Applications of SAT Solvers to Cryptanalysis of Hash Functions

Ilya Mironov and Lintao Zhang

Microsoft Research, Silicon Valley Campus
{mironov, lintaoz}@microsoft.com

Abstract. Several standard cryptographic hash functions were broken in 2005. Some essential building blocks of these attacks lend themselves well to automation by encoding them as CNF formulas, which are within reach of modern SAT solvers. In this paper we demonstrate effectiveness of this approach. In particular, we are able to generate full collisions for MD4 and MD5 given only the differential path and applying a (minimally modified) off-the-shelf SAT solver. To the best of our knowledge, this is the first example of a SAT-solver-aided cryptanalysis of a non-trivial cryptographic primitive. We expect SAT solvers to find new applications as a validation and testing tool of practicing cryptanalysts.

1 Introduction

Boolean Satisfiability (SAT) solvers have achieved remarkable progress in the last decade [MSS99, MMZ$^+$01, ES03]. The record-breaking performance of the state-of-the-art SAT solvers opens new vistas for their applications beyond what conventionally has been thought feasible. Still, most of the successful real world applications of SAT solvers belong to the traditional domains of formal verification and AI. In this paper we explore applications of SAT solvers to cryptanalysis of hash functions.

Several applications of SAT solvers to cryptanalysis have been described in the literature [Mas99, MM00, FMM03, JJ05]. Their strategy can be regarded as a "head-on" approach, in the sense that they are not using any new or existing cryptanalytic methods in their attacks. Unsurprisingly, these efforts failed to produce any attacks of interest to cryptologists.

Despite the previous (arguably unsuccessful) attempts, we are convinced that SAT solvers could be of use in practical cryptanalysis. Our strategy may be described as "meet-in-the-middle": after initial, highly creative work of cryptanalysts, we are able to delegate the more laborious parts of the attack to the SAT solver.

Recently, several important cryptographic hash functions were shown to be vulnerable to collision-finding attacks [WY05, WYY05b]. The original attacks consisted of several steps each of which involves a lot of bit-tweaking and manual work. It suffices to say that the attack on the simplest function of the family, MD4, requires keeping track of as many as 122 boolean conditions.

A. Biere and C.P. Gomes (Eds.): SAT 2006, LNCS 4121, pp. 102–115, 2006.

In this paper, we show that SAT solvers can be used to automate certain elements of these attacks. In particular, we demonstrate that SAT solvers may obviate the need for compiling tables of sufficient conditions and designing clever message-modifications techniques. Our successful attacks on MD4 and MD5 suggest that SAT solvers could be a valuable addition to cryptanalysts' toolkit.

The paper is structured as follows. Section 2 is a short primer on theory and practical constructions of hash functions. Section 3 covers recent attacks on hash functions; Section 4 presents experimental results of applying SAT solvers to automation of these attacks. We conclude in Section 5.

2 Theory and Constructions of Hash Functions

Cryptographic hash functions are essential for security of many protocols. Early applications of hash functions in systems security include password tables [JKW74] and signature schemes [RSA78, Lam79]; since then virtually any cryptographic protocol uses directly or indirectly a secure hash function as a building block.

The properties required of a secure hash function differ and often depend on the protocol in question. Still, the property of being *collision-resistant* is recognized as the "gold standard" of security of hash function. The first formal definition of collision-resistant hash functions (CRHF) was given by Damgård [Dam88]. A function H is said to be collision-resistant if it is infeasible to find two different inputs x, y such that $H(x) = H(y)$. Since any compressing function has collisions, a guarantee of collision-resistance may only be computational.

A first standard hash function, MD4, was designed by Ron Rivest [Riv91]; its strengthened version MD5 followed shortly thereafter [Riv92]. A first NIST-approved hash function, SHA (Secure Hash Algorithm), adopted the general structure (and even some constants!) of MD4 [NIS93]; it was withdrawn in 1995 and replaced with a new version, dubbed SHA-1 [NIS95], that differed in one additional instruction. To avoid confusion, the original SHA is commonly referred to as SHA-0. As of 2004, two hash functions were in wide-spread (and almost exclusive) use: MD5 and SHA-1. It is fair to say that all of these functions belong to one family that shares similar design principles.

Compression function. The basic construction block of CRHFs is a collision-resistant *compression function*, which maps a fixed-length input into a shorter fixed-length output.

