$\begin{aligned} IV_0 &= R_{-4} + R_{60}, \\ IV_3 &= R_{-3} + R_{61}, \\ IV_2 &= R_{-2} + R_{62}, \\ IV_1 &= R_{-1} + R_{63}, \end{aligned}$$

where "+" means addition modulo 2^{32}.

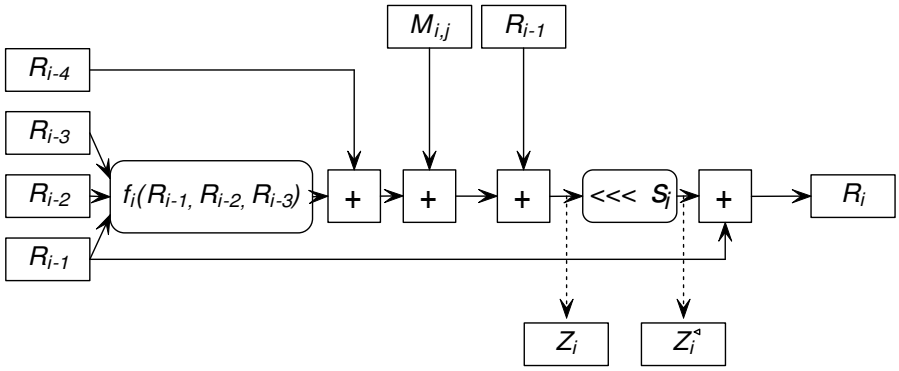


Figure 2: MD5 compression function

1.2 Notation

We shall use two different notations for a sequence A of 32 bits. The sequence A will be called a word or a register. The hexadecimal notation is emphasized by the prefix $0x$. For example

$$0x8000e000 = 8 * 16^7 + 14 * 16^3.$$

The value of A in binary representation will be denoted by

$$[10000000\ 00000000\ 11100000\ 00000000]_2 = 2^{31} + 2^{15} + 2^{14} + 2^{13}.$$

The i-th bit in A is denoted by $A[i]$, where i is the order of the bit. Bits in A are in the order $[A[31] \dots A[0]]_2$. By the symbol $A[30 - 28]$ we denote the sequence of bits $A[30], A[29], A[28]$ of the register A. In the following definition

$$n\ div\ k = \frac{n - (n \mod k)}{k}$$

, for a natural number n and a positive natural number k.

Definition 1.3. *Let* $a = [a[31], a[30], \dots, a[0]]_2$ *and* $b = [b[31], b[30], \dots, b[0]]_2$. *We set* $\tau_{-1} = 0$ *and for* $i = 0, \dots, 31$ *we set*

$$\tau_i \quad = \quad a[i] + b[i] + \tau_{i-1}\ div\ 2.$$

The value of τ_i *is called the carry from position* i *in the sum* $(a + b)$, *for* $-1 \leq i \leq 31$.

Note that

$$(a + b)[i] = a[i] + b[i] + \tau_{i-1} mod\ 2$$

for every $i = 0, \dots, 31$. The carry in the sum of more then two words can be defined similarly.

2 Attack Principle

Wang et al. [2] presented the first pair of colliding messages for MD5 at the rump session of the Crypto 04 conference in August 2004. Later at Eurocrypt 2005 the same authors explained the basic features of their algorithm for finding collisions in MD5 in more detail[3]. Their description consisted of a detailed set of conditions, the so-called differential path, that gives many conditions for the content of registers during the calculations of the MD5-hash value of a message $m = (M^0|M^1)$ consisting of two blocks, each of length of 512 bites. Although Wang at al claimed that their conditions on the content of registers were sufficient for finding another message $n = (N^0|N^1)$ of the same length as m and with the same hash value as M it was shown by Yajima and Shimoyamain in [6], by Liang and Lai in [7] and by Stevens in [8] that Wang's sufficient conditions in fact were not sufficient and further conditions, especially on the form of the sum Z_i used in the calculation of the register R_i were needed for certain indexes i. We will shortly describe the method of Wang et al. [3].

Although the specification of the MD5 hash function uses a fixed initial vector IV, the attack of Wang et al. works for any initial vector. We will denote the initial vector of the attack by IV^0, the first block of the first message by M^0, the second block of the first message by M^1, the first block of the second message by N^0 and the second block of the second message by N^1. To provide two different messages $m = (M^0|M^1)$ and $n = (N^0|N^1)$ with the same hash value it is necessary and sufficient to prove that

$$h_{MD5}(h_{MD5}(M^0, IV^0), M^1) = h_{MD5}(h_{MD5}(N^0, IV^0), N^1),$$

(padding is overlooked since it is the same for both messages m and n, since they have the same length). We set further $IV_m^1 = h_{MD5}(M^0, IV^0)$ and $IV_n^1 = h_{MD5}(N^0, IV^0)$.

The collisions presented by Wang et al. in [2] and [3] as well as all other colliding pairs found by various authors since the first Wang's announcement [2] have the following properties (so- called *differentials*)

$$\Delta^0 = M^0 - N^0 = (0,0,0,0,2^{31},0,0,0,0,0,0,+2^{15},0,0,2^{31},0) \quad (1)$$

$$\Delta^1 = M^1 - N^1 = (0,0,0,0,2^{31},0,0,0,0,0,0,-2^{15},0,0,2^{31},0) \quad (2)$$

$$IV_m^1 - IV_n^1 = (2^{31}, 2^{31}+2^{25}, 2^{31}+2^{25}, 2^{31}+2^{25}). \quad (3)$$

Wang et al. then proceed with giving precise forms for modular differences $R_i - R_i'$ and xor differences $R_i \oplus R_i'$ for $i = 0, \ldots, 63$, where R_i, R_i' are the registers obtained during the calculation $h_{MD5}(IV^0, M^0)$ and $h_{MD5}(IV^0, N^0)$, as well as precise forms of $IV_m^1 \oplus IV_n^1$ and forms of modular and xor differences for registers R_i, R_i' obtained during the calculation of $h_{MD5}(IV_m^1, M^1)$ and $h_{MD5}(IV_n^1, N^1)$. This is what they call the *differential path* for MD5

collision. From the differential path they derived a set of conditions for particular bits in the registers $R_0, \dots R_{63}$, obtained when processing the first block M^0, for particular bits in IV_m^1 and for particular bits in the registers $R_0, \dots R_{63}$, obtained when processing the second block M^1 (i.e. when calculating $h_{MD5}(IV_m^1, M^1)$). They claimed these conditions were sufficient for a message $m = (M^0|M^1)$ satisfying the conditions and another message $n = (M^0 + \Delta^0|N^0 + \Delta^1)$ to have the same hash value. As we stated above the claim about sufficiency of their conditions is false and in fact further conditions on Z_i for some indexes i are needed. However, it should be noted that all collisions in MD5 found so far have the same differential path as described by Wang et al. in [3]. The conditions we used in our algorithms are based on conditions in [7]. The list of all conditions is contained in Tables 31, 32, 33 and 34. In what follows we call them the *prescribed conditions* or briefly the conditions. Notation we use is taken from [8]. For the bit $R_i[j]$, $i = -4, \dots, 63$ and $j = 0, \dots, 31$:

$$R_i[j] = \begin{cases} \texttt{'.'} & \text{if there is no condition on the bit,} \\ \texttt{'0', '1'} & \text{if } R_i[j] \text{must be the value 0 or 1,} \\ \texttt{'\^{}'} & \text{if } R_i[j] \text{must be equal to } R_{i-1}[j], \\ \texttt{'!'} & \text{if } R_i[j] \text{must not be equal to } R_{i-1}[j]. \end{cases}$$

Bits in registers that are not prescribed are called *free* bits. Bits in registers that have a prescribed value are called *fixed* or *prescribed* bits.

2.1 The Principle of our Algorithm

The general method of the attack is to compute messages $M_i,\ 0 \leq i \leq 15$), from the first 16 registers R_i, where $R_i,\ 0 \leq i \leq 15$, satisfies the prescribed conditions. Then we can continue in computing of registers $R_i,\ 16 \leq i \leq 63$, and check (verify) if the prescribed conditions are satisfied. If we find a condition that is not satisfied then we stop the computing and change some bits in R_i, for some $0 \leq i \leq 15$. Then we calculate a new message word M_i for indexes i that were affected by the change in registers R_i and continue in computing and verifying of the conditions for R_i, $i \geq 16$. If all prescribed conditions are satisfied we have found the first (or the second) block of the colliding message m. The other colliding message n is then given by (1) (or (2)).

By the *procedure of verification* or simply the *verification* we will call the process of calculating the values of registers starting from R_{16} and checking the prescribed conditions for R_i, $i \geq 16$. If some condition is not verified then we start changing some bits in R_i for some $0 \leq i \leq 15$. Then we compute the affected message words M_i. By the *generating of a candidate for collision* we mean the process of changing bits in registers and computing affected message words M_i, where by a *candidate for collision* we mean the message block that can be calculated from the modified registers. We want

to note that the computation of affected message words M_i does not have to be done before the procedure of verification is started. In fact, the changes in R_i does not have to affect the M_i needed in calculation of R_{16} (or the first step in the verification procedure). We can use advantage of that fact and calculate the affected message word only after some of the first conditions are checked, what will make our algorithm faster.

The complexity of that kind of attack depends on the number of conditions we need to check and on the probability that these conditions are satisfied. The efficiency of this method can be improved if we can somehow satisfy some of the prescribed conditions for $R_i,\ 16 \leq i \leq 63$. In Section 2.2 we will show how the prescribed conditions for $R_i,\ 16 \leq i \leq 18$ can be satisfied by specific changes in $R_i,\ 12 \leq i \leq 15$. This kind of message modification we use for finding the second colliding block.

For finding the first colliding block we make use of the observation that there are no conditions for registers R_0, R_1 which means we do not have to set their value in advance. The message word M_1 is used message word M_1 in the calculation of R_{16} . This means we can set the value of register R_{16} as we need to and calculate M_1 from the registers $R_i,\ i = 12, \ldots, 16$. Four prescribed conditions for R_{16} are then fulfilled. The message word M_6 is used in the calculation of R_{17}, which can be calculated from $R_i,\ i = 2, \ldots, 6$. Similarly, the message word M_{11} used in calculation of R_{18} can be calculated from $R_i,\ i = 7, \ldots, 11$. This means we can calculate the value of R_{17} and R_{18}, without knowing the values of $R_0,\ R_1$. In Section 2.2 we will show how we can increase the probability that conditions for R_{17} and R_{18} are fulfilled. Now we can choose the value of the register R_{19} and calculate the message word M_0 from $R_i,\ i = 15, \ldots, 19$. Then we can calculate R_0, R_1 and the rest of the message words $M_i, i = 2, \ldots, 5$. Furthermore, after that we can just change bits in R_{19} and verify the conditions for $R_i,\ i = 20, \ldots, 63$ and IV^1. The registers R_{17} and R_{18} will not change. In Section 3.1.2 we present an analysis how to set the value of the register R_{19} that allows us to increase the probability of satisfying the conditions for R_{20} and R_{21} and to specify the probability that condition $Z_{22}[17]$ is satisfied. That means the procedure of verification starts with the calculation of the register R_{20} and the process of generating a new candidate for collision starts with changing of bits in R_{19}.

2.2 Satisfying of the Prescribed Conditions on R_{16}, R_{17} and R_{18}

The method of satisfying prescribed conditions for registers R_{16}, R_{17} and R_{18} we will be explained by the following example. Suppose we want to satisfy the condition $R_i[j] = 0$ or $R_i[j] = 1$. In the equation

$$R_i = R_{i-1} + (G(R_{i-1}, R_{i-2}, R_{i-3}) + R_{i-4} + K_i + W_i)^{\lll s_i}$$

We set :

$$\begin{aligned} Z_i &= (G(R_{i-1}, R_{i-2}, R_{i-3}) + R_{i-4} + K_i + W_i) \\ Z_i^{\triangleleft} &= (Z_i)^{\lll s_i} \\ G_i &= G(R_{i-1}, R_{i-2}, R_{i-3}) \\ KM_i &= K_i + W_i \\ KRM_i &= K_i + R_{i-4} + W_i \\ KGM_i &= K_i + G_i + W_i \end{aligned}$$

Further we denote the bits in these registers as it is in the Tables 1, 2, 3.

