The Study of Inversion Problems of Cryptographic Hash Functions From MD Family Using Algorithms for Solving Boolean Satisfiability Problem

Irina Gribanova, Oleg Zaikin, Stepan Kochemazov, Ilya Otpuschennikov, and Alexander Semenov

Matrosov Institute for System Dynamics and Control Theory SB RAS, Irkutsk, Russia
\{the42dimension,zaikin.icc,veinamond\}@gmail.com,\{otilya,biclop.rambler\}@yandex.ru

Abstract. In this paper we present the results of application of state-of-the-art SAT solvers to inversion of cryptographic hash functions from the MD family. In particular we consider the problems of finding preimages and collisions for MD4 and MD5. To solve them we use the approach based on reducing the original problems to Boolean satisfiability problem (SAT). The propositional encoding of the algorithms specifying the considered functions was performed using the Transalg software system. The features of this system make it possible to effectively augment the SAT encodings for MD4 and MD5 hash functions with various additional constraints that improve the effectiveness of SAT solvers on corresponding instances. The effectiveness of the proposed algorithms is better than that in a number of preceding papers. We used the developed algorithms to find new families of two-block collisions for MD5 and to construct new differential paths for finding single-block collisions for MD4.

Keywords: SAT, MD4, MD5, cryptanalysis, preimage attack, collision, inversion problem, hash functions, Transalg.

1 Introduction

Let us denote by $\{0,1\}^k, k \in N$ the set of all binary words of length k. By $\{0,1\}^*$ we denote the set of all binary words of an arbitrary finite length (i.e. $\{0,1\}^* = \bigcup\limits_{k=0}^{\infty} \{0,1\}^k$, where $\{0,1\}^0 = \emptyset$).

Let us remind [10] that a hash function is a total computable discrete function of the kind $\chi : \{0,1\}^* \to \{0,1\}^C$, where C is some constant representing a length of hash value. Hash functions are used in various areas of computer science. In particular, they are applied to speed up access to large data sets. In cryptography, the scope of hash functions applications is also quite wide and has both theoretical (to construct proofs in random oracle model [4]), and practical

aspects. For example, in all state-of-the-art algorithms for constructing digital signatures, the message to be signed is first hashed. Cryptographic hash functions have to meet additional requirements, which consist in the fact that the inversion problems of such functions should be computationally hard. In particular, the problem of finding a preimage for a given hash value and the problem of finding collisions should both be hard. Let us remind that if there exists a pair of messages $x_1, x_2 \in \{0,1\}^*$, $x_1 \neq x_2$ such, that $\chi(x_1) = \chi(x_2)$, then these messages form a collision for hash function χ.

In the present paper we describe the results of applying algorithms for solving Boolean satisfiability problem (SAT) to finding preimages and collisions of some cryptographic hash functions. These results are a part of actively developing direction of research known as SAT-based cryptanalysis. In particular, within the context of mentioned problems we present the results of SAT-based cryptanalysis of MD4 and MD5 cryptographic hash functions. Hereinafter by inversion problems we mean both the problem of finding preimages and the problem of finding collisions for considered hash functions.

Let us give a brief outline of the paper. In the next section we present the theoretical foundations of SAT-based cryptanalysis. In the third section we consider finding preimages and collisions for MD4 and MD5 as special cases of SAT, paying special attention to characteristic features of obtained SAT instances. In the fourth section we describe new methods and results obtained using them. The fifth section is a short review of the related works preceding our research.

2 Theoretical Foundations of SAT-based Cryptanalysis

Boolean Satisfiability Problem (SAT) consists in the following: for an arbitrary Boolean formula F to decide whether it is satisfiable or not, i.e. if there exists a set of values of Boolean variables from this formula that makes it TRUE. In general case this problem can be effectively (in polynomial time on the size of code of F) reduced to SAT for formula C_F in the Conjunctive Normal Form (CNF). That is why nowadays for convenience SAT is usually considered for some CNF. Also SAT is often understood as a search variant of the problem: given a CNF C to decide if it is satisfiable and if the answer is 'Yes' to provide a corresponding satisfying assignment.

In the last 15 years there was achieved a dramatic progress in the development of SAT-solving algorithms and techniques [6]. Today these algorithms are successfully applied in symbolic verification, computational combinatorics, bioinformatics and many other areas. In the recent years there can be seen a growing interest to applying SAT solvers to cryptanalysis problems. The corresponding direction is now known as SAT-based cryptanalysis and the number of works related to it is steadily growing.

SAT-based cryptanalysis relies on the ability to effectively reduce inversion problems of discrete functions to SAT. The effectiveness of these algorithms, called propositional encoding procedures, follows from the Cook theorem [8]. Let us consider some discrete function $f : \{0,1\}^* \to \{0,1\}^*$ specified by a polynomial

algorithm A. For each $n \in N$ this algorithm specifies a function of the kind: $f_n : \{0,1\}^n \to \{0,1\}^*$. Hereinafter by inversion problem for discrete function f_n we mean the following problem: given $y \in Range f_n$ and known algorithm A to find such $x \in \{0,1\}^n$ that $f_n(x) = y$. In the context of this general problem it is possible to consider many cryptanalysis problems. For example, suppose that A is an algorithm which specifies hash function $\chi : \{0,1\}^* \to \{0,1\}^C$. Then for a specific n program A specifies a function $\chi_n : \{0,1\}^n \to \{0,1\}^C$ and for a given hash image y of some message x we can consider the problem of finding a preimage x', $\chi_n(x') = y$, where x' does not have to be equal to x.