The heart of the construction is a *block cipher*, which is defined as a function of two inputs $E\colon \{0,1\}^k \times \{0,1\}^n \mapsto \{0,1\}^n$. Although $E(\cdot,\cdot)$ compresses its input by mapping $k+n$ bits into k bits, as is it is trivially invertible. However, the following trick, called the Davies-Meyer construction, results in a CRHF F under the assumption that E is an ideal block cipher (i.e., $E(x,\cdot)$ is an indexed collection of random permutations on $\{0,1\}^n$):

$$F(M,x) = E(x,M) \oplus M.$$

Among several methods for constructing block ciphers, the *Feistel ladder* is by far the best-known, being the method of choice for DES. The (unbalanced) Feistel ladder is a foundation of all block ciphers inside MDx and SHAx families. It is an iterative method, which consists of two separate components, a key-expansion algorithm and a collection of round functions. The ladder is parameterized by the number of rounds r and the size of the state. Assume for concreteness that the state consists of four 32-bit words (as the case of MD4 and MD5). The state goes through r rounds of transformation; let the initial, intermediate, and final states be (a_i, b_i, c_i, d_i) for $i \in \{0, \dots, r\}$. The key-expansion algorithm K maps M to r round keys denoted $K(M) = w_0, \dots, w_{r-1}$:

$$K\colon \{0,1\}^n \mapsto \underbrace{\{0,1\}^{32} \times \cdots \times \{0,1\}^{32}}_{r \text{ times}}.$$

Round functions $f_i\colon \{0,1\}^{128} \mapsto \{0,1\}^{32}$ for $i \in \{0, \dots, r-1\}$ are used to update the state. MD5's transformation is one example (k_i and s_i are constants):

$$(a_{i+1}, b_{i+1}, c_{i+1}, d_{i+1}) \leftarrow \\ (d_i, b_i + (a_i + f(b_i, c_i, d_i) + w_i + k_i) \lll s_i, b_i, c_i).$$

Notice that the transformation is reversible if the round key w_i is known.

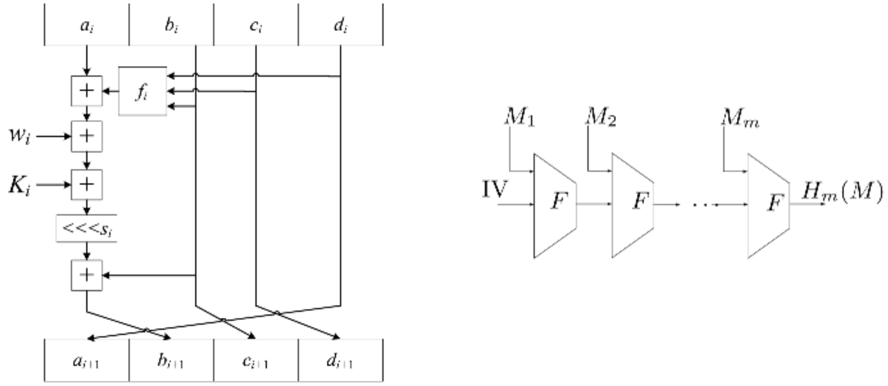

Fig. 1. One round of Feistel ladder for MD5 and the Merkle-Damgård construction

Merkle-Damgård paradigm. CRHFs are expected to take inputs of arbitrary length. A composition of fixed-length compression functions, discussed above, preserves the collision-resistance property. This method is called the Merkle-Damgård construction [Mer90, Dam90] (see Figure 1).

Generic Attacks. Generic attacks against hash functions are oblivious to the particulars of their constructions—they treat hash functions as black-boxes and in general provide an upper bound on security of different cryptographic properties.

We contrast two generic attacks on hash functions. Both attacks find two messages $x \neq y$ such that $H(x) = H(y)$. In the first attack the two messages are unrestricted, and thanks to the birthday paradox its generic complexity is $2^{n/2}$, where the output length of H is n bits. The goal of the second attack is to find colliding x and y such that $x = y \oplus \delta$ for some fixed δ. The generic complexity of this attack is 2^n.

Practical hash functions. Following the nomenclature developed above, designs of MD4, MD5, SHA-0, and SHA-1 hash functions follow the Merkle-Damgård paradigm, making use of a compression function built via the Davies-Meyer construction from block ciphers of the unbalanced Feistel ladder-type.