$R_{i-1}[j-0]$	$b_j \;\; b_{j-1} \;\; b_{j-2} \;\; \ldots \;\; b_0$
$Z_i^{\triangleleft}[j-0]$	$z_j \;\; z_{j-1} \;\; z_{j-2} \;\; \ldots \;\; z_0$
$R_i[j-0]$	$x_j \;\; x_{j-1} \;\; x_{j-2} \;\; \ldots \;\; x_0$

Table 1: R_i

$R_{i-4}[k-0]$	$a_k \;\; a_{k-1} \;\; a_{k-2} \;\; \ldots \;\; a_0$
$G_i[k-0]$	$g_k \;\; g_{k-1} \;\; g_{k-2} \;\; \ldots \;\; g_0$
$KM_i[k-0]$	$m_k \;\; m_{k-1} \;\; m_{k-2} \;\; \ldots \;\; m_0$
$Z_i[k-0]$	$z_j \;\; z_{j-1} \;\; z_{j-2} \;\; \ldots \;\; z_0$

Table 2: $Z_i,\ k \equiv j - s_i \ mod\ 32$

$R_{i-3}[k-0]$	$d_k \;\; d_{k-1} \;\; d_{k-2} \;\; \ldots \;\; d_0$
$R_{i-2}[k-0]$	$c_k \;\; c_{k-1} \;\; c_{k-2} \;\; \ldots \;\; c_0$
$R_{i-1}[k-0]$	$b_k \;\; b_{k-1} \;\; b_{k-2} \;\; \ldots \;\; b_0$
G_i	$g_k \;\; g_{k-1} \;\; g_{k-2} \;\; \ldots \;\; g_0$

Table 3: $G_i,\ k \equiv j - s_i \ mod\ 32$

We want to satisfy the condition $x_j = 0$ or $x_j = 1$. There are several places where we can influence the value of x_j:

- Bits b_j, b_{j-1}, $b_{j-2}, \ldots$ in R_{i-1} are *free*, meaning there are no prescribed conditions for them. We know:

$$x_j \equiv b_j + z_j + \tau_{j-1} \quad mod\ 2 \tag{4}$$

To choose the right value of b_j (4) we need to know the value of τ_{j-1}. The equality $\tau_{j-1} = v \in \{0, 1\}$ holds if and only if for the largest $l \leq j-1$ such that $b_l = z_l$ we have $b_l = z_l = v$. In most cases we will set $b_{j-1} = z_{j-1}$. Then we immediately know that $\tau_{j-1} = z_{j-1}$ and from (4) we can calculate the unknown b_j.

- Bits b_j, $b_{j-1}, \ldots$ in R_{i-1} are *fixed*, meaning there are prescribed conditions for these bits, so we need to arrange bits z_j, $z_{j-1}, \ldots$ in $Z_i^{\triangleleft}$ so that the equation (4) holds. This can be done in R_{i-4} or in G_i or in W_i. We must find free bits in these registers and change them in a way that we get the desired bits in $Z^{\triangleleft}$.

Example 2.1. We need to arrange bits z_k, z_{k-1}, z_{k-2}, in $Z_i^{\triangleleft}$, to be $(1, 1, 0)$ (or 6 in decimal expansion). Furthermore, the bits $a_k, a_{k-1}, a_{k-2}, a_{k-3}$ in R_{i-4} are free. Denote the sum $G_i + K_i + W_i$ by Y and by $y_k, y_{k-1}, y_{k-2}, y_{k-3}$ the bits $Y[k-(k-3)]$. Then we can write the equation:

$$(4a_k + 2a_{k-1} + a_{k-2}) + (4y_k + 2y_{k-1} + y_{k-2}) + \tau_{k-3} \equiv 6 \mod 8. \qquad (5)$$

If $y_{k-3} = 0$, then we set $a_{k-3} = 0$ and obtain $\tau_{k-3} = 0$. If $y_{k-3} = 1$, then we set $a_{k-3} = 1$ and obtain $\tau_{k-3} = 1$. This allows us to solve (5) for the unknowns a_k, a_{k-1}, a_{k-2}.

The situation when we can change the bits in G_i is similar. We denote again $Y = K_i + R_{i-4} + W_i$, then solve the equation

$$(4g_k + 2g_{k-1} + g_{k-2}) + (4y_k + 2y_{k-1} + y_{k-2}) + \tau_{k-3} \equiv 6 \mod 8$$

for the unknowns g_k, g_{k-1}, g_{k-2} and set the bits in R_{i-1}, R_{i-2} and R_{i-3} so that

$$G(R_{i-1}, R_{i-2}, R_{i-3}) = (g_k, g_{k-1}, g_{k-2}, g_{k-3}).$$

The most difficult situation happens when we need to change some bits in the message register $W_i = M_{i'}$, for $i = 16, 17, 18$ i.e. $i' = 1, 6, 11$, because we do not have enough free bits in R_{i-1}, R_{i-2}, R_{i-3} and R_{i-4}. In this case we need to write down the equation for the computation of $M_{i'}$, $i' = 1, 6, 11$, and see whether we can influence specified bits in $M_{i'}$ by changing of some of the bits in the registers $R_{i'}, \ldots, R_{i'-4}$. An example of changing bits in $M_{i'}$ can be found in 3.2.1, where we attempt to satisfy the condition $R_{16}[3] = R_{15}[3]$.

3 Our Algorithm

The algorithm we present consists of two separated parts. The first one, generates the first block of a colliding message. The second one finds the second block of a message to complete the collision. We begin with the pseudocode of both parts. We recall that the conditions for the verification procedure are described in Appendix B in Tables 31 - 34.

Algorithm 1 Finding the first block

Input: Initial vector IV^0
Output: Pair of messages M^0, N^0, IV_m^1, IV_n^1

1: $R_{-4} \leftarrow IV_0^0$, $R_{-3} \leftarrow IV_3^0$, $R_{-2} \leftarrow IV_2^0$, $R_{-1} \leftarrow IV_1^0$;
2: Initialize $R_2, R_3, \ldots, R_{16}$ with regard to the fixed bits;
3: **repeat**

(a) Randomly change free bits in R_2, R_3, R_7, R_8, R_9, R_{10}, R_{11};

(b) Calculate M_6, M_7, M_8, M_9, M_{10}, M_{11}, M_{12}, R_{17}, R_{18};

(c) **if** one of the conditions for R_{17} and R_{18} is false **then**
- Try to satisfy them by specific changes in registers $R_{13} - R_{16}$
- Calculate R_{17} and R_{18} again;

(d) **if** conditions for R_{17} and R_{18} are true **then**
- Calculate M_{13}, M_{14}, M_{15}, M_1;
- **if** $M_{15}[17] = 0$ **then execute** step 4;

4: Loop over all possible choices in $R_{19}[6-0]$ and $R_{19}[14-8]$

(a) Change R_{19};

(b) Calculate M_0, R_0, R_1, M_5, R_{20}, R_{21};

(c) **if** $R_{20}[31] \neq 0$ *or* $R_{20}[17] \neq R_{19}[17]$ *or* $R_{21}[31] \neq 0$ **then**
- Change some bits in R_{19} to satisfy the conditions;
- Calculate M_0, R_0, R_1, M_5, R_{20}, R_{21} again;

(d) **if** $R_{20}[31] = 0$ & $R_{20}[17] = R_{19}[17]$ & $R_{21}[31] = 0$ **then** (Verify the candidate)
- Calculate $R_{22}, M_4, R_{23}, M_2, M_3, R_{24}, R_{25}, \ldots, IV^1$;
- Verify the conditions for $R_{22}, \ldots, R_{63}, IV_m^1$;
- **if** all conditions are true **then return** $M^0,\ N^0,\ IV_m^1,\ IV_n^1$.

Algorithm 2 Finding the second block

Input: Pair of initial vectors $IV_m^1,\ IV_n^1$ (output from Algorithm 1)
Output: Pair of messages M^1, N^1.

1: $R_{-4} \leftarrow IV_{m,0}^1$, $R_{-3} \leftarrow IV_{m,3}^1$, $R_{-2} \leftarrow IV_{m,2}^1$, $R_{-1} \leftarrow IV_{m,1}^1$
2: Initialize $R_1, R_2, \ldots, R_{15}$ so that the fixed bits are satisfied;
3: Loop until conditions for R_{16}, R_{17}, R'_{17} are true

(a) Randomly change free bits in $R_0, \ldots, R_6$;

(b) Calculate M_1, M_6, R_{16}, R_{17}, R'_{17}, where R'_{17} is calculated from $R_{13}, R'_{14}, R_{15}, R_{16}, M_6$ and R'_{14} is R_{14} with changed bit $R_{14}[17]$;

(c) **if** conditions for R_{16}, R_{17}, are not satisfied **then**
- try to satisfy them by specific changes in R_{12}, R_{13}, R_{14}, R_{15};
- calculate R_{16}, R_{17};
- calculate R'_{17} from $R_{13}, R'_{14}, R_{15}, R_{16}, M_6$, where R'_{14} is R_{14} with changed bit $R_{14}[17]$;

4: Compute M_0, M_2, M_3, M_4, M_5;
5: **repeat** (Generate a new candidate)

(a) Change free bits in R_7, R_8, R_9, R_{10}, R_{11};

(b) Compute M_{11}, R_{18};

(c) **if** $R_{18}[31] \neq 0$ **then** replace R_{14} by R'_{14} and compute R_{18} from $R'_{14}, R_{15}, R_{16}, R'_{17}$;

(d) **if** $R_{18}[17] = 0$ and $Z_{18}[17-3] \neq (1, 1, \ldots, 1)$ **then** (Verify the candidate)
- Compute R_{19}, R_{20}, M_{10}, R_{21}, M_{15}, R_{22}, R_{23}, M_7, M_8, M_9, M_{12}, M_{13}, M_{14}, $R_{24}, \ldots, R_{63}$;
- Verify the conditions for $R_{22}, \ldots, R_{63}$;
- **if** all conditions are true **then return** $M^1,\ N^1$;

3.1 Explanation of Algorithm for the First Block

3.1.1 Step 3.c of Algorithm 1.

We want to satisfy the conditions for R_{17} and R_{18} in this step. The general method of satisfying the conditions is described in 2.2. Here we describe the process of satisfying the conditions for R_{17} and R_{18} in the first block. Notation we use is taken also from 2.2. The conditions to be satisfied are

- $R_{17}[31] = 0$ and $R_{17}[29] = R_{16}[29]$. We have fully under our control the register R_{16}. We will set the bits $R_{14}[31-28] = (0, 0, 0, 0)$ and arrange

the bits $Z_{17}^{\lhd}[31-28] = (0,0,0,0)$. The conditions $R_{17}[31] = 0$ and $R_{17}[29] = R_{16}[29] = 0$ then follow from the equation $R_{17} = R_{16} + Z_{17}^{\lhd}$ as can be seen from Table 4.

	31	30	29	28
R_{16}	0	0	0	0
$Z_{17}^{\lhd}$	0	0	0	0
R_{17}	0	0	0	?

Table 4: $R_{17}[31] = 0$

	22	21	20	19	18
KRM_{16}	y_{22}	y_{21}	y_{20}	y_{19}	y_{18}
R_{14}	**0**	**0**	**0**	**0**	**0**
R_{15}	$\mathbf{c_{22}}$	$\mathbf{c_{21}}$	$\mathbf{c_{20}}$	$\mathbf{c_{19}}$	$\mathbf{c_{18}}$
R_{16}	-	-	-	-	-
Z_{17}	0	0	0	0	?

Table 5: $R_{17}[31] = 0$

We have

$$Z_{17} = (G(R_{16}, R_{15}, R_{14}) + KRM_{17}).$$

To achieve the desired bits in Z_{17} we will change bits in $G(R_{16}, R_{15}, R_{14})$. We set $R_{14}[22-17] = (0,0,0,0,0)$. Then $G(R_{16}, R_{15}, R_{14})[22-17] = R_{15}[22-17]$. Now we need to solve the following equation for the unknowns $c_{22}, c_{21}, c_{20}, c_{29}$

$$(8y_{22} + 4y_{21} + 2y_{20} + y_{19}) + (8c_{22} + 4c_{21} + 2c_{20} + c_{19}) + \tau_{18} \equiv 0 \quad mod\ 16. \quad (6)$$

If $y_{k-4} = 0$, then we arrange $\tau_{18} = 0$ by setting $c_{k-4} = 0$, and if $y_{k-4} = 1$, then we arrange $\tau_{18} = 1$ by setting $c_{k-4} = 1$. Now the equation (6) can be solved. The changes in R_{14} and R_{15} could be done because the bits are free.

- $R_{17}[17] = 1$. We have another prescribed condition $R_{16}[17] = 0$ which causes we can not fulfill the condition for $R_{17}[17]$ only by changes in R_{16}. To control τ_{17} in $R_{16} + Z_{17}^{\lhd}$, we set $Z_{17}^{\lhd}[16] = R_{16}[16]$. Then we can find out, how we need to arrange $Z_{17}^{\lhd}[17]$. Bits $Z_{17}^{\lhd}[17-16]$ can be arranged by changes in G_{17} in the same way as we did for the condition $R_{17}[31] = 0$. The situation is described in Tables 6 and 7.

	17	16
R_{16}	0	b_{16}
$Z_{17}^{\lhd}$	z_{17}	z_{16}
R_{17}	0	.

Table 6: $R_{17}[17] = 1$

	8	7	6
KRM_{16}	y_8	y_7	y_6
R_{14}	**1**	**1**	**1**
R_{15}	.	.	.
R_{16}	$\mathbf{b_8}$	$\mathbf{b_7}$	$\mathbf{b_6}$
$Z_{17}^{\lhd}$	z_{17}	z_{16}	.

Table 7: $R_{17,31} = 0$

• $R_{18}[31] = 0$. To control τ_{30} in $R_{17} + Z_{18}^{\lhd}$, we set $Z_{18}^{\lhd}[30] = R_{17}[30]$. Then we find out how we need to arrange the bit $Z_{18}^{\lhd}[31]$. We arrange the bits $Z_{18}^{\lhd}[31-30]$ in $R_{14}[17-16]$, but this time we need to calculate τ_{15} in $R_{14} + KGM_{18}$ in the algorithm, because it can not be set as we did for the previous conditions. The situation is described in Tables 6 and 7.