The advantage of the presented above style of formulating cryptanalysis problems lies in the fact that we can effectively reduce any such problem to SAT for some satisfiable CNF $C(f_n, y)$. In recent years there appeared several different automated software systems designed to perform the corresponding reductions. From our point of view the most interesting of them are URSA [21] and Cryptol [18, 19]. In our computational experiments we used the Transalg system [29, 30] developed by us. Let us briefly describe its functional capabilities.

The Transalg system was specifically developed to automate reducing the inversion problems of discrete functions, specified by algorithmic descriptions, to SAT. It is based on Cook's ideas on propositional encoding of algorithms [8] and King's ideas on symbolic execution [24]. The Transalg system uses the domain specific TA-language with C-like syntax [23] to specify considered discrete functions. In the process of translating TA-programs to SAT it uses standard techniques from the compilation theory. The result of the translation is not an executable code but a set of Boolean formulas that naturally corresponds to a system of Boolean equations. Using Tseitin transformations [37] we can construct a SAT encoding from this set.

Thus, the initial stage of constructing a SAT instance encoding an inversion problem of some discrete function consists in describing this function on a specialized TA-language. Since TA-language is a procedural C-like language, it is usually enough to make small changes to the existing implementation of an algorithm written in the C language to obtain the corresponding TA-program. The TA-language supports basic constructions typical for procedural languages (variable, array and function declarations, assignment operators, conditional operators, loops, function calls), various bit and integer operations including bit shifting and numerical comparison.

Hereinafter, by translation we mean the process of constructing the propositional encoding of a discrete function for a given TA-program. The translation of any TA-program consists of two main stages. At the first stage Transalg parses the source code of a TA-program and constructs a syntax tree using standard techniques of the compilation theory [1]. At the second stage the system employs the concept of symbolic execution [24] to construct a propositional encoding of a considered TA-program. The propositional encoding can be provided in several standard formats: CNF, DNF (Disjunctive Normal Form), ANF (Algebraic Normal Form). Transalg has an option of applying miminization to

constructed Boolean formulas. For this purpose it uses the ESPRESSO Boolean minimization library which was embedded into system as one of its modules.

Let us briefly discuss the SAT solving algorithms used in SAT-based cryptanalysis. According to the Cook theorem, SAT is NP-hard problem. That is why it is unlikely that there are polynomial deterministic algorithms for its solving. Nevertheless, 'effective' (that make it possible to solve practical instances in reasonable time) algorithms for solving SAT are in high demand in a number of important areas of science, first of all in formal verification. In recent years this demand led to a 'boom' in the development of strategies and heuristics for solving SAT.

There are a number of general concepts underlying state-of-the-art SAT solvers. The CDCL (Conflict Driven Clause Learning) concept first described in [25] became one of the most fruitful. Since CDCL-solvers show the highest effectiveness in application to inversion of cryptographic functions, let us briefly describe their design features. The CDCL-solver is based on DPLL algorithm (Davis-Putnam-Logemann-Loveland) [13], complemented by Clause Learning technique, which allows to store the information about traversed fragments of the search space in the form of conflict clauses. A conflict clause is a logical implication of an original CNF C, so the conjunction of this clause with C gives CNF C', that is satisfiable exactly on the same assignments as C (or both are unsatisfiable). Conflict clauses are used to derive new information and therefore direct the search to a new path. In the worst case scenario both DPLL algorithm and DPLL + CDCL algorithms are exponential. It follows from the fact that the resolution method is exponential [20] and from the results of [3]. Nevertheless, state-of-the-art CDCL-solvers are surprisingly successful when dealing with 'industrial' SAT problems of large dimensions (tens of thousands of variables, hundreds of thousands of clauses). As it was already mentioned, it is the CDCL-solvers that perform best on cryptographic tests, including the inversion problems for hash functions.

3 Inversion Problems of MD4 and MD5 as SAT

In the present paper we apply SAT-based cryptanalysis to the problems of finding collisions and to inversion (i.e. finding preimages) of cryptographic hash functions MD4 [32] and MD5 [31]. Let us briefly discuss design features of these functions.

There is a number of algorithmic constructions used for building cryptographic hash functions. One of the most commonly used is the Merkle-Damgard construction [27, 12], underlying such well-known families of cryptographic functions as MD and SHA. This construction implies that an original message is split into blocks of fixed size (if necessary the length of a message can be extended using the so-called 'padding'). During the processing of the initial message each block is treated one at a time using special function called *compression function*. It converts each block into a short message, stored in specially allocated memory registers. The values of the registers are considered as the values of the so-called 'chaining variables'. After processing all message blocks the concate-

nation of current values of chaining variables gives us the hash value. At the initial moment the chaining variables are assigned with initial values, which are commonly known and usually are fixed in the algorithm specification. The general scheme of Merkle-Damgard construction is shown in Figure 1 (the message blocks that form the original message are denoted by $M_1, \ldots, M_N$, the initial values of chaining variables are denoted by IV, and H_N is a final hash value).

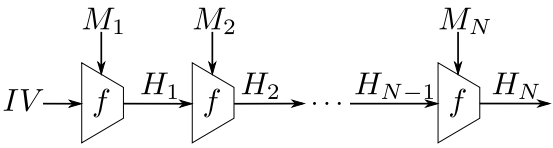

Fig. 1. The Merkle-Damgard construction.