The internal state of the compression function consists of four 32-bit words (a_i, b_i, c_i, d_i) for MD4 and MD5 and five 32-bit words $(a_i, b_i, c_i, d_i, e_i)$ for SHA-0 and SHA-1. Since the size of the internal state is also the size of the output of the hash function, the output length of MD4 and MD5 is 128 bits; SHA-0 and SHA-1 produce 160-bit outputs. MD4 applies 48 rounds of the Fiestel transform, MD5 has 64, and SHA-0,1 both use 80 rounds. For details of the constructions, including the round functions, the key expansion algorithms, and the constants, omitted in the interest of brevity, we refer the reader to [MvOV96] or corresponding standards.

Commonly held belief in security of MD5 and SHA-1 had been supported by a relative absence of attacks on these and related functions. Although a collision in MD4 was discovered in 1996 [Dob96a], some weaknesses were identified in MD5 and SHA-0 [dBB94, Dob96b, BC04] and a theoretical attack was known on SHA-0 [CJ98], no collisions had been found for MD5 and SHA-1 despite more than ten years of intense scrutiny.

The year 2005 brought about a sea change in our understanding of hash functions. A new and improved attack on MD4 [WLF$^+$05], collisions for MD5 and SHA-0 [WY05, WYY05b], and a theoretical attack on SHA-1 [WYY05a] were announced by a group of Chinese researchers led by Xiaoyun Wang in two consecutive conferences. Independently of them, attacks on SHA-0 and reduced-round SHA-1 were discovered by Biham et al. [BCJ$^+$05].

Most of these attacks are conceptually simple but their implementations tend to be extremely laborious. Although there is considerable interest in generalizing the attacks and applying them in other contexts, the required amount of manual work may be unsurmountable. We observe that some components of the attacks may be expressed as CNF formulas and be rather efficiently solved by advanced SAT solvers.

3 Attacks on Hash Functions

In this section we develop notation and a common framework describing collision-finding attacks on hash functions.

3.1 Notation

Throughout this section and in the rest of the paper, whenever we are trying to find a collision between $M = (m_0, \ldots, m_{15})$ and $M' = (m'_0, \ldots, m'_{15})$, variables w_i, a_i, b_i, c_i, d_i, e_i refer to the computation of the compression function on input M and their primed counterparts w'_i, a'_i, b'_i, c'_i, d'_i, e'_i to the computation of the same function on M'.

We will be interested in two types of *differentials*: in respect to XOR and in respect to difference modulo 2^{32}. Define

$$\Delta^+ a_i = a_i - a'_i \pmod{2^{32}} \text{ and similarly } \Delta^+ w_i, \Delta^+ m_i, \Delta^+ b_i, \ldots [, \Delta^+ e_i]$$

and

$$\Delta^{\oplus} a_i = a_i \oplus a'_i \text{ and similarly } \Delta^{\oplus} w_i, \Delta^{\oplus} m_i, \Delta^{\oplus} b_i, \ldots, [\Delta^{\oplus} e_i].$$

Let

$$\Delta^{\oplus}_i = (\Delta^{\oplus} a_i, \Delta^{\oplus} b_i, \Delta^{\oplus} c_i, \Delta^{\oplus} d_i, [\Delta^{\oplus} e_i]), \text{ and similarly} \Delta^+_i.$$

The sequence $\Delta^{\oplus}_0, \Delta^{\oplus}_1, \Delta^{\oplus}_2, \ldots$ (resp., $\Delta^+_0, \ldots$) is a called a *differential path* in respect to the XOR differential (resp., to the difference modulo 2^{32}). In the rest of the paper, $\circ$ stands for both $+$ and $\oplus$.

Strictly speaking, the attacks due to Wang et al. fix the exact settings for most of the differing bits, subsuming both differentials. Our encodings fully use this information.

3.2 Overview of the Attacks

Conceptually, the attacks on MDx and SHA-0,1 have a lot in common. Most remarkably, the attacks solve a seemingly harder problem, i.e., finding a 512-bit message such that $H(\mathrm{IV}, M) = H(\mathrm{IV}, M \circ \delta)$, where H is the compression function and δ is fixed.[1] As observed earlier, the generic complexity of this attack (the one that uses the function as a black-box) is 2^n, where $n = 128$ or 160. A judicious choice of δ and a collection of clever techniques for finding M that take advantage of the weaknesses of the compression function bring the complexity of the attack to fewer than 2^{42} evaluations of the hash function.

Conceptually, the attacks consist of four distinct stages.