	31	30
R_{17}	0	b_{30}
$Z_{18}^{\lhd}$	z_{31}	z_{30}
R_{18}	0	.

Table 8: $R_{18}[31] = 0$

	17	16	15
R_{14}	$\mathbf{a_{17}}$	$\mathbf{a_{16}}$	1
KGM_{18}	y_{17}	y_{16}	y_{15}
$Z_{17}^{\lhd}$	z_{31}	z_{30}	.

Table 9: $R_{18}[31] = 0$

• $R_{18}[17] = 0$. We satisfy this condition in our algorithm only probabilistically with probability $\frac{3}{4}$. That means we increase the probability of fulfilling the condition from obvious $\frac{1}{2}$ to $\frac{3}{4}$. We will try to change bit $Z_{18}^{\lhd}[17]$, what will change the bit $R_{18}[17]$. The change of bits $R_{15}[3] = R_{16}[3]$ will produce the change of $G_{18}[3]$ and $Z_{18}^{\lhd}[17]$ only in case $R_{17}[3] \neq R_{16}[3]$. We suppose that the condition $R_{17}[3] \neq R_{16}[3]$ holds with the probability $\frac{1}{2}$. Then with the probability $\frac{1}{2}$ we can successfully change $R_{18}[17]$. The situation is described in Tables 10 and 11.

	17	16
R_{17}	0	.
$Z_{18}^{\lhd}$	z_{17}	.
R_{18}	0	.

Table 10: $R_{18}[17] = 0$

	3	2	1
KRM_{18}	y_3	.	.
R_{15}	$\mathbf{c_3}$	.	.
R_{16}	$\mathbf{c_3}$	.	.
R_{17}	b_3	.	.
$Z_{18}^{\lhd}$	z_{17}	. run	.

Table 11: $R_{18}[17] = 0$

3.1.2 Step 4. of Algorithm 1.

First, we show how a change in the bit $R_{19}[i]$ affects the registers R_{20}, R_{21}. For simplicity we consider only the case $R_{19}[i] = 0$. Then

$$R'_{19} = R_{19} + 2^i. \tag{7}$$

The other case $R_{19}[i] = 1$ and $R'_{19} = R_{19} - 2^i$ is similar. For the register R'_{20} we have

$$R'_{20} \;=\; \mathbf{R'_{19}} + (R_{16} + G(\mathbf{R'_{19}}, R_{18}, R_{17}) + \mathbf{M'_5} + K_{20})^{\lll 5} \tag{8}$$

Changing R_{19} will affect only the terms highlighted by bold.
- The term R'_{19} in (8) will add difference $+2^i$.

- The term $G(R'_{19}, R_{18}, R_{17})$ in (8) after left shift rotation will add the difference $+2^{i+5}$ in case $R_{17,i} = 1$ and no difference in case $R_{17,i} = 0$. We will denote this as $(+2^{i+5})\chi_{(R_{17}[i]=1)}$, where

$$\chi_{(condition)} = \begin{cases} 1, & condition = TRUE \\ 0, & condition = FALSE \end{cases}$$

The expression $\chi_{(condition)}$ we will call the χ-*coefficient* of the difference $+2^{i+5}$.
- Term M'_5 is calculated from the message word

$$M'_5 = (R_5 - R_4)^{\lll 24} - \mathbf{R'_1} - F(R_4, R_3, R_2) - K_5 \tag{9}$$

where R'_1 is affected by the change in R_{19}. We can write the following equations for other affected registers.

$$\begin{aligned} R'_1 &= \mathbf{R'_0} + (R_{-3} + F(\mathbf{R'_0}, R_{-1}, R_{-2}) + M_1 + K_1)^{\lll 12} && (10) \\ R'_0 &= R_{-1} + (R_{-4} + F(R_{-1}, R_{-2}, R_{-3}) + \mathbf{M'_0} + K_0)^{\lll 7} && (11) \\ M'_0 &= (\mathbf{R'_{19}} - R_{18})^{\lll 12} - R_{15} - G(R_{18}, R_{17}, R_{16}) - K_{19} && (12) \end{aligned}$$

From (7) and (12) we have

$$M'_0 = M_0 + 2^{i+12} \tag{13}$$

and from (11) and (13) we get

$$R'_0 = R_0 + 2^{i+19}. \tag{14}$$

Denote $k = (i + 19)\ mod\ 32$. Then the bit $k + j,\ 0 \le j \le 31$ in the term $F(R'_0, R_{-1}, R_{-2})$ will be changed if the following conditions are satisfied

1. $R_{-1,(k+j)} \neq R_{-2,(k+j)}$.
2. $R'_{0,(k+j)} \neq R_{0,(k+j)}$ i.e. carry in $R_0 + 2^{i+19}$ affected the bit $k + j$.
3. $(k + j) < 32$ i.e. carry in R_0 can affect only bits from position k to position 31.

Using χ-coefficients we can rewrite this fact in the following form

$$\begin{aligned} F(R'_0, R_{-1}, R_{-2}) &= F(R_0, R_{-1}, R_{-2}) \\ &\pm 2^k \chi_{(R_{-1}[k] \neq R_{-2}[k])} \\ &\pm 2^{k+1} \chi_{(R_{-1}[k+1] \neq R_{-2}[k+1])} \chi_{(R_0[k+1] \neq R'_0[k+1])} \chi_{(k+1<32)} \\ &\pm 2^{k+2} \chi_{(R_{-1}[k+2] \neq R_{-2}[k+2])} \chi_{(R_0[k+2] \neq R'_0[k+2])} \chi_{(k+2<32)} \\ &\pm \dots \end{aligned}$$

Then from (10) and (9) it follows

$$\begin{aligned}
R'_1 = R_1 &+ 2^{i+19} \\
&\pm 2^{i+31}\chi_{(R_{-1}[k]\neq R_{-2}[k])} \\
&\pm 2^{i}\chi_{(R_{-1}[k+1]\neq R_{-2}[k+1])}\chi_{(R_0[k+1]\neq R'_0[k+1])}\chi_{(k+1<32)} \\
&\pm 2^{i+1}\chi_{(R_{-1}[k+2]\neq R_{-2}[k+2])}\chi_{(R_0[k+2]\neq R'_0[k+2])}\chi_{(k+2<32)} \\
&\pm \dots \\
M'_5 = M_5 &- 2^{i+19} \\
&\mp 2^{i+31}\chi_{(R_{-1}[k]\neq R_{-2}[k])} \\
&\mp 2^{i}\chi_{(R_{-1}[k+1]\neq R_{-2}[k+1])}\chi_{(R_0[k+1]\neq R'_0[k+1])}\chi_{(k+1<32)} \\
&\mp 2^{i+1}\chi_{(R_{-1}[k+2]\neq R_{-2}[k+2])}\chi_{(R_0[k+2]\neq R'_0[k+2])}\chi_{(k+2<32)} \\
&\mp \dots
\end{aligned}$$

and finally from (8) we get

$$\begin{aligned}
R'_{20} = R_{20} &+ 2^{i} \\
&+ 2^{i+5}\chi_{(R_{17,i}=1)} \\
&- 2^{i+24} \\
&\mp 2^{i+4}\chi_{(R_{-1}[k]\neq R_{-2}[k])} \\
&\mp 2^{i+5}\chi_{(R_{-1}[k+1]\neq R_{-2}[k+1])}\chi_{(R_0[k+1]\neq R'_0[k+1])}\chi_{(k+1<32)} \\
&\mp 2^{i+6}\chi_{(R_{-1}[k+2]\neq R_{-2}[k+2])}\chi_{(R_0[k+2]\neq R'_0[k+2])}\chi_{(k+2<32)} \\
&\mp \dots
\end{aligned} \tag{15}$$

where $k = (i + 19)\ mod\ 32$.

Then we also have

$$R'_{21} = \mathbf{R'_{20}} + (R_{17} + G(\mathbf{R'_{20}}, \mathbf{R'_{19}}, R_{18}) + M_{10} + K_{21})^{\lll 9}. \tag{16}$$

The term R'_{20} will add the same differences as in (15). The value of the nonlinear function G will change at the position j if one of the following conditions is satisfied.

1. $R_{18}[j] = 0$ and $R'_{19}[j] \neq R_{19}[j]$
2. $R_{18}[j] = 1$ and $R'_{20}[j] \neq R_{20}[j]$

Then from (15) we get

$$\begin{aligned}
R'_{21} &= R_{21} + 2^i \\
&+ 2^{i+5}\chi_{(R_{17,i}=1)} \\
&- 2^{i+24} \\
&\mp 2^{i+4}\chi_{(R_{-1}[k]\neq R_{-2}[k])} \\
&\mp 2^{i+5}\chi_{(R_{-1}[k+1]\neq R_{-2}[k+1])}\chi_{(R_0[k+1]\neq R'_0[k+1])}\chi_{(k+1<32)} \\
&\mp 2^{i+6}\chi_{(R_{-1}[k+2]\neq R_{-2}[k+2])}\chi_{(R_0[k+2]\neq R'_0[k+2])}\chi_{(k+2<32)} \\
&\mp \dots \\
&\pm 2^{i+9} \\
&\pm 2^{i+5+9}\chi_{(R_{18}[i+5]=1)}\chi_{(R_{17}[i]=1)} \\
&\pm 2^{i+24+9}\chi_{(R_{18}[i+24]=1)} \\
&\pm 2^{i+4+9}\chi_{(R_{18}[i+4]=1)}\chi_{(R_{-1}[k]\neq R_{-2}[k])} \\
&\pm 2^{i+5+9}\chi_{(R_{18}[i+5]=1)}\chi_{(R_{-1}[k+1]\neq R_{-2}[k+1])}\chi_{(R_0[k+1]\neq R'_0[k+1])}\chi_{(k+1<32)} \\
&\pm 2^{i+6+9}\chi_{(R_{18}[i+6]=1)}\chi_{(R_{-1}[k+2]\neq R_{-2}[k+2])}\chi_{(R_0[k+2]\neq R'_0[k+2])}\chi_{(k+2<32)} \\
&\pm \dots
\end{aligned} \tag{17}$$

From (15) and (17) we see that the term $+2^{i+5}\chi_{(R_{17,i}=1)}$ strongly depends on the bit $R_{17}[i]$. It means if the register R_{17} contains many 0's then this term will not contribute to differences $R'_{20} - R_{20}$ and $R'_{21} - R_{22}$.

The second observation we should make is the fact that the term

$$\mp 2^{i+4}\chi_{(R_{-1}[k]\neq R_{-2}[k])}$$

depends on the registers R_{-1} and R_{-2}, which are words of the initial vector for the attack. If we can choose the IV, then we can set $R_{-1} = R_{-2}$ and avoid all differences of that kind in (15) and (17).

The last observation we should make is the fact that not all the conditions in the χ-coefficients of the differences in (15) and (17) have the same probability. For example the term

$$\mp 2^{i+6}\chi_{(R_{-1}[k+2]\neq R_{-2}[k+2])}\chi_{(R_0[k+2]\neq R'_0[k+2])}\chi_{(k+2<32)}$$

in (15) will add the difference $\mp 2^{i+6}$ if and only if $R_{-1}[i+21] \neq R_{-2}[i+22]$ and there is carry in $R_0 + 2^{i+19}$ from bit $i+19$ to $i+21$, which means that $R_0[i+19]$ and $R_0[i+20]$ both have to be equal 1. If we suppose $R_{-1}[i+21] \neq R_{-2}[i+22]$, then the probability that $R_0[i+19] = R_0[i+20] = 1$ is $\frac{1}{4}$.

Now there are two interesting questions:

1. Which bit in R_{19} do we need to change, in order to change (with high probability) the bits $R_{20}[17]$, $R_{20}[31]$, $R_{21}[31]$?

2. Which bits in R_{19} can we change, in order not to change (with high probability) any of bits $R_{20}[17]$, $R_{20}[31]$, $R_{21}[31]$?

An answer to the first question will help us to increase the probability of satisfying the conditions in R_{20}, R_{21}. From (15) and (17) we choose the bits at positions $7, 25, 22$. We give an overview of most probable changes that these differences bring to registers R_{20} and R_{21} in Table 12. We want to note that in column R_{21} we omit the differences that are the same for R_{20} and R_{21}.

	R_{20}	R_{21}
$\pm 2^7$	$\pm 2^7, \pm 2^{12}, \mp \mathbf{2^{31}}$, $\mp 2^{11}, \mp 2^{12}, \mp 2^{13}$	$\pm 2^{16}, \pm 2^{21}, \pm 2^8$, $\pm 2^{20}, \pm 2^{21}, \pm 2^{22}$
$\pm 2^{25}$	$\pm 2^{25}, \pm 2^{30}, \mp \mathbf{2^{17}}$, $\mp 2^{29}, \mp 2^{30}, \mp 2^{31}$	$\pm 2^2, \pm 2^7, \pm 2^{26}$, $\pm 2^6, \pm 2^7, \pm 2^8$
$\pm 2^{22}$	$\pm 2^{22}, \pm 2^{27}, \mp 2^{14}$, $\mp 2^{26}, \mp 2^{27}, \mp 2^{28}$	$\pm \mathbf{2^{31}}, \pm 2^4, \pm 2^{23}$, $\pm 2^3, \pm 2^4, \pm 2^5$
$+2^0$	$+2^0, +2^5, -2^{24}$, $\mp 2^4, \mp 2^5, \mp 2^6$	$\pm 2^9, \pm 2^{14}, \pm 2^1$, $\pm 2^{13}, \pm 2^{14}, \pm 2^{15}$

Table 12: Change in R_{19}

We claim that:

1. if $R_{20}[31] \neq 0$, then the probability of satisfying the condition $R_{20}[31] = 0$ can be increased by changing the bit $R_{19}[7]$. This change can also change bit $R_{21}[31]$.