Let us illustrate the work of the Merkle-Damgard construction using the MD5 algorithm as an example. The MD5 algorithm begins with padding an original message. In this process we add to it a special bit sequence $p_1, \ldots, p_q$, $q \geq 65$. The bit p_1 is always equal to 1, while bits $p_{q-63}, \ldots, p_q$ are the binary representation of 64-bit number equal to the number of bits in the original message. Bits $(p_2, \ldots, p_{q-64})$ if they are present, are equal to 0. The parameter $q \geq 65$ is chosen in such a way that the length of the padded message is a multiple of 512 bits. After padding the message is divided into 512-bit blocks $M = (M_1, \ldots, M_N)$. According to the Merkle-Damgard construction an iterative process of calculating the hash value can be described by the equation:

$$H_i = f(H_{i-1}, M_i), 1 \leq i \leq N,$$

where $H_0 = IV$ is the initial value and H_N is the value of hash function. For each $i \in \{1, \ldots, N\}$ the value H_i is the concatenation of 32-bit values of chaining variables. At the initial moment these variables have the following values (according to the specification of the MD5 algorithm): a=`0x67452301`, b=`0xefcdab89`, c=`0x98badcfe`, d=`0x10325476`. Concatenation of these values, i.e. the word $a|b|c|d$, forms IV.

Schematically the compression function f_{MD5} used in the MD5 algorithm is presented in Figure 2. Let us comment the Figure 2. The function f_{MD5} receives as input an array with the current values of chaining variables – a_0, b_0, c_0, d_0. The corresponding values are given in form of 32-bit words. Also compression function f_{MD5} operates with 512-bit input message block M_i, divided into 16 32-bit words: $M_i = (m_1, \ldots, m_{16})$. The process of computing f_{MD5} can be divided into 4 stages, called rounds. In each round an iterative recalculation of chaining variables is performed, the value of each variable is updated 4 times (in Figure 2, each round is represented by transformation Φ^j, $j \in \{1, 2, 3, 4\}$). The transformations corresponding to the first update of chaining variables values in

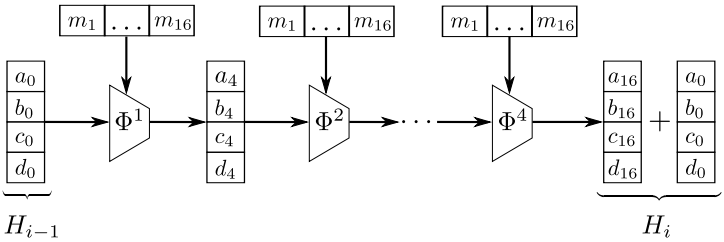

Fig. 2. MD5 compression function.

the first round are specified by the following equations:

$$\begin{aligned} a_1 &= b_0 + ((a_0 + \phi^1(b_0, c_0, d_0) + m_1 + t_1) \ll s_a^1), \\ d_1 &= a_0 + ((d_0 + \phi^1(a_0, b_0, c_0) + m_2 + t_2) \ll s_d^1), \\ c_1 &= d_0 + ((c_0 + \phi^1(d_0, a_0, b_0) + m_3 + t_3) \ll s_c^1), \\ b_1 &= c_0 + ((b_0 + \phi^1(c_0, d_0, a_0) + m_4 + t_4) \ll s_b^1). \end{aligned} \tag{1}$$

Here, '+' means addition modulo 2^{32}, '$\ll s$' means s-fold circular left shift of a 32-bit word, s_ζ^1, $\zeta \in \{a, b, c, d\}$ are known constants defined in the algorithm specification (for example, $s_a^1 = 7$, $s_d^1 = 12$, $s_c^1 = 17$, $s_b^1 = 22$), constants t_k, $k \in \{1, 2, 3, 4\}$ are also known. The result of second recalculation of chaining variables values is the set of values a_2, b_2, c_2, d_2 computed using formulas that are different from (1) in that their right parts contain a_1, b_1, c_1, d_1 instead of a_0, b_0, c_0, d_0 and m_5, m_6, m_7, m_8 instead of m_1, m_2, m_3, m_4. Also the constants t_k, $k \in \{5, 6, 7, 8\}$ are used. The round ends after four recalculations, i.e. after all the words $m_1, \ldots, m_{16}$ have been used. The equations used to recalculate chaining variables values at further rounds are generally similar to (1) with the difference that they use other constants and functions ϕ^j. The functions ϕ^j, $j \in \{1, 2, 3, 4\}$ are functions of the kind:

$$\phi^j : \{0,1\}^{32} \times \{0,1\}^{32} \times \{0,1\}^{32} \to \{0,1\}^{32}.$$

They are called round functions and are specified as follows:

$$\begin{aligned} \phi^1(X, Y, Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ \phi^2(X, Y, Z) &= (X \wedge Z) \vee (Y \wedge \neg Z) \\ \phi^3(X, Y, Z) &= X \oplus Y \oplus Z \\ \phi^4(X, Y, Z) &= Y \oplus (X \wedge \neg Z) \end{aligned}$$

In these formulas it is assumed that Boolean operations are performed componentwise over 32-bit words.

The sequence of data transformations listed above forms the algorithm that defines the hash function $\chi_{MD5} : \{0,1\}^* \to \{0,1\}^{128}$. When we fix the length of an original message we obtain functions, the corresponding inversion problems of which can be reduced to SAT using the techniques described above. In what follows we use the Transalg system to perform this reduction. Below we will

briefly consider the important stages of the process of constructing propositional encoding of the algorithm specifying χ_{MD5}.