Stage I. Choose $\Delta^\circ m_0, \ldots, \Delta^\circ m_{15}$.
Stage II. Choose a differential path $\Delta^\circ_0, \ldots, \Delta^\circ_{r-1}$, where r is the number of rounds ($r = 48, 64$ or 80).
Stage III. Find a set of *sufficient conditions* on the message $M = (m_0, \ldots, m_{15})$ and the intermediate variables $a_i, \ldots, d_i$ that guarantee (with high probability) that the message pair M, $M' = (m_0 \circ \Delta^\circ m_0, \ldots, m_{15} \circ \Delta^\circ m_{15})$ follows the differential path $\Delta^\circ_0, \ldots, \Delta^\circ_{r-1}$.
Stage IV. Choose a message M such that all sufficient conditions hold.

[1] Some attacks solve the problem in two steps: they seek to find two messages M_0 and M_1, and differences δ, δ_0, and δ_1 such that $H(\mathrm{IV}, M_0) = H(\mathrm{IV}, M_0 \circ \delta_0) \circ \delta$ and $H(H(\mathrm{IV}, M_0), M_1) = H(H(\mathrm{IV}, M_0 \circ \delta_0), M_1 \circ \delta_1)$.

The attacks may seem counter-intuitive: rather than finding *any* two messages that collide under the hash function, we first severely constrain the space of possible message-pairs by fixing their difference, and by choosing the differential path we restrict the space of possible solutions even further.

There are two reasons that make this approach work. First, the differential path is carefully chosen to maximize the probability of a collision. Second, by deliberately constraining the solution space, we know some important properties of the solution, which allow us to construct one by an iterative process.

The first stage of the attack is usually done by hand or by applying some heuristics, such as trying to find a difference with low Hamming weight.

The second stage is the most creative stage of all four.There is fair amount of flexibility in it, for the curios reason that, as the round functions are non-linear, a given difference in the input may result in many possible differences in the output. Ironically, the attack turns the very foundation of security of hash functions—non-linearity of the round transformation—to its advantage. There are several constraints imposed on the differential path. First, it must be feasible. Further, it must be likely and finally it should facilitate the third stage of the attack.

The third stage is tightly coupled with the previous one. Most sufficient conditions naturally follow from the properties of the round function.

The fourth stage is computationally most intensive. Spectacular attacks due to Wang et al. would not be possible with a breakthrough in this stage. Indeed, a random pair of messages M and M' is very unlikely to follow the differential path. This is where the recent attacks depart most radically from earlier work. The idea is to start with an arbitrary message M and then carefully "massage" it into the differential path. The crucial contribution of Wang et al. was to come up with a set of tools of fixing errors in the differential path one by one.

We claim that SAT solvers might be very helpful in automating the third and the fourth stages of the attack. Since the actual attack requires a lot of iterations between the second stage and the next two, any method that would speed up testing and validation of differential paths becomes a useful cryptanalytic tool.

3.3 Attacks on MD4 and MD5

The terse exposition of the attacks due to Wang et al. was lacking some details and very short on intuition. Several papers attempted to explain, fill in omitted details, correct, improve, and automate these attacks [HPR04, Kli05, Dau05, BCH06, SO06]. In particular, we refer to Black et al. and Oswald et al. [BCH06, SO06] for intuition on the discovery process of the differential path for MD4 and MD5 (Stages I and II of our framework).

For ease of exposition, examples below are given for the attack on MD5 [WLF$^+$05]. Attacks on MD4, SHA-0 and (reduced-round) SHA-1 are similar, with the main difference being usage of XOR differentials for the SHAx functions instead of differentials in respect to subtraction for MDx.

Stage III of the attacks produces a list of probabilistically sufficient conditions on the internal variables for the differential path to hold (of the type: lsb of c_7 is 0 and the 11th bit of a_7 is 1). There are as many as 310 of them; fortunately, most of these conditions appear in the first 25 rounds. If all conditions are met, the differential path is very likely to hold as well.

Successful completion of Stage IV of the attack is ensured by a combination of *single-message* and *multi-message modification techniques* and a probabilistic argument. More precisely, conditions in the first 16 rounds of MD5 can be fixed by going from b_{i+1}, chosen to satisfy all conditions, to m_i, as follows:

$$m_i \leftarrow ((b_{i+1} - b_i) \ggg s_i) - K_i^{(5)} - a_i - f(b_i, c_i, d_i).$$

This operation is called a single-message modification as it only affects one block of the message. Unfortunately, if we try to apply this method to correcting errors in later rounds, we are most likely to introduce new errors in earlier rounds.

Multi-message modifications do just that—they correct errors in the differential path in rounds beyond 16, by changing several blocks of the message at a time. The example given in [WY05] and discussed in more details in [BCH06] is (roughly) the following.