2. if $R_{20}[17] \neq R_{19}[17]$, then the probability of satisfying the condition $R_{20}[17] = R_{19}[17]$ can be increased by changing the bit $R_{19}[25]$.

3. if $R_{21}[31] \neq 0$, then the probability of satisfying the condition $R_{21}[31] = 0$ can be increased by changing the bit $R_{19}[22]$.

An intuitive proof of this claim follows from (15) and (17) From the same reason we claim that if a new collision candidate is generated from the register R_{19} by adding difference $+2^0$ to the value of the register R_{19}, then the new candidate will (with a high probability) satisfy the conditions for R_{20} and R_{21}. Better statistical analysis would be required. We decided to generate a new register R_{19} by changing the bits $R_{19}[6-0]$ and $R_{19}[14-8]$. The change is done by adding the difference $+2^0$ to the previous register R_{19}. If the conditions of the registers R_{20} and R_{21} are not satisfied, then we change in the register R_{19} the bits at the positions $i = 7, 22, 25$.

Implementing of the first block of the attack showed that for the original IV the probability of fulfilling the **if** condition in step 4.c was only in 15.78% of cases. That means that the number of calculations $MD5$ *step* in one run of loop 4 Algorithm 1 was in average

$$(0.8422 * 6 + 0.1578 * 12) \doteq 6.9468 \ MD5 \ steps.$$

The **if** condition in step 4.d of Algorithm 1 was fulfilled with the probability approximately 86.67%. That means we generated

$$(0.8667)2^{14} \doteq 14200.0128$$

of collision candidates that were satisfying all prescribed condition for $R_i, i \leq 21$ in the whole loop 4.(2^{14} runs). The total number of $MD5$ *steps* that were needed to create these candidates was $k_0 + (6.9468)2^{14}$, where k_0 is the average number of $MD5$ *steps* calculated in the step 3 of Algorithm 1 before executing the step 4. The average price for generating of one of these candidates is then

$$\frac{k_0 + (6.9468)2^{14}}{14200.0128} \doteq 8.0242 \ MD5 \ steps,$$

where we estemated $k_0 \approx 2^7$.

For a random initial vector was the average price for generating of collision candidate satisfying all prescribed condition for $R_i, i \leq 21$, even smaller - approximately 7.86 $MD5$ *steps*.

Tests for estimating of the probabilities for **if** conditions in the loop 4. were done for more than 2^{33} runs of the loop 4 of Algorithm 1.

3.1.3 Improving the Probability of $Z_{22}[17] = 0$

After implementing the algorithm for the first block we observed that the first condition in the verifications procedure is not satisfied with the expected probability 0.5. In some cases the probability was remarkably higher, in some cases it was remarkably lower. We will explain what was behind this observation.

The condition on $Z_{22}[17] = 0$ is checked only when all the previous prescribed conditions are satisfied. That means $R_{18}[17] = 0$ and we have specified the prescribed condition $R_{19}[17] = R_{20}[17] = 0$. (Saying otherwise we have set bit $R_{19}[17] = 0$). That causes $G(R_{19}, R_{20}, R_{21})[17] = 0$. The message word M_{15} used in the calculation of Z_{22} is not affected by the change in R_{19} and remains in the step 4 the same, but we do not know it's value. We noticed that the probability of the satisfying the condition $Z_{22}[17] = 0$ was remarkably higher than $\frac{1}{2}$ in case $M_{15}[17] = 1$ and remarkably lower than $\frac{1}{2}$ in case $M_{15}[17] = 0$. The situations are described in Tables 13 and 14.

	17	16
R_{18}	0	.
G_{22}	0	.
M_{15}	1	.
K_{22}	0	1
Z_{22}	0	.

Table 13: $Z_{22}[17]$, good situation

	17	16
R_{18}	0	.
G_{22}	0	.
M_{15}	0	.
K_{22}	0	1
Z_{22}	0	.

Table 14: $Z_{22}[17]$, bad situation

The difference is caused by the carry from the 16-th bit τ_{16} in the sum $R_{18} + G_{22} + M_{15} + K_{22}$. For the value of τ_{16} can be calculated from the values of the words $R_{18}, G_{22}, M_{15}, K_{22}$ as

$$\tau_{16} = \big((R_{18} \ mod \ 2^{17}) + (G_{22} \ mod \ 2^{17}) + (M_{15} \ mod \ 2^{17}) + (K_{22} \ mod \ 2^{17})\big) \ div \ 2.$$

We denote by S the sum on the right side of the equation. The value $(K_{22} \ mod \ 2^{17}) = 12454$ in decimal expansion. If we suppose that the values of $R_{18} \ mod \ 2^{17}$, $G_{22} \ mod \ 2^{17}$, $M_{15} \ mod \ 2^{17}$ are "random" i.e. chosen from the uniform distribution, then

$$\tau_{16} = \begin{cases} 0 & \text{if } S \in [124545, 2^{17} - 1] & \doteq 1.66\% \\ 1 & \text{if } S \in [2^{17}, 2_{18} - 1] & \doteq 33.33\% \\ 2 & \text{if } S \in [2^{18}, 2^{18} + 2^{17} - 1] & \doteq 33.33\% \\ 3 & \text{if } S \in [2^{18} + 2^{17}, 517758] & \doteq 31.67\% \end{cases}$$

This fact explaines why the probability of satisfying the condition $Z_{22}[17] = 0$ was higher in case $M_1[15] = 1$ then in case $M_1[15] = 0$. We needed $\tau_{16} = 1$ *or* 3 which has probability approximately 65% in the first case and we needed $\tau_{16} = 0$ *or* 2 which has the probability approximately 35% in the second case. Then we improved our algorithm for the first block by adding the condition $m_{15}[17] = 0$ to the step 3.d of the algorithm for the first block. This condition will ensure that the probability of the condition $Z_{22}[17] = 0$ is satisfied will be 0.65.

3.2 Explanation of Algorithm for the Second Block

3.2.1 Step 3.c of Algorithm 2.

We are satisfying the prescribed conditions for R_{16} and R_{17} in this step. The general method of satisfying the conditions is described in 2.2. Computation of the register R'_{17} is being done to prepare for step 4. and will be described in 3.2.2.

The conditions to be satisfied are

	3	2	1	0
R_{15}	b_3	$\mathbf{b_2}$	$\mathbf{b_1}$	$\mathbf{b_0}$
$Z_{16}^{\triangleleft}$	z_3	z_2	z_1	z_0
R_{16}	x_3			

Table 15: $R_{16,3} = R_{15,3}$

	30	29	28	27
R_{12}	0	1	1	1
G_{16}	?	1	1	1
M_1	m_{30}	m_{29}	m_{28}	m_{27}
K_{16}	1	1	1	0
Z_{16}	z_3	z_2	z_1	z_0

Table 16: Z_{16}

- $R_{16}[3] = R_{15}[3]$. We want to satisfy the condition by changing of bits $R_{15,3-0}$. The situation is described in Table 15.

If $Z_{16}^{\triangleleft}[3-0] \neq (1,0,0,0)$, then we can set the bits b_2, b_1, b_0 so that $b_3 = x_3$. If $Z_{16}^{\triangleleft}[3-0] = (1,0,0,0)$, the condition $R_{16}[3] = R_{15}[3]$ can not be satisfied by changing of b_2, b_1, b_0. In this case ($\frac{1}{16}$ of all possibilities) we need to change at least one bit in $M_1[30-27]$.

$$M_1 = (R_1 - R_0)^{\lll(32-12)} - (F(R_0, R_{-1}, R_{-2}) + R_{-3} + K_1)$$

That means $M_1[30-28]$ can be changed by some change in $R_0[10-8]$ or $R_1[10-8]$. Then we obtain $Z_{16}^{\triangleleft}[3-0] \neq (1,0,0,0)$ and the condition can be satisfied by changing of bits b_2, b_1, b_0.

- $R_{16}[15] = R_{15}[15]$. We satisfy the condition by the specific changes in R_{15} and R_{12} as it is shown in Tables 17 and 18.

	15	14
R_{15}	b_{15}	$\mathbf{0}$
$Z_{16}^{\triangleleft}$	0	0
R_{16}	x_{15}	x_{14}

Table 17: $R_{16}[15] = R_{15}[15]$

	10	9	8
R_{12}	$\mathbf{a_{10}}$	$\mathbf{a_9}$	$\mathbf{a_8}$
KGM_{16}	y_{10}	y_9	y_8
Z_{16}	0	0	?

Table 18: Z_{16}

We set bit $R_{15}[14] = 0$ and then arrange $Z_{16}^{\triangleleft}[15-14] = (0,0)$. Then the condition $R_{16}[15] = R_{15}[15]$ is satisfied. Bits in $Z_{16}^{\triangleleft}[15-14]$ we arrange in $R_{12}[10-8]$ as it is shown in Table 18.

- $R_{16}[17] = 0$. We have free bits $R_{15}[17-16]$, that means we can satisfy the condition by changing of these bits. The situation is shown in Table 19. We set bit $b_{16} = z_{16} = \tau_{16}$. Bit b_{17} is the solution of the following equation

$$b_{17} + z_{17} + \tau_{16} \equiv 0 \mod 2.$$

	17	16
R_{15}	$\mathbf{b_{17}}$	$\mathbf{b_{16}}$
$Z_{16}^{\lhd}$	z_{17}	z_{16}
R_{16}	0	

Table 19: $R_{16}[17] = 0$

• $R_{17}[17] = 1$. According to $R_{16}[16]$ we find out how we need to arrange bits in $Z_{17,17-16}^{\lhd}$. If $b_{16} = 0$, then we need to arrange $Z_{17}^{\lhd}[17-16] = (1, 0)$. If $b_{16} = 1$, then we need $Z_{17}^{\lhd}[17-16] = (0, 1)$. $Z_{17}^{\lhd}[17-16]$ can be arranged by changing $G_{17}[8-6]$. Bits $G_{17}[8-6]$ we arrange by setting $R_{14}[8-6] = (0, 0, 0)$, which causes $G_{17}[8-6] = R_{15}[8-6]$. The situation is described in Tables 20 and 21.

	17	16
R_{16}	0	b_{16}
$Z_{17}^{\lhd}$	z_{17}	z_{16}
R_{17}	1	?

Table 20: $R_{17,17} = 1$

	8	7	6
G_{17}	$\mathbf{g_8}$	$\mathbf{g_7}$	$\mathbf{g_6}$
KRM_{17}	y_8	y_7	y_6
$Z_{17}^{\lhd}$	z_{17}	z_{16}	?

Table 21: Z_{17}

• $R_{16}[31] = 0$. The situation is described in Tables 22 and 23.

	31	30
R_{15}	0	$\mathbf{b_{30}}$
$Z_{16}^{\lhd}$	z_{31}	z_{30}
R_{16}	0	?

Table 22: $R_{16,31} = 0$

	26	25	24
R_{12}	1	1	1
G_{16}	1	0	1
M_1	m_{26}	m_{25}	m_{24}
K_{16}	1	1	0
Z_{16}	z_{31}	z_{30}	?

Table 23: Z_{16}

If $z_{31} = z_{30}$, then the condition can be satisfied by setting $b_{30} = z_{30}$. If $z_{31} \neq z_{30}$, then changing of z_{30} can but don't have to help us. Problem is with the carry bit τ_{29} in the sum $R_{16} + Z_{16}^{\lhd}$. That is why we need to change $Z_{16,31-30}^{\lhd}$. There are no free bits in $R_{12}[26-24]$ and $G_{16}[26-24]$ and we need to change bits in $M_1[26-24]$. It holds for a message word M_1

$$M_1 = (R_1 - R_0)^{\lll(20)} - (F(R_0, R_{-1}, R_{-2}) + R_{-3} + K_1).$$

The change in $M_1[26-24]$ can by done by the change of $R_1[4], R_0[4]$ or in some cases in $R_0[24]$ (it depends on $R_{-1}[24]$ and $R_{-2}[24]$).

• $R_{17}[31] = 0$ & $R_{17}[29] = R_{16}[29]$. We are fulfilling these two conditions at once. The situation is similar to the situation of satisfying $R_{17}[17] = 1$. According to bits $R_{16}[31-26]$ we find out how we need to arrange $Z^{\triangleleft}_{17}[31-27]$. Then we change bits in $G_{17}[22-18]$ to obtain the required bits $Z_{17}[22-18]$. We set bits $R_{14}[22-18] = (0,0,0,0,0)$ and $R_{15}[22-18] = (g_{22}, g_{21}, g_{20}, g_{19}, g_{18})$.

	31	30	29	28
R_{16}	0	b_{30}	b_{29}	b_{28}
$Z^{\triangleleft}_{17}$	z_{31}	z_{29}	z_{29}	z_{28}
R_{17}	0	x_{30}	x_{29}	x_{28}

Table 24: $R_{17}[31] = 0$

	22	21	20	19	18
G_{16}	$\mathbf{g_{22}}$	$\mathbf{g_{21}}$	$\mathbf{g_{20}}$	$\mathbf{g_{19}}$	$\mathbf{g_{18}}$
KRM_{16}	y_{22}	y_{21}	y_{20}	y_{19}	y_{18}
Z_{17}	z_{31}	z_{29}	z_{29}	z_{28}	?