As it follows from the above, the value χ_{MD5} is formed as a result of an iterative recalculation of chaining variables values. The operations used in this process are integer addition modulo 2^{32}, circular bit shift operation and bitwise conjunction, disjunction and addition modulo 2. Let $a = (a_n, \ldots, a_1)$, $b = (b_n, \ldots, b_1)$ be n-bit numbers (let us use the notation in which the most significant bit is the leftmost bit). Then the integer addition $a + b$ is encoded by the following set of Boolean formulas:

$$
\begin{aligned}
&c_1 \equiv a_1 \oplus b_1\\
&p_1 \equiv a_1 \wedge b_1\\
&c_i \equiv a_i \oplus b_i \oplus p_{i-1}, i = 2, \ldots, n\\
&p_i \equiv a_i \wedge b_i \ \vee a_i \wedge p_{i-1} \vee b_i \wedge p_{i-1}, i = 2, \ldots, n\\
&c_{n+1} \equiv p_n
\end{aligned}
$$

Here, p_i, $i = 1, \ldots, n$ are carry bits, $c = a + b = (c_{n+1}, c_n, \ldots, c_1)$ represents the result of addition. In case of integer addition modulo 2^{32} numbers a, b are represented by 32-bit vectors ($n = 32$), and as a result of the operation we take 32 less significant bits of vector c.

As it was shown in [29], when encoding the circular shift operation it is not necessary to create new Boolean variables. This is a consequence of the fact that the circular shift can be considered as a process of redefinition of relations between variables, already used at the previous stages of construction of propositional encoding.

Operations that appear in round functions are bitwise. Thus, for example, propositional code for the function $\phi^1(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$ is based on the set of formulas of the kind:

$$u_i \leftrightarrow (x_i \wedge y_i) \vee (\neg x_i \wedge z_i), i = 1, \ldots, 32,$$

where u_i is a Boolean variable encoding i-th component in the vector $\phi^1(X, Y, Z)$, $X = (x_1, \ldots, x_{32})$, $Y = (y_1, \ldots, y_{32})$, $Z = (z_1, \ldots, z_{32})$.

All of these procedures can be described using the TA-language. The result of the translation of the constructed TA-program is the CNF $C(\chi_{MD5}, m)$ that encodes the calculation of χ_{MD5} for an input of a fixed length m. To obtain the problem of inversion for this function in the form of SAT instance one has to add to this CNF the specific value of the hash, for which we want to find a preimage, in the form of unit clauses. To construct the SAT encoding for finding a collision (in general case, the k-block collision) two similar TA-programs are used, resulting in the construction of two CNFs $C_1(\chi_{MD5}, m)$ and $C_2(\chi_{MD5}, m)$ over disjoint sets of variables. Then we construct the CNF $C_1(\chi_{MD5}, m) \wedge C_2(\chi_{MD5}, m)$ to which we append the conditions specifying that functions outputs are equal, while functions inputs are not. From the properties of translation procedures used in Transalg, it follows that the resulting CNF, hereinafter denoted by $\tilde{C}$ has as many satisfying assignments as there are collisions of length m. We will say that CNF $\tilde{C}$ encodes the problem of finding collisions of the corresponding hash function (MD5 or MD4).

4 Solving Inverison Problems for Hash Functions from MD family via SAT

Apparently, the first example of applying SAT solvers to cryptanalysis of hash functions can be found in paper [22]. In this work, there were considered the problems of finding collisions of hash functions from MD family in form of SAT. However, in [22] the collisions for the unweakened algorithms have not been found. From this perspective the first successful experience was presented in [28]. In this paper there was implemented the SAT-version of the famous differential attack by X. Wang. Using this attack, first introduced in [38, 39], one can effectively construct single-block collisions for MD4 and two-block collisions for MD5. Below let us briefly consider the main stages of the Wang scheme.

Thus, consider two messages $M = (M_1, \ldots, M_N)$, and $M' = (M'_1, \ldots, M'_N)$, $M \neq M'$. Let

$$H_i = a|b|c|d, H'_i = a'|b'|c'|d'$$

be the values of chaining variables after calculating hash values for message blocks number $i \in \{0, 1, \ldots, N\}$. Consider the following relation:

$$\Delta H_i = a - a'|b - b'|c - c'|d - d'$$

in which we use integer differences modulo 2^{32} between the corresponding values of chaining variables. Given this notation a differential path for a considered hash function can be defined as follows:

$$\Delta H_0 \xrightarrow{(M_1, M'_1)} \Delta H_1 \xrightarrow{(M_2, M'_2)} \ldots \xrightarrow{(M_{N-1}, M'_{N-1})} \Delta H_{N-1} \xrightarrow{(M_N, M'_N)} \Delta H_N = \Delta H$$

It's obvious that $\Delta H_0 = 0$. If $\Delta H = 0$, then the corresponding messages M and M' form an N-block collision. If for some reasons there were selected a particular type of differences ΔH_i, $1 \leq i \leq N$ and of differences between the intermediate values of chaining variables during the calculation of H and H', then the problem of finding collisions transforms into the problem of selecting a pair (M, M') that satisfies the resulting differential path. This is the main idea of differential attacks on MD family presented in [38, 39]. More specifically, in [38, 39] for the MD family hash functions there have been proposed differential paths and described the collision finding method which consists of two simple steps: random message selection and deterministic modification of messages to fit a particular differential path. This method has allowed to construct single-block collisions for the MD4 algorithm and two-block collision for the MD5 algorithm in a reasonable time. The attacks of the described type were often applied in later papers. Among the latest achievements in this direction there should be noted the paper [36], in which single-block collisions for the MD5 hash function were built using differential attacks.