Suppose, we would like to force the msb of b_{17} be 1. This can be achieved by modifying $w_{16} = m_1^* \leftarrow m_1 + 2^{26}$. This modification changes the value of b_2 to b_2^* (and, since $b_2 = c_3 = d_4 = a_5$, other values as well), which is where m_1 was used for the first time. To absorb the change, we modify

$$\begin{aligned}
m_2^* &\leftarrow ((b_3 - b_2^*) \ggg 17) - a_2 - f(b_2^*, c_2, d_2) - K_2^{(5)},\\
m_3^* &\leftarrow ((b_4 - b_3^*) \ggg 22) - a_3 - f(b_3, c_3^*, d_3) - K_3^{(5)},\\
m_4^* &\leftarrow ((b_5 - b_4^*) \ggg 7) - a_4 - f(b_4, c_4, d_4^*) - K_4^{(5)},\\
m_5^* &\leftarrow ((b_6 - b_5^*) \ggg 12) - a_5^* - f(b_5, c_5, d_5) - K_5^{(5)}.
\end{aligned}$$

As a result, none of the values computed in rounds 1–5 got changed; only the message blocks $m_1,\ldots,m_5$, intermediate variables $b_2 = c_3 = d_4 = a_5$ and $b_{17} = c_{18} = d_{19} = a_{20}$ did.

This is a representative example of a multi-message modification method, although by no means the trickiest. None of this methods expand beyond round 22, and they become progressively more complicated as more message blocks get affected.

Since as many as 37 conditions cannot be fixed using message modifications, Wang et al. fall back on the probabilistic method. Since all conditions are satisfied (heuristically) with probability 2^{-37}, it suffices to apply known message modifications to random messages 2^{37} times to have a non-trivial chance of generating a collision. Subsequent papers enriched the toolkit of message modifications, reducing the number of conditions that need to be satisfied probabilistically to around 30 [Kli05, BCH06].

4 Automation Via SAT Solvers

Arguably, the most annoying aspects of the attacks due to Wand et al. is the need to keep track of hundreds of conditions with a corresponding code to take care of them by means of message modifications. Our approach completely eliminates this difficulty, as we encode the entire differential path and leave the task of finding a message satisfying it to the SAT solver.

In this section, we report our preliminary results on using SAT solvers to implement the attacks due to Wang et al.

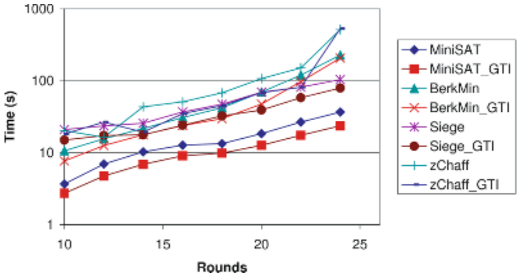

Fig. 2. Comparison of Different Solvers on MD5 Instances

To apply SAT solver to cryptanalysis of hash functions, we need to transform the hash functions' code into boolean circuits and perform CNF clausification. We experimented with several boolean circuit implementations for the adders and multiplexors, and found that different implementations of these primitives may have some tangible effect on the size of the resulting CNFs and efficiency of SAT solvers applied to these formulas. In this section, we report results on using a simple full-adder implementation and a multiplexer implemented with two AND gate and one OR gate. We found that this combination works reasonably well in our experiments. For clausification, currently we use the straightforward Tseitin transformation [Tse68] on the propositional formulas. We believe that more optimization can be obtained from careful examination of the encoding process [ES06].

Since we are dealing exclusively with satisfiable instances, we evaluated a number of stochastic and complete SAT solvers. We found that stochastic methods were not suitable for this task, which was unsurprising because the instances were highly structured with complicated interactions among the variables. For the complete solvers, we experimented with several state-of-the-art DLL search-based solvers, including the latest versions of ZChaff [MMZ$^+$01], MiniSAT [ES03], BerkMin [GN02], and Siege [Rya04]. Each solver was tested by itself and with the SatELite preprocessor [EB05]. Figure 2 gives the average runtime of these solvers for finding a first solution of MD5. Each data point is the average of 30 runs with randomly generated IV vectors. The SatELite preprocessor followed by MiniSat (collectively known as SatELiteGTI) produced best results, which are reported in this section.

Wang et al.'s attacks on MD4 and MD5. The sizes of the resulting formulas appear in Table 1.