Table 25: Z_{17}

3.2.2 Step 4. of Algorithm 2.

We are generating a new collision candidate for the second block and we are trying to satisfy the condition $R_{18}[31] = 0$ in this step. We know that changing of bits in $R_7, \ldots, R_{11}$ will not affect the equations for M_1 and M_6. The registers R_{16} and R_{17} are computed from $R_{12}, \ldots, R_{15}, M_1$ and from $R_{13}, \ldots, R_{16}, M_6$. That means R_{16} and R_{17} will not change as well.

The computation of $R_{18}[31]$ is described in Tables 26 and 27. The change of bit $R_{14}[17]$ will cause the change of $Z^{\triangleleft}_{18}[31]$ and the change of $Z^{\triangleleft}_{18}[31]$ will cause the change of $R_{18}[31]$ what was our goal.

	31	30
R_{17}	0	b_{30}
$Z^{\triangleleft}_{18}$	z_{31}	z_{30}
R_{18}	0	?

Table 26: $R_{18,31} = 0$

	17	16	15
R_{14}	$\mathbf{a_{17}}$	0	0
KGM_{18}	y_{17}	y_{16}	y_{15}
Z_{18}	z_{31}	z_{30}	?

Table 27: Z_{18}

Now we can see why we calculated the register R'_{17} in step [3.b]. We wanted to be sure that changing of $R_{14}[17]$ will not change the conditions for R_{16} and R_{17}, but it will change $R_{18}[31]$.

$$R_{16} = R_{15} + (G(R_{15}, \mathbf{R_{14}}, R_{13}) + R_{12} + M_1 + K_{16})^{\lll 5}.$$

The function $G(R_{15}, R'_{14}, R_{13})$ will not change because the bit $R_{13}[17] = 1$. That means R_{16} will not change as well.

$$R'_{17} = R_{16} + (G(R_{16}, R_{15}, R'_{14}) + R_{13} + M_6 + K_{17})^{\lll 9}.$$

We don't know how the register R'_{17} will change. To ensure that the prescribed conditions for R'_{17} will not change we did the calculation of R'_{17} in

step 3.c of Algorithm 2. We want to note that R'_{17} will be satisfying the prescribed conditions for R_{17} with very high probability. The change in $R_{14}[17]$ would have to change $G(R_{16}, R_{15}, \mathbf{R_{14}})$ and then the carry in $Z^{\lhd}_{17}$ would have to change bits $Z^{\lhd}_{17}[26-29]$ to change the condition $R_{17}[29] = R_{16}[29]$.

The verifying of the condition $R_{18}[17]$ will cause that in average we will go to the verification procedure (Step 5.d) after the calculation of 4 $MD5$ *steps* (2x M_{11} and 2x R_{18}). Probability that the condition $Z^{\lhd}_{18}[17-3] \neq (1, 1, \ldots, 1)$ is not satisfied is 2^{-15} and can be neglected.

4 Klima's and Stevens' Algorithms

Most recently two new algorithms for collision attack on MD5 were published. The first one was published by Marc Stevens [8], the second one by Vlastimil Klima [9]. First we will present the Stevens' algorithm. The pseudo code is slightly different from Stevens' presentation in [8] but corresponds more exactly to the source code published by Stevens in [8] Appendix B. We must note that the original description is probably better for understanding and the changes we made are minor. On the other hand the changes improve the running time of the algorithm. We also try to give an overview of the Klima's algorithm. The computational complexity of the algorithms will be calculated in the next section.

Both Stevens and Klima use different indexing for registers. The initial vector IV occupies the registers Q_{-3},Q_{-2},Q_{-1},Q_0 and the MD5-compression function then proceeds with calculating the registers $Q_1, \ldots Q_{64}$. To emphasize the different indexing of registers we will use the letter Q when the registers are counted from -3 to 64 and the letter R when they are counted from -4 to 63.

4.1 Stevens' Algorithm

Notation used in [8] is following

$$\begin{aligned} Q_i &= R_{i-1}, \quad 1 \leq i \leq 64, \\ m_i &= M_i, \quad 0 \leq i \leq 15, \\ T_i &= Z_i, \quad 0 \leq i \leq 63. \end{aligned}$$

A new collision candidate in Algorithm 3 is computed in Step 4, where registers Q_9, Q_{10} are changed. Then a message m_{10} is calculated and the verification starts in Step 4.(b). Stevens' Block 2 search algorithm [8] (Section 6) differs from the Block 1 search algorithm only in Step 3 where the loop is repeated until the registers $Q_{17}, \ldots, Q_{21}$ fulfill the conditions. (In the Block 1 search algorithm Q_{17} fullfills the conditions immediately.) This change will extend the duration of Step 3. But because this step is computed only once for 2^{15} collision candidates, it will not extend the duration of the whole algorithm remarkably.

Algorithm 3 Stevens' Block 1 search algorithm

1: Choose $Q_1, Q_3, \ldots, Q_{16}$ fulfilling conditions;
2: Calculate $m_0, m_6, \ldots, m_{15}$;
3: Loop until $Q_{18}, \ldots, Q_{21}$ are fulfilling conditions:

   (a) Choose Q_{17} fulfilling conditions;

   (b) Calculate m_1 from $Q_{16}, Q_{15}, Q_{14}, Q_{13}, Q_{12}$;

   (c) Calculate Q_2 and m_2, m_3, m_4, m_5;

   (d) Calculate $Q_{18}, \ldots, Q_{21}$;

4: Loop over all possible Q_9, Q_{10} satisfying conditions such that m_{11} does not change:

   (a) Calculate m_{10};

   (b) Calculate $Q_{22}, Q_{23}, Q_{24}, \mathbf{m_8}, \mathbf{m_9}, \mathbf{m_{12}}, \mathbf{m_{13}}, Q_{25} \ldots, Q_{64}$;

   (c) Verify conditions for $Q_{22}, \ldots, Q_{64}, T_{22}, T_{34}$ and the iv-conditions for the next block. Stop searching if all conditions are satisfied and a near-collision is verified.

5: Start again at step 1.

4.2 Klima's Algorithm

In the paper "Tunnels in Hash Functions: MD5 Collisions Within a Minute" [9] a new method of *tunneling* is explained. The paper was published in March 2006 in the time of finishing of this text, but we try to present the method by using our terminology. Klima noticed that the method of multi-message modifications as suggested by Wang et al. and improved in this thesis has some limits. Multi-message modifications can help to satisfy some of the conditions for $R_i,\ i \geq 16$, but it is difficult to satisfy all conditions up to R_{23}. He asked himself the following: "Suppose we can find a collision candidate satisfying the conditions up to register R_{23}. Can we change some bits in registers $R_i,\ i = 0, \ldots 15$, and calculate a new collision candidate that will satisfy the conditions for $R_{16}, \ldots, R_{23}$?" The answer is "yes". He specified several collections of bits in $R_0 - R_{19}$ that have the property that a change to the bits does not influence (with high probability) the prescribed conditions for the registers $R_{16}, \ldots, R_{23}$. He calls these collections of bits *tunnels*. According to the numbers of changes that can be done in the tunnel he defines the *strength* of the tunnel. The tunnel of strength n can create 2^n of different collision candidates satisfying all the conditions for $R_{16}, \ldots, R_{23}$. Different tunnels can be composed. If we can obtain 2^r of collision candidates

using a tunnel of strength r and then we apply another tunnel of strength s then we obtain together 2^{r+s} of collision candidates. He further describes different types of tunnels (Section 3). *A probabilistic* tunnel is a tunnel that will create a new collision candidate satisfying the conditions for all $R_i,\ i \leq 23$, with probability of success p. The probability p is called the probability of the tunnel. *A deterministic* tunnel is a tunnel that will create a new collision candidate satisfying the conditions for all $R_i,\ i \leq 23$, with the probability 1. *A dynamic* tunnel is a tunnel in which bits can be changed according to the values of bits in the other registers. We will add a new parameter called the *price* of the tunnel to the description of the tunnels. This parameter will tell us how many calculations of $MD5$ *steps* need to be done to create a new collision candidate in this tunnel. Suppose that T is a tunnel and the number of $MD5$ *steps* that are computed in the calculation of a new collision candidate is n. The price of the tunnel T is an average number of $MD5$ *steps* needed to create a collision candidate satisfying the conditions for all $R_i, i \leq 23$. In the case of some tunnels of probability p, we have $c = p^{-1}n$. In the case of more complicated tunnels, some of the registers affected by the change in the tunnel can be calculated only after we have checked that the change in the tunnel has generated a new collision candidate satisfying the prescribed conditions for all $R_i,\ i \leq 23$. In this case the price of the tunnel is $c = p^{-1}n_1 + n_2$, where n_1 is the number of calculations of $MD5$ *steps* that need to be done to check whether the change in tunnels has generated a new collision candidate satisfying the prescribed conditions for all $R_i,\ i \leq 23$, and n_2 is the number of calculations of $MD5$ *steps* needed to be done because of the change in the tunnel.

We summarize the information about the tunnels given in Section 3 of [9] in the following table, where we named the particular tunnels by T_i for $i = 1, \ldots, 6$. We must note that the numbers in this table are very roughly estimated and further analysis is required. However, we will show the idea of calculation of the complexity of the attack in the next section.

tunnel	strength	probability	price	changed words
T_6 : Q9	3	1	3	M_8, M_9, M_{12}
T_5 : Q4	1	1	3	M_3, M_4, M_7
T_4 : Q14	8	1	8	$M_2, M_3, M_4, R_{23}, M_6$ M_7, M_{13}, M_{14}
T_3 : Q10	2	$\frac{1}{2}$	4*2 + 3	$M_{10}, R_{21}, R_{22}, R_{23},$ M_9, M_{12}, M_{13}
T_2 : Q13	> 10	$\frac{1}{3}$	3*7 + 3	$M_5, R_{20}, R_{21}, M_{15}, R_{22},$ $M_4, R_{23}, M_1, M_2, M_3$
T_1 : Q20	> 3	roughly $\frac{1}{7}$	7*6 + 3	$M_5, R_{20}, R_{21}, R_{22}, M_4,$ R_{23}, M_1, M_2, M_3

Table 28: Tunnels in the first block

Algorithm 4 Klima's algorithm

```
1: Loop until the conditions for R_16, ..., R_23 are fulfilled
2: Loop over all possible changes in T_1
3:    Generate a new candidate for collision
4:    Verify the candidate
5:    Loop over all possible changes in T_2.
6:       Generate a new candidate for collision
7:       Verify the candidate
8:       Loop over all possible changes in T_3
9:          Generate a new candidate for collision
10:         Verify the candidate
11:         Loop over all possible changes in T_4
12:            Generate a new candidate for collision
13:            Verify the candidate
14:            Loop over all possible changes in T_5
15:               Generate a new candidate for collision
16:               Verify the candidate
17:               Loop over all possible changes in T_6
18:                  Generate a new candidate for collision
19:                  Verify the candidate
20:               End of loop T_6
21:            End of loop T_5
22:         End of loop T_4
23:      End of loop T_3
24:   End of loop T_2
25: End of loop T_1
```

5 Algorithm Complexity

We enumerate the prescribed conditions for registers $R_{20}, \ldots, R_{63}$ in first block and we denote by A_j the event that condition number j is true as in Table 29. For the second block we denote by B_j the event that condition number j is true as it is in Table 30. For the conditions on more than one bit in Z_i we used symbol 1^*, that means that all specified bits are 1, and 0^* to mean that all specified bits are 0.