As we noted above, in [28] there was implemented a 'SAT-version' of Wang attack. More precisely, the Boolean constraints encoding the differential paths from [38, 39] were added to the propositional encoding for finding collisions for the MD4 and MD5 hash functions. To find one MD4 collision it took them

about 10 minutes (500 seconds on average) using MINISAT solver [16]. Finding two-block collisions for MD5 proved to be much more difficult.

In our opinion the main disadvantage of [28] is the use of highly specialized tools to construct propositional encodings, because that makes it impossible to reproduce their experiments. In our experiments we used the TRANSALG system. The encodings obtained were more compact compared to encodings presented in [28]. Using CRYPTOMINISAT solver [35] one collision for MD4 was found in a fraction of a second and to find 10,000 collisions it took less than 2 hours. To find two-block MD5 collisions using the encodings constructed by TRANSALG we employed PLINGELING and TREENGELING solvers [5], the winners of the latest SAT competitions. Using TREENGELING solver we managed to isolate one special class of two-block collisions for MD5 with the first 10 zero bytes. To find one collision on one node of 'Academician Matrosov' computing cluster of Irkutsk Supercomputer Center SB RAS (two 16-core Opteron 6276 processors + 64 Gb RAM)[1] it took from a few hours to a couple of days. There were found dozens of collisions of this type. In Figure 3 we present one of them.

M_1	00 00 00 00 00 00 00 00 00 00 20 74 67 a6 f5 48 cb c1 6d a5 3e f7 b8 bc 67 a3 8d d9 3c 9b f5 b8 55 ed 32 06 06 0a 74 a3 0f b6 84 87 47 cf 91 d0 db 4c 6f 43 ef 64 f0 8d a4 1d 50 c6 26 df 95 fe ff d1 2e c9 a0 90 aa b3 7d e7 e5 bc f2 3a 4e ab 24 b8 d4 13 4c cc 7b 1b 00 29 eb f5 53 7a 0d d1 5d 1f b7 79 af 36 ce 08 1e 44 a2 d0 51 ec 91 fb c5 4c a2 89 75 b3 a3 84 ac 97 7f f2 7e 50 d4 56
M_2	00 00 00 00 00 00 00 00 00 00 20 74 67 a6 f5 48 cb c1 6d 25 3e f7 b8 bc 67 a3 8d d9 3c 9b f5 b8 55 ed 32 06 06 0a 74 a3 0f b6 84 87 47 4f 92 d0 db 4c 6f 43 ef 64 f0 8d a4 1d 50 46 26 df 95 fe ff d1 2e c9 a0 90 aa b3 7d e7 e5 bc f2 3a 4e ab 24 b8 d4 93 4c cc 7b 1b 00 29 eb f5 53 7a 0d d1 5d 1f b7 79 af 36 ce 08 1e 44 a2 d0 51 6c 91 fb c5 4c a2 89 75 b3 a3 84 ac 97 7f 72 7e 50 d4 56
$Hash$	c22664780a9766ceb57065eba36af06b

Fig. 3. An example of two-block MD5 collision with first 10 zero bytes.

Next we applied SAT approach to generate new differential paths (different from that proposed by X.Wang et. al.) for finding collisions for the MD4 hash function. Thus, consider the problem of computing MD4 hash values for two different messages M and M'. As we mentioned above, by H and H' we denote hash registers, where H is a hash image of X, and H' is a hash image of X'. We assume that hash images for X and X' are calculated simultaneously. The data recorded in H and H' in a particular time moment can be considered as an

[1] http://www.hpc.icc.ru/

instantaneous configuration of memory registers of the computing device that implements two independent copies of the MD4 algorithm. In total in the given algorithm there are 48 such instantaneous configurations corresponding to basic steps – each round consists of 16 steps (in the case of MD5 there are 4 rounds of 16 steps, making it a total of 64 steps). Let us denote these configurations by h_k and h'_k for registers H and H', respectively, $k \in \{1, \ldots, 48\}$ ($k = 0$ corresponds to an initial configuration). Thus, on the k-th step the configurations h_k and h'_k contain four 32-bit numbers each, corresponding to the values of chaining variables from the considered step. During the transition from the configuration h_k to the configuration h_{k+1} only one of the four chaining variables values is recalculated (the same holds for the transition from h'_k to h'_{k+1}). Let us consider the difference between 32-bit integers modulo 2^{32} only for the values of chaining variables that were recalculated during the transition $k \to k+1$ and denote it as δ_{k+1}.

Let $\tilde{C}$ be a CNF encoding the problem of finding collisions for the MD4 hash function. By C_Δ we denote a CNF, which takes the value 1 (true) if and only if hashed messages M and M' satisfy a fixed differential path (for example, the one from [38]). Let $C_{\delta_k=c}$ be a CNF which takes the value 1 (true) if and only if the value of the difference δ_k is equal to c (here c is some 32-bit number). Since in each particular case c is a 32-bit constant it is not difficult to go through all possible values of c and consider the SAT for corresponding CNFs of the kind:

$$\tilde{C} \wedge C_\Delta \wedge C_{\delta_k=c}.$$

In our experiments we found that for the vast majority of possible values state-of-the-art SAT solvers can prove unsatisfiability of CNF $\tilde{C} \wedge C_\Delta \wedge C_{\delta_k=c}$ very fast (in a fraction of second). The solving of SAT for CNFs of the kind $\tilde{C} \wedge C_\Delta \wedge C_{\delta_k=c}$ for which we could not obtain the answer in a relatively short period of time (a few seconds), were interrupted. Assume that c' is the value of δ_k for which the solving of SAT instance $\tilde{C} \wedge C_\Delta \wedge C_{\delta_k=c'}$ was interrupted and that this c' is not equal to corresponding value of δ_k in the path from [38]. Then this value can be considered as a candidate for the value of δ_k in a new differential path. To solve the corresponding SAT instance it is possible to spend more effort using embarrassing parallelism if necessary. If $\tilde{C} \wedge C_\Delta \wedge C_{\delta_k=c'}$ turns out to be satisfiable, then it means that we constructed a new differential path that is different from the Wang path in the value of δ_k.

This algorithm has been implemented in the form of parallel MPI-program, and launched on the computing cluster 'Academician Matrosov' of Irkutsk Supercomputer Center SB RAS. As a result, several differential paths for MD4, that are different from the Wang path, have been found. Below in Figure 4 we present one of the found paths, which is different from the Wang path in values δ_k for $k \in \{13, 17, 20, 21\}$. Using this differential path and message difference from [38] we constructed CNF $\tilde{C}'$ and used CRYPTOMINISAT solver [35] to evaluate its effectiveness. It took the solver 416 seconds to find 1000 collisions. When applied to the CNF $\tilde{C}$ based on the Wang differential path it took CRYPTOMINISAT 520 seconds to find 1000 collisions.

Step 13	0x62c00000
Step 17	0xb6000000
Step 20	0xe0000000
Step 21	0xf0000000

Fig. 4. Differential path for finding single-block collisions for MD4. Only values of differences for chaining variables that are different from that in the path from [38] are presented.

We also applied the SAT approach to the problem of finding preimages for the weakened version of MD4 algorithm. Despite the fact that the problem of finding collisions for MD4 can be effectively solved, the problem of finding preimages for the full version of this function remains computationally hard. We started our research in this direction from reviewing the results of [15], where there were showed that two-round version of MD4 can be easily inverted. In [14] there was used the SAT approach to find preimages of weakened versions of MD4. Again, the results of computational experiments from this work are hard to reproduce due to the use of specialized tools for constructing propositional encodings of the considered functions. In our experiments we used the propositional encoding constructed by means of the Transalg system.

The attack on MD4, described in [15] is an iterative procedure, in which each iteration includes two stages: random selection of some data and calculating hashed message blocks based on the selected data. Randomly generated data is actually supposed to fill the contents of some of the hash registers at certain steps. Thus, the values corresponding to the chaining variable a, assumed to be equal to a constant K on steps $12, 16, 20, 24$. This and similar constraints are assumed also regarding other chaining variables (in total 12 constraints). The constant K is chosen randomly at each new iteration. Besides K at each iteration the value of chaining variable b previous to the calculation of a hash value is randomly selected (we remind that two-round version of MD4 is considered, i.e. 32 steps of algorithm). Then using the available data the blocks of the hashed message are calculated. For this purpose the equations for calculating chaining variables at corresponding steps are used. It is possible that as a result the value of b will not coincide with its randomly selected value. In this case the iteration is considered unsuccessful. If the corresponding values coincide, then the preimage for the hash was found.

In [14] there was proposed a SAT-attack on MD4, based on the ideas of [15]. More precisely, in [14] the constant K was fixed to be equal to 0. The values of the variable b in steps preceding the calculation of a hash were not fixed. Also, the authors of [14] rejected one of the conditions from [15]. Thus, in [14] there were used 11 conditions. The conditions of 'Dobbertin' type were added to the propositional encoding of the MD4 algorithm in the form of unit clauses. The best result presented in [14] was succesful solving of inversion of 39-step version of MD4, which took Minisat SAT solver [16] about 8 hours.

In order to develop the results of [15] and [14] we built a special technique that makes it possible to add to a propositional encoding various additional constraints (such as the conditions of 'Dobbertin' type), vary it and find the best ones in the sense of the effectiveness of inversion problem solving. The main idea of this approach consists in the following. Let C be a CNF that encodes the inversion of some function and X be a set of Boolean variables from C. Assume that we need to add to C new constraints that specify some predicate over variables from a set $\tilde{X}$, $\tilde{X} \subseteq X$. Let $R(\tilde{X})$ be a formula specifying this predicate. Now let us introduce new Boolean variable u, $u \notin X$. Consider formula $C' = C \wedge (\neg u \vee R(\tilde{X}))$. It is clear that the constraint $R(\tilde{X})$ will be inactive when $u = 0$. Let us call the variables similar to u the *switching variables*. Varying values of switching variables it is possible to find a set of additional constraints that makes it possible to significantly decrease the time required to find the preimage of the considered hash image. To work with switching variables one can use various 'learning' techniques embedded into state-of-the-art SAT solvers. In particular it is possible to consider the values of switching variables as the so-called assumptions [17].

Hereinafter we considered the inversion of MD4 hash image

`0xffffffff, 0xffffffff,0xffffffff,0xffffffff`.

At the initial stage of our experiments we implemented the search over combinations of Dobbertin-like conditions with $K = 0$. Thus we tried all possible combinations of values of switching variables for 28 steps of the first two rounds (2^{28} variants in total).