Table 1. Applying SAT solvers to MD4, MD5, and SHA-0

Hash	Total	Modifications		Formula	
	Rnd	Manual	SAT	Vars.	Cls
MD4	48	all	all	53228	221440
MD5	64	22	46	89748	375176
SHA-0	80	20	35	114809	486185

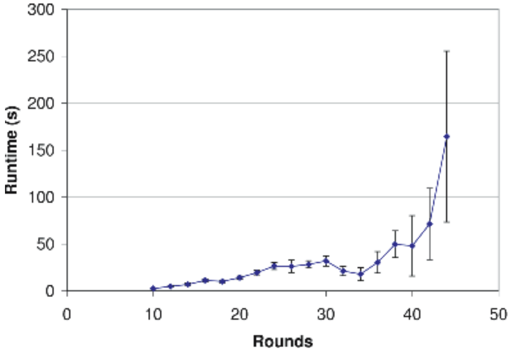

Fig. 3. Mean running time over 100 runs for MD5

Another information given in Table 1 is the number of rounds for which SatELiteGTI can find a message satisfying the differential in the median time of less than 15 min on a PC (3.2GHz PIV, 1Gb RAM) versus the number of rounds for which manual message modifications techniques are known.

As implied by Table 1, finding a collision in MD4 is easy and usually takes less than 10 minutes (the median value is approximately 500 s).

Making SatELiteGTI work for the full MD5 is less straightforward. Recall that for rounds beyond the limits of the message modifications techniques, Wang et al. use a probabilistic argument, whose running time increases exponentially with the number of sufficient conditions in the additional rounds. Although the behavior of SatELiteGTI is more complex, we discover that the best mode of operation for the solver is to follow exactly the same paradigm! The slowdown experienced by SatELiteGTI as a function of the number of rounds makes it more economical to run the solver for $n < 64$ rounds and than screen the results for solutions that hold for more rounds. The optimal value for n is 25 rounds. In other words, $n = 25$ minimizes the ratio of the number of solutions for the n-round MD5 produced in one second to the probability that a solution for n rounds will be a solution for the full MD5.

We modified SatELiteGTIto produce multiple solutions for a single SAT instance. We observe that finding subsequent solutions is much faster than finding the first solution (the runtime for finding the first solution is shown in Figure 3). In Figure 4, we plot the number of conflicts encountered between the first

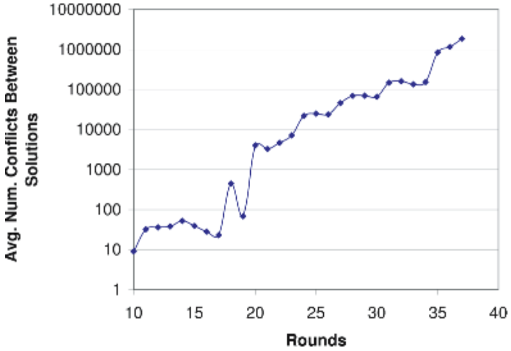

Fig. 4. Num. of Conflicts for Finding 10000 Solutions in MD5

and the 10000th solution. This graph illustrates the inverse density of solutions in the boolean space as explored by the SAT solver. For example, at round 25, on the average the solver encounters about 15 conflicts before finding a solution. We can generate around 350 solutions per second on our machine. The probability that one solution that follows the differential path for 25 rounds will lead to a collision is approximately 2^{-27}. Finding a solution for the full MD5 (first block of the attack) takes approximately 100 hours, which is comparable, albeit slower than an hour on an IBM supercomputer required by the original attack (the best known attack takes approximately 8 minutes [BCH06]). Finding solutions for the second block is significantly faster (less than an hour on our PC).

An interesting result of our experiments with SAT solvers is the importance of having a differential path encoded in the formula. Discovering a differential part is a difficult process (Stage II of the framework), which ideally should be automated. Currently, the only method of automatic enumeration of differential paths is known for MD4 [SO06], extending it to other hash functions is an open problem. Our experiments with applying SAT solvers to this task were not very encouraging. Specifically, absent restrictions imposed by the differential path the SAT solver performs much worse than in the presence of these restrictions.

Dobbertin's attack on MD4. We automated Dobbertin's attack on MD4 [Dob96a], which is considerably weaker than the more recent attacks. The reader is referred to Dobbertin's paper for details of the attack. The most difficult part of Dobbertin's method is finding a solution to a non-linear system of equations defined by rounds 12–19 of the hash function. He describes an ad hoc iterative process that starts with a random assignment and then fixes errors four bits at a time. We observe that this part may be encoded as a CNF formula *without doing any additional transformations.* Namely, we translate the rounds 12–19 of the compression function of MD4 into a CNF formula with 10364 variables and 39482 clauses. In fact, we may do it in more generality than Dobbertin, by accepting any initial value for $(a_{12}, b_{12}, c_{12}, d_{12})$, which simplifies Part 2 of his attack.