$A_1 : R_{20}[17] = 0$	$A_2 : R_{20}[31] = 0$	$A_3 : R_{21}[31] = 0$
$A_4 : Z_{22}[17] = 0$	$A_5 : R_{22}[31] = 0$	$A_6 : R_{23}[31] = 0$
$A_7 : Z_{34}[15] = 0$	$A_8 : R_{47}[31] = R_{45}[31]$	$A_9 : R_{48}[31] = R_{46}[31]$
$A_{10} : R_{49}[31] \neq R_{47}[31]$	$A_{11} : R_{50}[31] = R_{48}[31]$	$A_{12} : R_{51}[31] = R_{49}[31]$
$A_{13} : R_{52}[31] = R_{50}[31]$	$A_{14} : R_{53}[31] = R_{51}[31]$	$A_{15} : R_{54}[31] = R_{52}[31]$
$A_{16} : R_{55}[31] = R_{53}[31]$	$A_{17} : R_{56}[31] = R_{54}[31]$	$A_{18} : R_{57}[31] = R_{55}[31]$
$A_{19} : R_{58}[31] = R_{56}[31]$	$A_{20} : R_{59}[25] = 0$	$A_{21} : R_{59}[31] \neq R_{57}[31]$
$A_{22} : R_{60}[25] = 1$	$A_{23} : R_{60}[31] = R_{58}[31]$	$A_{24} : R_{61}[25] = 0$
$A_{25} : R_{61}[31] = R_{59}[31]$	$A_{26} : R_{62}[25] = 0$	$A_{27} : R_{62}[31] = R_{60}[31]$
$A_{28} : IV_3^1[25] = 0$	$A_{29} : IV_2^1[25] = 1$	$A_{30} : IV_2^1[26] = 0$
$A_{31} : IV_2^1[31] = IV_3^1[31]$	$A_{32} : IV_1^1[5] = 0$	$A_{33} : IV_1^1[25] = 0$
$A_{34} : IV_1^1[26] = 0$	$A_{35} : IV_1^1[31] = IV_2^1[31]$	

Table 29: Conditions block 1

5.1 Conditions on the Initial Vector

The influence of the initial vector value IV^0 on the running time of collision search algorithm was first described by Stevens in [8] (section 5). There are 8 conditions for the initial vector of the second block IV^1. The initial vector IV^0 depends on the initial vector IV^0 as follows

$$\begin{aligned} IV_0^1 &= IV_0^0 + R_{60}, \\ IV_3^1 &= IV_3^0 + R_{61}, \\ IV_2^1 &= IV_2^0 + R_{62}, \\ IV_1^1 &= IV_1^0 + R_{63}. \end{aligned}$$

Since the conditions $IV_3^1[25] = 0$ and $IV_2^1[25] = 1$ are being verified when the conditions for $R_{61}[25]$, $R_{62}[25]$ are satisfied, the conditions $IV_3^1[25] = 0$

$B_1 : Z_{16}[24-26] \neq 1^*$	$B_2 : R_{16}[3] = R_{15}[3]$	$B_3 : R_{16}[15] = R_{15}[15]$
$B_4 : R_{16}[17] = 0$	$B_5 : R_{16}[31] = 0$	$B_6 : R_{17}[17] = 1$
$B_7 : R_{17}[29] = R_{16}[29]$	$B_8 : R_{17}[31] = 0$	$B_9 : Z_{18}[17-3] \neq 1^*$
$B_{10} : R_{18}[17] = 0$	$B_{11} : R_{18}[31] = 0$	$B_{12} : Z_{19}[31-29] \neq 0^*$
$B_{13} : R_{19}[31] = 0$	$B_{14} : R_{20}[17] = R_{19}[17]$	$B_{15} : R_{20}[31] = 0$
$B_{16} : R_{21}[31] = 0$	$B_{17} : Z_{22}[17] = 0$	$B_{18} : R_{22}[31] = 0$
$B_{19} : R_{23}[31] = 0$	$B_{20} : Z_{34}[15] = 0$	$B_{21} : R_{47}[31] = R_{45}[31]$
$B_{22} : R_{48}[31] = R_{46}[31]$	$B_{23} : R_{49}[31] \neq R_{47}[31]$	$B_{24} : R_{50}[31] = R_{48}[31]$
$B_{25} : R_{51}[31] = R_{49}[31]$	$B_{26} : R_{52}[31] = R_{50}[31]$	$B_{27} : R_{53}[31] = R_{51}[31]$
$B_{28} : R_{54}[31] = R_{52}[31]$	$B_{29} : R_{55}[31] = R_{53}[31]$	$B_{30} : R_{56}[31] = R_{54}[31]$
$B_{31} : R_{57}[31] = R_{55}[31]$	$B_{32} : R_{58}[31] = R_{56}[31]$	$B_{33} : R_{59}[25] = 0$
$B_{34} : R_{59}[31] \neq R_{57}[31]$	$B_{35} : R_{60}[25] = 1$	$B_{36} : R_{60}[31] = R_{58}[31]$
$B_{37} : Z_{61}[21-15] \neq 1^*$	$B_{38} : R_{61}[25] = 1$	$B_{39} : R_{61}[31] = R_{59}[31]$
$B_{40} : R_{62}[25] = 1$	$B_{41} : R_{62}[31] = R_{60}[31]$	$B_{42} : R_{63}[25] = 1$

Table 30: Conditions block 2

and $IV_2^1[25] = 1$ depend on the values of IV_3^0 and IV_2^0. Stevens proposed to add conditions $IV_3^0[25] = IV_3^0[24]$ and $IV_2^0[25] \neq IV_2^0[24]$. We present the calculation of probability of the event $A_{28} : IV_3^1[25] = 0$ given that all events A_i, $i < 28$, have already occurred and the calculation of probability of the event $A_{29} : IV_2^1[25] = 1$ given that all events A_i, $i < 28$, have already occurred. The calculation is based on the following proposition.

Proposition 5.1. *The sum of two registers and the carry bit.*

i. Let $A = [a_{31}, a_{30}, \ldots, a_0]_2$ and $B = [b_{31}, b_{30}, \ldots, b_0]_2$ be a random 32-bit long registers. Bits a_i, $0 \leq i \leq 31$, and b_i $0 \leq i \leq 31$ are independent and chosen from the uniform distribution. Denote by $\tilde{p}_i$ the probability that $\tau_i = 0$ in the sum $A + B$ and denote by $\tilde{q}_i = 1 - \tilde{p}_i$, the probability that $\tau_i = 1$ in the sum $A + B$. Then

$$\tilde{p}_i = \left(\sum_{j=0}^{i-1} \left(\frac{1}{2}\right)^j \left(\frac{1}{4}\right) \right) + \left(\frac{1}{2}\right)^i \left(\frac{3}{4}\right) = \frac{1}{2} + \left(\frac{1}{2}\right)^{i+2} \quad (18)$$

$$\tilde{q}_i = \left(\sum_{j=0}^{i} \left(\frac{1}{2}\right)^j \left(\frac{1}{4}\right) \right) = \frac{1}{2} - \left(\frac{1}{2}\right)^{i+2} \quad (19)$$

ii. Let $A = [a_{31}, a_{30}, \ldots, a_0]_2$ be given in advance and $B = [b_{31}, b_{30}, \ldots, b_0]_2$ be a random 32-bit long register, where b_i $0 \leq i \leq 31$ are independent,

and chosen from the uniform distribution. Denote by $\tilde{p}_i$ the probability that $\tau_i = 0$ in the sum $A + B$ and denote by $\tilde{q}_i = 1 - \tilde{p}_i$ the probability that $\tau_i = 1$ in the sum $A + B$. Then

$$\tilde{p}_i = \left(\sum_{j=0}^{i} (1 - a_{i-j}) \left(\frac{1}{2} \right)^{j+1} \right) + \left(\frac{1}{2} \right)^{i+1}, \tag{20}$$

$$\tilde{q}_i = \sum_{j=0}^{i} a_{i-j} \left(\frac{1}{2} \right)^{j+1}. \tag{21}$$

Proof.

i. We prove the formula for $\tilde{p}_i$ by the mathematical induction on i. The formula for $\tilde{q}_i$ then follows For $i = 0$, $\tilde{p}_0 = \frac{3}{4}$. $\tau_0 = 1$ only if $a_0 = b_0 = 1$, and $\tau_0 = 0$ in the other 3 cases.

Suppose that $k > 0$ and the proposition holds for $i = k - 1$. For $i = k$, $\tau_k = 0$ if $a_k = b_k = 0$ or if $a_k \neq b_k$ and $\tau_{k-1} = 0$. We can write

$$\begin{aligned} \tilde{p}_k &= \frac{1}{4} + \frac{1}{2}\tilde{p}_{k-1} \\ &= \frac{1}{4} + \frac{1}{2}\left(\left(\sum_{j=0}^{k-2} \left(\frac{1}{2} \right)^{j} \left(\frac{1}{4} \right) \right) + \left(\frac{1}{2} \right)^{k-1} \left(\frac{3}{4} \right) \right) \\ &= \left(\sum_{j=0}^{k-1} \left(\frac{1}{2} \right)^{j} \left(\frac{1}{4} \right) \right) + \left(\frac{1}{2} \right)^{k} \left(\frac{3}{4} \right) \end{aligned}$$

The second equality in (18) and (19) are obvious.

ii. For $i = 0$

$$\begin{aligned} \text{for } a_0 = 0, \quad \tilde{p}_0 &= 1, \\ \text{for } a_0 = 1, \quad \tilde{p}_0 &= P(b_0 = 0) = \frac{1}{2}. \end{aligned}$$

Suppose that $k > 0$ and the proposition (ii) holds for $i = k - 1$. Then

$$\begin{aligned} \text{for } a_k = 0, \quad \tilde{p}_k &= P(b_k = 0) + P(b_k = 1 \mid \tau_{i-1} = 0) \\ &= P(b_k = 0) + P(b_k = 1) P(\tau_{i-1} = 0) \\ &= \frac{1}{2} + \frac{1}{2}\tilde{p}_{k-1}, \\ \text{for } a_k = 1, \quad \tilde{p}_k &= P(b_k = 0 \mid \tau_{k-1} = 0) \\ &= \frac{1}{2}\tilde{p}_{k-1}. \end{aligned}$$

We used the fact that the bits $b_i, i = 0, \ldots, 31$ are mutually independent. Then

$$\begin{aligned}
\tilde{p}_k &= \frac{1}{2}(1 - a_k) + \frac{1}{2}\tilde{p}_{k-1} \\
&= \frac{1}{2}(1 - a_k) + \left(\frac{1}{2}\right)^{k+1} + \sum_{j=0}^{k-1}(1 - a_{k-1-j})\left(\frac{1}{2}\right)^{j+2} \\
&= \left(\sum_{j=0}^{k}(1 - a_{k-j})\left(\frac{1}{2}\right)^{j+1}\right) + \left(\frac{1}{2}\right)^{k+1},
\end{aligned}$$

where we used the induction assumption for $\tilde{p}_{k-1}$. The formula for $\tilde{q}_i$ can be proved similarly.

□

It follows from Proposition 5.1.*ii*.:

$$\begin{aligned}
p_{28} &= P(A_{28} | \cap_{i<28} A_i) \\
&= \begin{cases}
\tilde{p}_{24} = \left(\sum_{j=0}^{24}(1 - d_{24-j})\left(\frac{1}{2}\right)^{j+1}\right) + \left(\frac{1}{2}\right)^{24+1}, \\
\qquad \text{for } IV^0_{1,25} = 0, \quad (\tau_{24} = 0), \\
q_{24} = \sum_{j=0}^{24} d_{24-j}\left(\frac{1}{2}\right)^{j+1}, \\
\qquad \text{for } IV^0_{1,25} = 1, \quad (\tau_{24} = 1),
\end{cases}
\end{aligned} \tag{22}$$

where $IV^0_3 = [d_{31}, d_{30}, \ldots, d_0]_2$, and τ_{24} is the carry from the position 24 in the sum $R_{61} + IV^0_3$.

The value of p_{29} can be calculated as follows

$$\begin{aligned}
p_{29} &= P(A_{29} | \cap_{i<29} A_i) \\
&= \begin{cases}
q_{24} = \sum_{j=0}^{24} d_{24-j}\left(\frac{1}{2}\right)^{j+1}, \\
\qquad \text{for } IV^0_{2,25} = 0, \quad (\tau_{24} = 1), \\
p_{24} = \left(\sum_{j=0}^{24}(1 - c_{24-j})\left(\frac{1}{2}\right)^{j+1}\right) + \left(\frac{1}{2}\right)^{24+1}, \\
\qquad \text{for } IV^0_{2,25} = 1, \quad (\tau_{24} = 0),
\end{cases}
\end{aligned} \tag{23}$$

where $IV^0_2 = [c_{31}, c_{30}, \ldots, c_0]_2$, and τ_{24} is the carry from the position 24 in the sum $R_{62} + IV^0_2$. For the original initial vector we get $p_{28} \doteq 0.9017$ and $p_{29} \doteq 0.3650$.

Remark 5.2. Stevens in [8] (Section 5) proposed to add conditions $IV_3^0[25] = IV_3^0[24]$ and $IV_2^0[25] \neq IV_2^0[24]$ on the initial vector of the attack. These conditions will cause that $p_{28} \geq \frac{1}{2}$ and $p_{29} \geq \frac{1}{2}$. If the rest of the bits in the IV_3^0 and IV_2^0 are chosen randomly from the uniform distribution, then the average value of p_{28} and p_{29} should be close to the value 0.75.

5.2 The Calculation of Complexity

In this section we present the calculation of computational complexity of the three presented algorithms independent from such values as CPU performance or programming language in which the algorithms were implemented.

Basic unit in which we measure complexity is computation of one MD5 step from definition 1.1. This computation consists of four additions modulo 2^{32}, one bitwise boolean function at registers of length of 32 bits and one left shift rotation:

$$R_i = R_{i-1} + (f_i(R_{i-1}, R_{i-2}, R_{i-3}) + R_{i-4} + K_i + W_i)^{\lll s_i}.$$

The computation of M_i,

$$M_i = (R_i - R_{i-1})^{\lll (32-s_i)} - (f_i(R_{i-1}, R_{i-2}, R_{i-3}) + R_{i-4} + K_i),$$

has the same complexity as 1 MD5 step. Computation of the whole compression function h_{MD5} has complexity $2^6 MD5$ *steps*.

We will ask three different questions for all the algorithms.

1. What is the average number of collision candidates that need to be verified in order to find one collision?

2. What is the average *price* for creating one candidate for verification, i.e. what is the average number of $MD5$ *steps* that are computed in generating one candidate for collision?

3. What is the average *price* for one verification procedure i.e. what is the expected value of the number of $MD5$ *steps* being computed in the procedure of verification?