The exhaustive search was performed using 'Academician Matrosov' computing cluster of Irkutsk Supercomputer Center SB RAS (in the experiment we employed 160 cores of Opteron 6276 processors). Each SAT instance produced by fixing active switching variables had a time limit of 0.005 seconds. In total the experiment took about 10 minutes. As a result there were synthesized 5 sets of values of switching variables for which the SAT solver managed to find satisfying assignment within the time limit. Less than 5 % of all SAT instances were interrupted due to the time limit, and more than 94% turned out to be UNSATISFIABLE. Among the 5 found sets of values one of the sets exactly coincided with Dobbertin conditions with $K = 0$. Thus, it can be said that we managed to synthesize the conditions presented in [15] automatically using the SAT solver. After this we considered the problem of inversion for 39-step version of MD4 using Transalg encodings and state-of-the-art multithreaded SAT solvers. The best results were shown by the Treengeling solver [5]: on average to solve one inversion problem for 39-step version of MD4 it took less than 10 minutes of work using one node of 'Academician Matrosov' cluster (32 cores). It is quite surprising that we did not manage to solve inversion problem for 40-step version of MD4 in reasonable time. We believe that we will manage to do it in the nearest future thanks to generation of new 'Dobbertin-like' conditions. For this purpose we will employ the SAT-based cryptanalysis techniques presented in this paper.

5 Related Works

The idea of applying SAT solving algorithms to cryptanalysis problems was first suggested, apparently, in [9]. It is usually assumed that the first practical implementation of this idea was performed in [26]. From that moment on the flow of research in the direction of SAT-based cryptanalysis has been constantly increasing. The first example of successful application of SAT-based cryptanalysis to the cryptographic functions used in the real world can be found in [28]. In a number of later works SAT-based cryptanalysis has also been applied quite successfully. Thus, in [34] the problem of cryptanalysis of the unweakened A5/1 algorithm used to encrypt GSM-traffic was solved using the SAT solvers. In [35] there were given the complexity estimations of SAT-based cryptanalysis of the Bivium cipher. These estimations were better than similar ones for other known methods of cryptanalysis of this cipher. In a recent paper [33] estimations from [35] have been improved.

For the fist time the method that used differential paths to construct collisions of cryptographic hash functions was presented in the papers by X. Wang et.al. [38, 39]. Later there appeared a lot of different works following [38, 39]. Probably one of the most interesting of them is the differential attack by M. Stevens [36] that makes it possible to construct single block collisions for the MD5 hash function.

As we already mentioned, for the first time the problem of finding collisions for cryptographic hash functions was considered as SAT in [22]. However, real progress in this direction has been achieved a year later in [28]. The decisive factor in this direction was the use of SAT encodings of differential paths given in [38, 39].

In the context of finding collisions of hash functions from the MD family the main result of the present paper consists in using SAT to find a previously unknown class of two-block collisions for MD5 where first 10 bytes are zeroes. Also we described the SAT-based method for generating new differential paths based on the known ones. Using one of the paths found by this method, we managed to enumerate MD4 collisions faster than we could do it using the Wang path.

The nontrivial attack on incomplete version of MD4 was first proposed by H. Dobbertin in [15], where it was shown that the first two rounds of MD4 are not one-way. In our paper we automatically synthesized SAT equivalents of Dobbertin constraints by showing that in some sense these constraints are the best possible ones. For this purpose we used a special parallel SAT solver.

6 Conclusions

In this paper we presented the results of application of SAT-based cryptanalysis to inversion problems of the cryptographic functions MD4 and MD5. With regards to finding collisions (single-block for MD4 and two-block for MD5) the effectiveness of proposed methods exceeds that of previous papers. We showed

that SAT approach is applicable to the construction of differential paths for finding collisions of cryptographic hash functions: we managed to construct a new differential path for finding MD4 collisions. Also we applied SAT approach to construct additional constraints that make it possible to improve the effectiveness of finding preimages of truncated variants of the MD4 hash function. Among the automatically synthesized constraints there were constructed the constraints used in [15, 14]. As a concluding remark we would like to note, that SAT-based cryptanalysis is a rapidly developing area. SAT solvers can be used not only for cryptanalysis, but also as oracles for solving various auxiliary problems, for example, to solve nonlinear equations over finite fields in algebraic cryptanalysis [2]. The evident progress of computing architectures, and especially intensive development of parallel computing technologies gives us hope that new results in SAT-based cryptanalysis, including the problems related to cryptographic hash functions, would not take long to appear.

Acknowledgments. The research was suported by Russian Science Foundation (grant №16-11-10046).

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2006)
2. Bard, G.V.: Algebraic Cryptanalysis. Springer Publishing Company, Incorporated, 1st edn. (2009)
3. Beame, P., Kautz, H.A., Sabharwal, A.: Towards understanding and harnessing the potential of clause learning. J. Artif. Intell. Res. (JAIR) 22, 319–351 (2004)
4. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993. pp. 62–73. ACM (1993)
5. Biere, A.: Lingeling essentials, A tutorial on design and implementation aspects of the the SAT solver lingeling. In: Berre, D.L. (ed.) POS-14. Fifth Pragmatics of SAT workshop, July 13, 2014, Vienna, Austria. EPiC Series, vol. 27, p. 88. EasyChair (2014)
6. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
7. Brassard, G. (ed.): Advances in Cryptology - CRYPTO '89, Proceedings, Lecture Notes in Computer Science, vol. 435. Springer (1990)
8. Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA. pp. 151–158. ACM (1971)
9. Cook, S.A., Mitchell, D.G.: Finding hard instances of the satisfiability problem: A survey. In: DIMACS Series in Discr. Math. and Theoretical Comp. Sci. vol. 35, pp. 1–17. American Mathematical Society (1997)

10. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. The MIT Press and McGraw-Hill Book Company (1989)
11. Cramer, R. (ed.): Advances in Cryptology - EUROCRYPT 2005, Proceedings, Lecture Notes in Computer Science, vol. 3494. Springer (2005)
12. Damgård, I.: A design principle for hash functions. In: Brassard [7], pp. 416–427
13. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
14. De, D., Kumarasubramanian, A., Venkatesan, R.: Inversion attacks on secure hash functions using SAT solvers. In: Marques-Silva, J., Sakallah, K.A. (eds.) Theory and Applications of Satisfiability Testing - SAT 2007, Proceedings. Lecture Notes in Computer Science, vol. 4501, pp. 377–382. Springer (2007)
15. Dobbertin, H.: The first two rounds of md4 are not one-way. In: Vaudenay, S. (ed.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 1372, pp. 284–292. Springer Berlin Heidelberg (1998)
16. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT. Lecture Notes in Computer Science, vol. 2919, pp. 502–518. Springer (2003)
17. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electr. Notes Theor. Comput. Sci. 89(4), 543–560 (2003)
18. Erkök, L., Matthews, J.: High assurance programming in cryptol. In: Sheldon, F.T., Peterson, G., Krings, A.W., Abercrombie, R.K., Mili, A. (eds.) Fifth Cyber Security and Information Intelligence Research Workshop, CSIIRW '09, Knoxville, TN, USA, April 13-15, 2009. p. 60. ACM (2009)
19. Erkök, L., Matthews, J.: Pragmatic equivalence and safety checking in cryptol. In: Proceedings of the 3rd ACM Workshop Programming Languages meets Program Verification, PLPV 2009, Savannah, GA, USA, January 20, 2009. pp. 73–82 (2009)
20. Haken, A.: The intractability of resolution. Theor. Comput. Sci. 39, 297–308 (1985)
21. Janicic, P.: URSA: a System for Uniform Reduction to SAT. Logical Methods in Computer Science 8(3), 1–39 (2012)
22. Jovanovic, D., Janicic, P.: Logical analysis of hash functions. In: Gramlich, B. (ed.) Frontiers of Combining Systems, 5th International Workshop, FroCoS 2005, Proceedings. Lecture Notes in Computer Science, vol. 3717, pp. 200–215. Springer (2005)
23. Kernighan, B.W., Ritchie, D.: The C Programming Language, Second Edition. Prentice-Hall (1988)
24. King, J.C.: Symbolic execution and program testing. Commun. ACM 19(7), 385–394 (Jul 1976)
25. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Trans. Computers 48(5), 506–521 (1999)
26. Massacci, F., Marraro, L.: Logical cryptanalysis as a SAT problem. J. Autom. Reasoning 24(1/2), 165–203 (2000)
27. Merkle, R.C.: A certified digital signature. In: Brassard [7], pp. 218–238
28. Mironov, I., Zhang, L.: Applications of SAT solvers to cryptanalysis of hash functions. In: Biere, A., Gomes, C.P. (eds.) SAT. Lecture Notes in Computer Science, vol. 4121, pp. 102–115. Springer (2006)
29. Otpuschemmikov, I., Semenov, A., Kochemazov, S.: Transalg: a tool for translating procedural descriptions of discrete functions to SAT. In: WCSE 2015-IPCE: Proceedings of The 5th International Workshop on Computer Science and Engineering: Information Processing and Control Engineering. pp. 289–294 (2015)

30. Otpuschennikov, I., Semenov, A., Gribanova, I., Zaikin, O., Kochemazov, S.: Encoding cryptographic functions to SAT using Transalg system. In: Kaminka, G.A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum, V., Dignum, F., van Harmelen, F. (eds.) ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016). Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1594–1595. IOS Press (2016)
31. Rivest, R.: The md5 message-digest algorithm (1992)
32. Rivest, R.L.: The MD4 message digest algorithm. In: Menezes, A., Vanstone, S.A. (eds.) Advances in Cryptology - CRYPTO '90, Proceedings. Lecture Notes in Computer Science, vol. 537, pp. 303–311. Springer (1990)
33. Semenov, A., Zaikin, O.: Algorithm for finding partitionings of hard variants of boolean satisfiability problem with application to inversion of some cryptographic functions. SpringerPlus 5(1), 1–16 (2016)
34. Semenov, A., Zaikin, O., Bespalov, D., Posypkin, M.: Parallel Logical Cryptanalysis of the Generator A5/1 in BNB-Grid System. In: Malyshkin, V. (ed.) PaCT. Lecture Notes in Computer Science, vol. 6873, pp. 473–483. Springer (2011)
35. Soos, M., Nohl, K., Castelluccia, C.: Extending SAT solvers to cryptographic problems. In: Kullmann, O. (ed.) SAT. Lecture Notes in Computer Science, vol. 5584, pp. 244–257. Springer (2009)
36. Stevens, M.: Single-block collision attack on MD5. IACR Cryptology ePrint Archive 2012, 40 (2012)
37. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Siekmann, J., Wrightson, G. (eds.) Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970, pp. 466–483. Springer, Berlin, Heidelberg (1983)
38. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions MD4 and RIPEMD. In: Cramer [11], pp. 1–18
39. Wang, X., Yu, H.: How to break MD5 and other hash functions. In: Cramer [11], pp. 19–35