SatELiteGTI finds 2^{22} solutions to the resulting formula in less than one hour on a PC. This attack is again slower than the original attack, but has the added benefit of being substantially simpler and conceptually simpler.

Wang et al.'s attack on SHA-0. Translating the first two stages of the attack due to Wang et al. on SHA-0 into a CNF formula is analogous to the similar task for MD4 and MD5. The optimal number of rounds after which the probabilistic method outdoes the solver is 20, after which the probability that a solution results in a collision is 2^{-42}. The size of the formula for 20 rounds is 29217 variables and 117169 clauses. Based on several test runs, we estimate that generating a full collision using SatELiteGTI would require approximately 3 million CPU hours.

5 Conclusion and Directions for Future Work

In this work, we describe our initial results on using SAT solvers to automate certain components of cryptanalysis of hash functions of the MDx and SHAx families. Our implementations are considerably easier (but also slower) than the originals, since we delegate many minute details of the attacks to the SAT solver.

In particular, our method has no use for sufficient conditions and message-modification techniques, which are automatically (and implicitly) discovered by the SAT solver. We conjecture that finding a differential path, a formidable problem in itself, may be simplified if one is able to quickly test feasibility of the path without having to compile and consult tables with sufficient conditions.

Our results can be summarized as follows. We are able to launch a complete attack on MD4 by applying SatELiteGTI to the most straightforward encoding of a differential path as a CNF formula, finding a solution in less than 10 minutes. Finding a collision in MD5 follows closely the probabilistic method of Wang et al.: the SAT solver generates a lot of solutions for truncated MD5, which are then filtered in order to find a solution for the full hash function. This attack generates one collision in approximately 100 hours. A successful SAT-solver-aided attack on SHA-0 is still a theoretical possibility.

We rely in our experiments on an off-the-shelf SAT solver, SatELiteGTI, with a simple modification to allow generation of multiple solutions. To the extent of our knowledge, this is a first work that demonstrates the practicality of using SAT solvers for real cryptanalysis.

This work opens many intriguing possibilities. First, we currently use general-purpose SAT solvers. Optimizing and specializing them for cryptographic applications are interesting research directions. For example, one optimization may take advantage of the fact that many internal signals are likely to be correlated between the two input messages. Faster SAT solvers may directly translate into better cryptanalyses.

Second, the primitives studied in this work are restricted to modular addition and bit-wise boolean operations, which allow simple translation into CNF formulas. Multiplications and table lookups are widely used in construction of other cryptographic primitives. It is well known these operations pose a challenge for SAT solvers. There are some known methods for tackling such operations in a SAT solver. One approach is to use CAD tools to synthesize lookup table before translation into CNF [MM00]. Another method consists of rearranging signals during search to handle arithmetic operations more efficiently [WSK04]. The effectiveness of these approaches in cryptanalytic context is largely unexplored.

Finally, we observe that the probabilistic method of generating solutions for truncated hash function and screening them for a complete solution, a necessity in the case of Wang et al. attacks, might be avoidable in the case of a SAT solver. Indeed, there are no reasons why SAT solvers should experience a dramatic slowdown on large formulas compared to the method that uses *the same* SAT solver to produce multiple solutions for a shorter formula and test them by substitution into the longer formula. We find the current behavior counter-intuitive and amusing.

References

[BC04] Eli Biham and Rafi Chen. Near-collisions of SHA-0. In Matthew K. Franklin, editor, *Advances in Cryptology—CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2004.

[BCH06] John Black, Martin Cochran, and Trevor Highland. A study of the MD5 attacks: Insights and improvements. In *Fast Software Encryption 2006*, 2006.

[BCJ$^+$05] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and reduced SHA-1. In Cramer [Cra05], pages 36–57.

[Bra90] Gilles Brassard, editor. *Advances in Cryptology—CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20–24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[CJ98] Florent Chabaud and Antoine Joux. Differential collisions in SHA-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

[Cra05] Ronald Cramer, editor. *Advances in Cryptology—EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22–26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

[Dam88] Ivan Damgård. Collision free hash functions and public key signature schemes. In David Chaum and Wyn L. Price, editors, *Advances in Cryptology—EUROCRYPT '87*, volume 304 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 1988.

[Dam90] Ivan Damgård. A design principle for hash functions. In Brassard [Bra90], pages 416–427.