The following consideration will give us the answer to the first question. Suppose a random experiment has two possible outcomes, success with probability p and failure with probability $q = 1 - p$. The experiment is repeated until a success happens. The number of experiments needed until the first success occurs is a random variable X with the probability density function

$$f(x) = P(X = x) = q^{(x-1)}p.$$

The expected value and the variance of a random variable X that has a geometric distribution with parameter p are

$$\begin{aligned} E(X) &= \frac{1}{p}, \\ var(X) &= \frac{1-p}{p^2}. \end{aligned} \tag{24}$$

Proofs of these facts can be found in every basic course of the probability theory. An interpretation of the expected value is that we need in average 6 tries to throw a 6 on a die. The same fact will be used in calculation of the number of collision candidates that need to be verified. We need to find out the parameter p i.e. probability that all conditions for the registers R_i in the verification procedure are true. Then

$$\begin{aligned} p = P\left(A_1 \cap \cdots \cap A_{35}\right) &= P\left(A_{35} | A_1 \cap \cdots \cap A_{34}\right) P(A_1 \cap \cdots \cap A_{34}) \\ &= P\left(A_{35} | \cap_{i<35} A_i\right) P\left(A_{34} | \cap_{i<34} A_i\right) P(\cap_{i<34} A_i) \\ &= \prod_{j=1}^{35} P\left(A_j | \cap_{i<j} A_i\right) \end{aligned}$$

For $j = 2, \ldots 35$, we denote $p_j = P\left(A_j | \cap_{i<j} A_i\right)$ and $p_1 = (P(A_1))$. That means

$$p = \prod_{j=1}^{35} p_i.$$

Similarly we will denote p_j for the events B_j, $j = 1, \ldots, 42$ as $P\left(B_j | \cap_{i<j} B_i\right)$ in the second block.

Proposition 5.3. *We denote by C the random variable specifying the number of collision candidates needed to be verified until a colliding block is found. Then for the first colliding block we have*

i. $E(C_s) = 2^{31} \frac{1}{p_{28}} \frac{1}{p_{29}}$, *for Stevens' algorithm,*

ii. $E(C_k) = 2^{27} \frac{1}{p_{28}} \frac{1}{p_{29}}$, *for Klima's algorithm,*

iii. $E(C) = 2^{29} \frac{1}{p_4} \frac{1}{p_{28}} \frac{1}{p_{29}}$, *for our algorithm,*

where the values of p_{28} and p_{29} are given in section 5.1 and they depend on the initial vector of the first block, the values of $p_3, \ldots, p_6$ in the Stevens' algorithm are estimated as $\frac{1}{2}$, the value p_4 of our algorithm is given in subsection 3.1.3, where it is also explained why the probability p_4 is 0.65. For the second colliding block holds

i. $\frac{128}{127} 2^{26}$ *for Stevens' algorithm.*

ii. $\frac{128}{127}\frac{8}{7}2^{29}$ *for our algorithm.*

Proof. The random variable C has geometric distribution. Then from the description of the algorithms for the first block we can write

$$\begin{aligned} p_s &= \prod_{j=3}^{35} p_j \\ p_k &= \prod_{j=7}^{35} p_j \\ p_o &= \prod_{j=1}^{35} p_j. \end{aligned}$$

For the first block we will make assumption that the value of p_j, for $j = 7, \ldots, 27, 30, \ldots, 35$, is $\frac{1}{2}$, because the occurrence of the events A_i, $i < j$ does not have any influence on the occurrence of the event A_j and the probability that condition is satisfied is $\frac{1}{2}$. We have estimated the value of $p_3, \ldots, p_6$ in the Stevens' algorithm at $\frac{1}{2}$. The calculations of p_{28} and p_{29} are given in Section 5.1. Proposition then follows from (24). Propositions for the second block can be calculated in the same way. We note that the constant $\frac{127}{128}$ is from the probability of the event B_{37} given that all events B_i, for $i < 29$ have already occurred and the constant $\frac{8}{7}$ is from the probability of the event B_{12} given that all events B_i, for $i < 12$ have already occurred. □

For the original IV^0, $p_{28}^{-1}p_{29}^{-1} \doteq 3.0384$. That means the average numbers of collision candidates should be close to $(3.0384)2^{31}$ for Stevens' algorithm, $(3.0384)2^{27}$ for Klima's algorithm, $(4.6745)2^{29}$ for our algorithm. Conditions on the IV^0 added by Stevens will in average change $p_{28}^{-1}p_{29}^{-1} \doteq 0.75^{-2} \doteq 1.77$, what should speed up the attacks.

Proposition 5.4. *We denote by G the random variable specifying the price for generating of one collision candidate for the first colliding blocks. Then for generating of the first block holds*

i.

$$E(G_s) = 1 + \frac{k_{s1}}{2^{15}} \doteq 1.03 \ MD5 \ steps$$

where k_{s1} is the average number of $MD5$ steps being computed in steps 1.-3. of Algorithm 3,

ii.

$$E(G_k) = \frac{c_0 + \sum_{i=1}^{6} 2^{\sum_{j<i} s_j}(2^{s_i} - 1)c_i}{2^{s_1+s_2+s_3+s_4+s_5+s_6}} \doteq 3.314 \ MD5 \ steps$$

where c_0 is the price for finding the first collision candidate satisfying the prescribed conditions for $R_i, i \leq 23$, s_i is the strength of the tunnel i and c_i is the price of the tunnel i, for $i = 1, \ldots, 6$.

iii.

$$E(G_o) \doteq 8.02 \ MD5 \ steps$$

for our algorithm.

For generating of the second block holds

i.

$$E(G_s) = 1 + \frac{k_{s2}}{2^{15}} \doteq 2 \ MD5 \ steps,$$

where k_{s2} is the average number of $MD5$ steps being computed in steps 1.-3. of Stevens' algorithm for the second block.

ii.

$$E(G_o) \doteq 4 \ MD5 \ steps$$

for our algorithm.

Proof. The computation of the average price for our Algorithm 1 is given at the end of subsection 3.1.2. The average price for generating the candidate our Algorithm 2 follows from the explanation of the algorithm in subsection 3.2.2.

For Stevens' algorithms the value follow from the construction of the algorithms. The constant k_s1 and k_s1 can be estimated according to the implementation of step 3 in the algorithms. For the first block there are 9 prescribed conditions for $R_{17}, \ldots, R_{20}$ to be satisfied and we can suppose that $k_{s1} < 2^{10}$. For the second block there are 14 prescribed conditions and we can suppose that $k_{s2} < 2^{15}$.

We used the following consideration for the Klima's algorithm with the tunnels. In the beginning we must generate the first collision candidate satisfying all the prescribed conditions until R_{23} for the price c_0. Then we clone the first collision candidate in the first tunnel into the 2^{s_1} collision candidates. That means we need to do $2^{s_1} - 1$ computations of the new collision candidates for the price c_1. Then we clone the 2^{s_1} candidates into the $2^{s_1+s_2}$ candidates in the tunnel 2. That means we need to do $2^{s1}(2^{s_2} - 1)$ computations for the price c_2. Applying this to all tunnels we calculate the total price for creating of $2^{s_1+\cdots+s_6}$ candidates that are satisfying the prescribed conditions for the registers $R_i, i \leq 23$. We get the number 3.314 by substituting the values of s_i and c_i from the Table 28. □

Proposition 5.5. *We denote by V the random variable specifying the expected price for the verification of one collision candidate. Then for the algorithms for the first block holds*

i. $E(V_s) \doteq 2.9999 \ MD5$ *steps for Stevens' algorithm,*

ii. $E(V_k) \doteq 16.9999 \ MD5$ *steps for Klima's algorithm,*

iii. $E(V_o) \doteq 4.5750$ $MD5$ *steps for our algorithm.*

For the algorithms for the second block holds

i. $E(V_s) \doteq 2.9999$ $MD5$ *steps for Stevens' algorithm,*

ii. $E(V_o) = 1.9440$ $MD5$ *steps for our algorithm.*

Proof. We can count the number of $MD5$ *steps* from the beginning of the verification procedure to the point when particular condition is being verified and calculate the probability that the verification procedure will end when the condition j is not satisfied. This probability can be calculated from

$$p_{x_i} = \left(1 - \prod_{j=i}(p_j)\right)\left(\prod_{j<i} p_j\right),$$

where the first product goes over all conditions for step i and the second product goes over all conditions for steps $j < i$. The values are for both blocks are given in Appendix B in Tables 35 and 36. Now we can calculate the expected value of the random variable V given by Table 35 and 36 as

$$E(V) = \sum_i x_i p_{x_i},$$

where i goes over all steps in the particular verification procedure. □

Now the average running time of the algorithms until they find a colliding block for the original IV can be calculated as

$$E(C)\left(E(G) + E(V)\right),$$

where C, P and V are the random variables defined in Propositions 5.3, 5.4, 5.5.

For the first colliding block and the original initial vector the average running time is

i. $(3.0384)2^{31}(1.032 + 2.999) \doteq 24.5010(2^{30})$ $MD5$ *steps* for Stevens' algorithm,

ii. $(3.0384)2^{27}(3.314 + 16.999) \doteq 7.7153(2^{30})$ $MD5$ *steps* for Klima's algorithm,

iii. $(4.6745)2^{29}(8.02{+}4.5750) \doteq 29.437(2^{30})$ $MD5$ *steps* for our algorithm.

For the first colliding block and given initial vector we have

i. $8.0638(p_{28}p_{29})^{-1}2^{30}$ $MD5$ *steps* for Stevens' algorithm,

ii. $2.5392(p_{28}p_{29})^{-1}2^{30}$ $MD5$ *steps* for Klima's algorithm,

iii. $9.6884(p_{28}p_{29})^{-1}2^{30}$ $MD5$ *steps* for our algorithm, where the values p_{28} and p_{29} depends on the given initial vector and formulas for their calculation are given in Section 5.1.

We did not calculate the complexity of the Klima's algorithm with tunnels for the second colliding block, but it can be calculated similarly than for the first block algorithm. For Stevens' and our algorithms we have

i. $\frac{128}{127}2^{26}(2 + 2.999) \doteq 0.3149(2^{30})$ $MD5$ *steps* for Stevens' algorithm,

ii. $\frac{128}{127}\frac{8}{7}2^{29}(4 + 1.9440) \doteq 3.4233(2^{30})$ $MD5$ *steps* for our algorithm.

Now we can estimate that Klima's algorithm for the first block is theoretically about 3.17 times faster than Stevens' algorithm and Stevens' algorithm is theoretically only slightly faster than our algorithm for the first block. The Stevens' algorithm for the second block is about eleven times faster than our algorithm for the second block. If we compare the complexity of our algorithms for the first and the second block, then the complexity of the algorithm for the first block is about 8 times higher than the complexity of the algorithm for the second block. For Stevens' algorithms is the complexity of the algorithm for the first block approximately 93 times higher than the complexity of the algorithms for the second block. It says that the complexity of algorithms for the first block is more important for the complexity of finding two colliding messages.

6 Results

We implemented our proposed algorithm for the first block as described in Section 3. First we measured the running time of 2^{26} full MD5 compression functions which is equivalent to 2^{32} $MD5$ *steps* of our implementation on AMD Athlon XP1800+ and we obtained that 2^{32} $MD5$ *steps* $\doteq 58.12$ sec. From this we can calculate that $29.437(2^{30})$ $MD5$ *steps* $\doteq 427.719$ sec. Then we ran 239 tests of the algorithm for the first block and original IV and we got the average running time 456.83 sec. This test confirmed that our theoretical calculation is quite close. Stevens in Section 6 of [8] gives his results for the average number of $MD5$ *steps* for his first block and recommended IV is $2^{33.6}$. If we suppose that his recommendation gives the average value of probabilities from Section 5.1 $p_{28} = p_{29} = 0.75$, then according to our calculation of complexity in Section 5.2. the average running time should be

$$(0.75)^{-2}(1.032 + 2.999)2^{31} \doteq (7.168)2^{31} \doteq 2^{33.84}.$$

That also confirms our theoretical calculations of the complexity, based not on the measuring.

Our implementation of the algorithm for the second block differs from the proposed algorithm in Section 3, but allows us to find the second block in 171 sec in average.

Remark 6.1. Some other statistics of the Klima's and Stevens' implementation can be found at http://crypto-world.info/info/result.pdf. The statistics confirm our theoretical calculation that Klima's algorithm is approximately three times faster than the Stevens'.

Conclusion

Our presented algorithm using multi-message modification allows us to find collisions in MD5 in only minutes on a common personal computer. We compared computional complexity of our algorithm with algorithms of Klima and Stevens. The new method of tunneling seems to be very efficient compared to the multi-message modification method. We have also shown the impact of the initial vector value to the running time of the algorithms.

References

[1] Ron Rivest *The MD5 Message-Digest Algorithm*, Request for Comments: 1321, April 1992, http://rfc.net/rfc1321.html.

[2] X. Wang, X. Lai, D. Feng, and H. Yu. *Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD*, presented at the rump session of CRYPTO 2004, August 2004, http://eprint.iacr.org/2004/199.

[3] Xiaoyun Wang, Hongbo Yu. *How to break MD5 and other hash functions*, presented at EUROCRYPT 2005, 2004, http://www.infosec.sdu.edu.cn/paper/md5-attack.pdf.