[Dau05] Magnus Daum. *Cryptanalysis of Hash Functions of the MD4-Family*. PhD thesis, Ruhr-Universität Bochum, 2005.

[dBB94] Bert den Boer and Antoon Bosselaers. Collisions for the compressin function of MD5. In Tor Helleseth, editor, *Advances in Cryptology—EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 1994.

[Dob96a] Hans Dobbertin. Cryptanalysis of MD4. In Dieter Gollmann, editor, *Fast Software Encryption '96*, volume 1039 of *Lecture Notes in Computer Science*, pages 53–69. Springer, 1996.

[Dob96b] Hans Dobbertin. The status of MD5 after a recent attack. *CryptoBytes*, 2(2):1–6, 1996.

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the International Symposium on the Theory and Applications of Satisfiability Testing (SAT)*, pages 61–75, 2005.

[ES03] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Proceedings of the International Symposium on the Theory and Applications of Satisfiability Testing (SAT)*, 2003.

[ES06] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *J. Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[FMM03] Claudia Fiorini, Enrico Martinelli, and Fabio Massacci. How to fake an RSA signature by encoding modular root finding as a SAT problem. *Discrete Applied Mathematics*, 130(2):101–127, 2003.

[GN02] Evgueni Goldberg and Yakov Novikov. BerkMin: A fast and robust Sat-solver. In *DATE*, pages 142–149, 2002.

[HPR04] Philip Hawkes, Michael Paddon, and Gregory G. Rose. Musings on the Wang et al. MD5 collision. Cryptology ePrint Archive, Report 2004/264, 2004. `http://eprint.iacr.org/`.

[JJ05] Dejan Jovanović and Predrag Janičić. Logical analysis of hash functions. In Bernhard Gramlich, editor, *Frontiers of Combining Systems (FroCoS)*, volume 3717 of *Lecture Notes in Artificial Intelligence*, pages 200–215. Springer Verlag, 2005.

[JKW74] Arthur Evans Jr., William Kantrowitz, and Edwin Weiss. A user authentication scheme not requiring secrecy in the computer. *Commun. ACM*, 17(8):437–442, 1974.

[Kli05] Vlastimil Klima. Finding MD5 collisions on a notebook PC using multi-message modifications. Cryptology ePrint Archive, Report 2005/102, 2005. `http://eprint.iacr.org/`.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, October 1979.

[Mas99] Fabio Massacci. Using Walk-SAT and Rel-SAT for cryptographic key search. In *International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 290–295, 1999.

[Mer90] Ralph C. Merkle. One way hash functions and DES. In Brassard [Bra90], pages 428–446.

[MM00] Fabio Massacci and Laura Marraro. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24:165–203, 2000.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference (DAC)*, June 2001.

[MSS99] João P. Marques-Silva and Karem A. Sakallah. GRASP—a search algorithm for propositional satisfiability. *IEEE Transactions in Computers*, 48(5):506–521, May 1999.

[MvOV96] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[NIS93] NIST. Secure hash standard. FIPS PUB 180, National Institute of Standards and Technology, May 1993.

[NIS95] NIST. Secure hash standard. FIPS PUB 180-1, National Institute of Standards and Technology, April 1995.

[Riv91] Ronald L. Rivest. The MD4 message digest algorithm. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology—CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer, 1991.

[Riv92] Ronald L. Rivest. The MD5 message-digest algorithm. RFC 1321, The Internet Engineering Task Force, April 1992.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[Rya04] Lawrence Ryan. Efficient algorithms for clause-learning SAT solvers. M.Sc. Thesis, Simon Fraser University, February 2004.

[Sho05] Victor Shoup, editor. *Advances in Cryptology—CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14–18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.

[SO06] Martin Schläffer and Elisabeth Oswald. Searching for differential paths in MD4. In *Fast Software Encryption 2006*, 2006.

[Tse68] Gregory Tseytin. *On the complexity of derivation in propositional calculus.*, pages 115–125. Consultant Bureau, New York-London, 1968.

[WLF$^+$05] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In Cramer [Cra05], pages 1–18.

[WSK04] Markus Wedler, Dominik Stoffel, and Wolfgang Kunz. Arithmetic reasoning in DPLL-based SAT solving. In *DATE*, pages 30–35, 2004.

[WY05] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Cramer [Cra05], pages 19–35.

[WYY05a] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In Shoup [Sho05], pages 17–36.

[WYY05b] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In Shoup [Sho05], pages 1–16.