[4] Philip Hawkes, Michael Paddon, Gregory G. Rose *Musings on the Wang et al. MD5 Collision*, Cryptology ePrint Archive, Report 2004/264, http://eprint.iacr.org/2004/264.pdf

[5] Vlastimil Klima *Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications*, Cryptology ePrint Archive, http://eprint.iacr.org/2005/102.pdf

[6] Jun Yajima, Takeshi Shimoyama *Wangs sufficient conditions of MD5 are not sufficient*, Cryptology ePrint Archive: Report 263/2005, http://eprint.iacr.org/2005/263.

[7] Jie Liang, Xuejia Lai *Improved collision attack on hash function MD5*, Cryptology ePrint Archive: Report 425/2005, http://eprint.iacr.org/2005/425.

[8] Marc Stevens *Fast Collision Attack on MD5*, Cryptology ePrint Archive: Report 104/2006, http://eprint.iacr.org/2006/104.

[9] Vlastimil Klima *Tunnels in Hash Functions: MD5 Collisions Within a Minute*, Cryptology ePrint Archive: Report 105/2006, http://eprint.iacr.org/2006/105.

A Prescribed Conditions

i	$R_i[31]$			$R_i[0]$
0	`........`	`........`	`........`	`........`
1	`........`	`........`	`........`	`........`
2	`........`	`....0...`	`....0...`	`.0......`
3	`1...0...`	`0^^^1^^^`	`^^^^1^^^`	`^011....`
4	`1000100.`	`01000000`	`00000000`	`0010.1.1`
5	`0000001^`	`01111111`	`10111100`	`0100^0^1`
6	`00000011`	`11111110`	`11111000`	`00100000`
7	`00000001`	`1..10001`	`0.0.0101`	`01000000`
8	`11111011`	`...10000`	`0.1^1111`	`00111101`
9	`0111....`	`0..11111`	`1.01...0`	`01....00`
10	`0010....`	`....0001`	`1.00...0`	`11....10`
11	`000...^^`	`....1000`	`0001...1`	`0.......`
12	`01....01`	`....1111`	`111....0`	`0...1...`
13	`000...00`	`....1011`	`111....1`	`1...1...`
14	`011...01`	`........`	`10......`	`...0....`
15	`001.....`	`........`	`........`	`........`

Table 31: Prescribed conditions for $R_0, \ldots, R_{15}$, block 1

Notation:

$$R_i[j] = \begin{cases} \texttt{'.'} & \text{if there is no condition on the bit,} \\ \texttt{'0', '1'} & \text{if } R_i[j] \text{must be the value 0 or 1,} \\ \texttt{'^'} & \text{if } R_i[j] \text{must be equal to } R_{i-1}[j], \\ \texttt{'!'} & \text{if } R_i[j] \text{must not be equal to } R_{i-1}[j]. \end{cases}$$

i	$R_i[31]$			$R_i[0]$	Z_i
16	01.....	.....0.	^.....	....^...	
17	0.^.....	.....1.	.....	.....	
18	0.....	.....0.	.....	.....	$Z_{18}[17-3] \neq 1^*$
19	0.....	.....	.....	.....	$Z_{19}[31-29] \neq 0^*$
20	0.....	.....^.	.....	.....	
21	0.....	.....	.....	.....	
22	0.....	.....	.....	.....	$Z_{22}[17] = 0$
23	1.....	.....	.....	.....	
24-44	.....	.....	.....	.....	$Z_{34}[15] = 0$
45	I.....	.....	.....	.....	
46	J.....	.....	.....	.....	
47	I.....	.....	.....	.....	
48	J.....	.....	.....	.....	
49	K.....	.....	.....	.....	
50	J.....	.....	.....	.....	
51	K.....	.....	.....	.....	
52	J.....	.....	.....	.....	
53	K.....	.....	.....	.....	
54	J.....	.....	.....	.....	
55	K.....	.....	.....	.....	
56	J.....	.....	.....	.....	
57	K.....	.....	.....	.....	
58	J.....	.....	.....	.....	
59	I.....0.	.....	.....	.....	
60	J.....1.	.....	.....	.....	
61	I.....0.	.....	.....	.....	
62	J.....0.	.....	.....	.....	
63	.....	.....	.....	.....	
$IV^1_{m,0}$	.....	.....	.....	.....	
$IV^1_{m,3}$	.....0.	.....	.....	.....	
$IV^1_{m,2}$	^.....01.	.....	.....	.....	
$IV^1_{m,1}$	^.....00.	.....	.....	..0.....	

Table 32: Prescribed conditions for $R_{16}, \ldots, R_{63}$ and IV^1_m, block 1
$I, J, K \in \{0, 1\}, I \neq K$

i	$R_i[31]$			$R_i[0]$
-3	......0.	....	....	....
-2	^.....01.	....	....	....
-1	^.....00.	....	....	..0.....
0	!...010.	..1....1	....0...	..0.....
1	^^^^010.	..0^^^^1	0..^1...	^^0..00.
2	^011111.	..011111	1..01..1	011^^111
3	^011101.	..000100	...00^^0	00010001
4	!10010..	..101111	...01110	01010000
5	^..0010.	..10..10	.1.01100	01010110
6	!..1011^	^.00..01	^0.111110	00.....1
7	^..00100	0.11..10	1.....11	111...^0
8	^..11100	0.....01	0..^..01	110...01
9	^....111	1....011	1..0..11	11....00
10	^.......	....^101	1^^0..11	11....11
11	^^^^^^^^	1000	0001....	1.......
12	!0111111	0...1111	111.....	0...1...
13	^1000000	1...1011	111.....	1...1...
14	01111101	0	00.....	....0...
15	0.10....	1	....	....

Table 33: Prescribed Conditions for $R_{-3}, \ldots, R_{15}$, block 2

i	$R_i[31]$			$R_i[0]$	Z_i
16	0.....	.....0.	^.....	....^...	$Z_{16}[24-26] \neq 1^*$
17	0.^.....	.....1.	.....	.....	
18	0.....	.....0.	.....	.....	$Z_{18}[17-3] \neq 1^*$
19	0.....	.....	.....	.....	$Z_{19}[31-29] \neq 0^*$
20	0.....	.....^.	.....	.....	
21	0.....	.....	.....	.....	
22	0.....	.....	.....	.....	$Z_{22}[17] = 0$
23	1.....	.....	.....	.....	
24-44	.....	.....	.....	.....	$Z_{34}[15] = 0$
45	I.....	.....	.....	.....	
46	J.....	.....	.....	.....	
47	I.....	.....	.....	.....	
48	J.....	.....	.....	.....	
49	K.....	.....	.....	.....	
50	J.....	.....	.....	.....	
51	K.....	.....	.....	.....	
52	J.....	.....	.....	.....	
53	K.....	.....	.....	.....	
54	J.....	.....	.....	.....	
55	K.....	.....	.....	.....	
56	J.....	.....	.....	.....	
57	K.....	.....	.....	.....	
58	J.....	.....	.....	.....	
59	I.....0.	.....	.....	.....	
60	J.....1.	.....	.....	.....	
61	I.....1.	.....	.....	.....	$Z_{61}[21-15] \neq 1^*$
62	J.....1.	.....	.....	.....	
63	.....1.	.....	.....	.....	

Table 34: Prescribed conditions for $R_{16}, \ldots, R_{63}$ block 2
$I, J, K \in \{0, 1\}, I \neq K$

B Procedure of Verification

			Our Alg.		Stevens' Alg.		Klima's Alg.	
i	n_i	p_j	x_i	p_{x_i}	x_i	p_{x_i}	x_i	p_{x_i}
21	1	2^{-1*}			1	(2^{-1})		
22	2	$p_4^*,\ 2^{-1}$	1	0.675	2	$3(2^{-3})$		
23	1	2^{-1}	3	$0.65(2^{-2})$	3	2^{-4}		
24 - 33	0	0	$4-13$	0	$4-17$	0	$1-9$	0
34	1	2^{-1}	14	$0.65(2^{-3})$	18	2^{-5}	10	2^{-1}
35 - 46	0	0	$15-26$	0	$19-30$	0	$11-22$	0
47	1	2^{-1}	27	$0.65(2^{-4})$	31	2^{-6}	23	2^{-2}
48	1	2^{-1}	28	$0.65(2^{-5})$	32	2^{-7}	24	2^{-3}
49	1	2^{-1}	29	$0.65(2^{-6})$	33	2^{-8}	25	2^{-4}
50	1	2^{-1}	30	$0.65(2^{-7})$	34	2^{-9}	26	2^{-5}
51	1	2^{-1}	31	$0.65(2^{-8})$	35	2^{-10}	27	2^{-6}
52	1	2^{-1}	32	$0.65(2^{-9})$	36	2^{-11}	28	2^{-7}
53	1	2^{-1}	33	$0.65(2^{-10})$	37	2^{-12}	29	2^{-8}
54	1	2^{-1}	34	$0.65(2^{-11})$	38	2^{-13}	30	2^{-9}
55	1	2^{-1}	35	$0.65(2^{-12})$	39	2^{-14}	31	2^{-10}
56	1	2^{-1}	36	$0.65(2^{-13})$	40	2^{-15}	32	2^{-11}
57	1	2^{-1}	37	$0.65(2^{-14})$	41	2^{-16}	33	2^{-12}
58	1	2^{-1}	38	$0.65(2^{-15})$	42	2^{-17}	34	2^{-13}
59	2	$2^{-1}, 2^{-1}$	39	$1.95(2^{-17})$	43	$3(2^{-19})$	35	$3(2^{-15})$
60	2	$2^{-1}, 2^{-1}$	40	$1.95(2^{-19})$	44	$3(2^{-21})$	36	$3(2^{-17})$
61	2	$2^{-1}, 2^{-1}$	41	$1.95(2^{-21})$	45	$3(2^{-23})$	37	$3(2^{-19})$
62	2	$2^{-1}, 2^{-1}$	42	$1.95(2^{-23})$	46	$3(2^{-25})$	38	$3(2^{-21})$
63	0	0	43	0	47	0	39	0
IV^1	8		44	$0.65(2^{-23})$	48	2^{-25}	40	2^{-21}
Total				32		33		29

Table 35: The verification procedure block 1

*$p_4 = P(Z_{22}[17] = 0) = 0.65$ for our Algorithm 1. For Stevens' Algorithm is estimated as 0.5.

			Our Alg.		Stevens' Alg.		Klima's Alg.	
i	n_i	p_j	x_i	p_{x_i}	x_i	p_{x_i}	x_i	p_{x_i}
19	1+1*	$2^{-1*}, 2^{-1*}$	1	$9(2^{-4})$				
20	2	$2^{-1*}, 2^{-1*}$	2	$21(2^{-6})$				
21	1	2^{-1*}	4	$7(2^{-7})$	1	(2^{-1})		
22	2	2^{-1*}, 2^{-1}	6	$21(2^{-9})$	2	$3(2^{-3})$		
23	1	2^{-1}	7	$7(2^{-10})$	3	2^{-4}		
24 - 33	0	0	$8-23$	0	$4-17$	0	$1-9$	0
34	1	2^{-1}	24	$7(2^{-11})$	18	2^{-5}	10	2^{-1}
35 - 46	0	0	$25-36$	0	$19-30$	0	$11-22$	0
47	1	2^{-1}	37	$7(2^{-12})$	31	2^{-6}	23	2^{-2}
48	1	2^{-1}	38	$7(2^{-13})$	32	2^{-7}	24	2^{-3}
49	1	2^{-1}	39	$7(2^{-14})$	33	2^{-8}	25	2^{-4}
50	1	2^{-1}	40	$7(2^{-15})$	34	2^{-9}	26	2^{-5}
51	1	2^{-1}	41	$7(2^{-16})$	35	2^{-10}	27	2^{-6}
52	1	2^{-1}	42	$7(2^{-17})$	36	2^{-11}	28	2^{-7}
53	1	2^{-1}	43	$7(2^{-18})$	37	2^{-12}	29	2^{-8}
54	1	2^{-1}	44	$7(2^{-19})$	38	2^{-13}	30	2^{-9}
55	1	2^{-1}	45	$7(2^{-20})$	39	2^{-14}	31	2^{-10}
56	1	2^{-1}	46	$7(2^{-21})$	40	2^{-15}	32	2^{-11}
57	1	2^{-1}	47	$7(2^{-22})$	41	2^{-16}	33	2^{-12}
58	1	2^{-1}	48	$7(2^{-23})$	42	2^{-17}	34	2^{-13}
59	2	$2^{-1}, 2^{-1}$	49	$21(2^{-25})$	43	$3(2^{-19})$	35	$3(2^{-15})$
60	2	$2^{-1}, 2^{-1}$	50	$21(2^{-27})$	44	$3(2^{-21})$	36	$3(2^{-17})$
61	2+1*	$2^{-1}, 2^{-1}$	51	$\frac{25}{32}\frac{7}{2^{27}}$	45	$25(2^{-27})$	37	$25(2^{-23})$
62	2	$2^{-1}, 2^{-1}$	52	$\frac{7}{32}\frac{3}{4}\frac{7}{2^{27}}$	46	$21(2^{-29})$	38	$21(2^{-25})$
63	1	0	53	$\frac{7}{32}\frac{1}{4}\frac{7}{2^{27}}$	47	$7(2^{-29})$	39	$7(2^{-25})$
Total				$29+2^*$		$26+1^*$		$22+1^*$

Table 36: The verification procedure block 2