Adaptive Restart and CEGAR-based Solver for Inverting Cryptographic Hash Functions

Saeed Nejati, Jia Hui Liang, Vijay Ganesh,
Catherine Gebotys and Krzysztof Czarnecki

University of Waterloo, Waterloo, ON, Canada

Abstract

SAT solvers are increasingly being used for cryptanalysis of hash functions and symmetric encryption schemes. Inspired by this trend, we present MapleCrypt which is a SAT solver-based cryptanalysis tool for inverting hash functions. We reduce the *hash function inversion problem for fixed targets* into the satisfiability problem for Boolean logic, and use MapleCrypt to construct preimages for these targets. MapleCrypt has two key features, namely, a multi-armed bandit based adaptive restart (MABR) policy and a counterexample-guided abstraction refinement (CEGAR) technique. The MABR technique uses reinforcement learning to adaptively choose between different restart policies during the run of the solver. The CEGAR technique abstracts away certain steps of the input hash function, replacing them with the identity function, and verifies whether the solution constructed by MapleCrypt indeed hashes to the previously fixed targets. If it is determined that the solution produced is spurious, the abstraction is refined until a correct inversion to the input hash target is produced. We show that the resultant system is faster for inverting the SHA-1 hash function than state-of-the-art inversion tools.

1 Introduction

Over the last 15 years we have seen a dramatic improvement in the efficiency of conflict-driven clause-learning (CDCL) SAT solvers [35, 43, 2, 7] over industrial instances generated from a large variety of applications such as verification, testing, security, and AI [11, 9, 46]. Inspired by this success many researchers have proposed the use of SAT solvers for cryptanalysis of hash functions and symmetric encryption schemes [41]. The use of SAT solvers in this context holds great promise as can be seen based on their success to-date in automating many aspects of analysis of cryptographic primitives [51]. SAT solvers are increasingly an important tool in the toolbox of the practical cryptanalyst and designer of hash functions and encryption schemes. Examples of the use of SAT solvers in cryptanalysis include tools aimed at the search for cryptographic keys in 1999 [36], logical cryptanalysis as a SAT problem in 2000 [37], encoding modular root finding as a SAT problem in 2003 [21] and logical analysis of hash functions in 2005 [26]. Most of these approaches used a direct encoding of the said problems into a satisfiability problem and used SAT solvers as a blackbox.

In this paper, we propose a set of techniques and an implementation, we call MapleCrypt, that dramatically improve upon the state-of-the-art in solving the *cryptographic hash function inversion problem*. The problem of inverting hash function is of great importance to cryptographers and security researchers, given that many security protocols and primitives rely on these functions being *hard-to-invert*. Informally, the problem is "given a specific hash value (or target) H find an input to the hash function that hashes to H". We focus on the inversion problem, as opposed to the more well-studied collision problem. The value of this research is not only the fact that cryptanalysis is an increasingly important area of application for SAT solvers, but also that instances generated from cryptographic applications tend to be significantly harder

for solvers than typical industrial instances, and hence are a very good benchmark for solver research.

Summary of Contributions. We focus on the SHA-1 cryptographic hash function in this paper, and make the following contributions:

1. We present a counter-example guided abstraction-refinement [13] (CEGAR) based technique, wherein certain steps of the hash function under analysis are abstracted away and replaced by the identity function. The inversion problem for the resultant abstracted hash function is often much easier for solvers, and we find that we do not have to do too many steps of refinement. An insight from this experiment is that certain steps of the SHA-1 hash function do not have sufficient levels of diffusion, a key property of hash functions that makes them difficult to invert.

2. In addition to the above-mentioned CEGAR technique, we present a multi-armed bandit [50] based adaptive restart policy. The idea is that a reinforcement learning technique is used to select among a set of restart policies in an online fashion during the run of the solver. This method is of general value, beyond cryptanalysis. The result of combining these two techniques is a tool, we call MapleCrypt, that is around two times faster on the hash function inversion problem than most state-of-the-art SAT solvers. More importantly, with a time limit of 72 hours, MapleCrypt can invert a 23-step reduced version of SHA-1 consistently, whereas other tools we compared against can do so only occasionally.

3. We perform extensive evaluation of MapleCrypt on the SHA-1 inversion problem, and compare against CryptoMiniSat, Lingeling, MiniSAT, Minisat-BLBD. In particular, MapleCrypt is competitive against the best tools out there for inverting SHA-1.

2 Background on Cryptographic Hash Functions

In this section we provide a brief background on cryptographic hash functions, esp. SHA-1. We refer the reader to [22] for a more detailed overview of hash functions. A hash function maps an arbitrary length input string to a fixed length output string (e.g., 160 bits in the case of SHA-1). There are three main properties that are desired for a cryptographic hash function [30]. Informally, they are:

- *Preimage Resistance*: Given a hash value H, it should be computationally infeasible to find a message M, where $H = hash(M)$.
- *Second Preimage Resistance*: Given a message M_1, it should be computationally infeasible to find another message M_2, where $hash(M_1) = hash(M_2)$ and $M_1 \neq M_2$.
- *Collision Resistance*: It should be computationally infeasible to find a pair of messages M_1 and M_2, where $hash(M_1) = hash(M_2)$ and $M_1 \neq M_2$. (There is a subtle difference between second pre-image resistance and collision resistance, in that the message M_1 is not fixed in the case of collision resistance).

Preimage resistance implies that the hash function should be hard to invert. The terms preimage attack and inversion attack are used interchangeably. Standard cryptographic hash functions at their core have a compression function, which can essentially be seen as repeated application of a step function on its input bits for a fixed number of steps. The compression function takes as input a fixed length input and outputs a fixed length (with smaller length) output. For making a collision resistant compression function, one method is to use a block cipher and apply the Davis-Meyer method. Feistel ladder which is widely used in hash functions like MD5 and SHA-1 (and also block ciphers like DES), is an implementation of this method, where the key is a message word. If the key is known, each step is easily reversible. For making

a hash function able to accept arbitrary long messages as input, one can use Merkle-Damgard structure [40], where it is shown that if one block is collision resistant, the whole structure would be collision resistant.

2.1 SHA-1

SHA-1 (Secure Hash Algorithm), was designed by NSA, and adopted as a standard in 1995 [22], and is still widely used in many applications. SHA-1 consists of iterative application of a so-called compression function which takes a 160-bit chaining value and transforms it into the next chaining value using a 512-bit message block. The current chaining value would be added to the output of compression function to make the next chaining value: $CV_{i+1} = Comp(CV_i, M_i) + CV_i$, and the CV_0 is set to a fixed initialization vector. Before starting the process, the message is padded with a single bit '1' followed by a set of '0's and original message length as a 64-bit value. The length of padded version is a multiple of 512. The message is broken down to blocks of 512 bits. Each compression function breaks down the input message block into sixteen 32-bit words. Then it will go through a message expansion phase, which extends sixteen words to eighty words, using the following formulation (consider that W_i for $0 \leq i \leq 15$ refers to input message words):

$$W_i = (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) \lll 1 \quad (16 \leq i \leq 79) \tag{1}$$

where '$\lll 1$' is left rotation by one position.

There are five 32-bit words (namely, a, b, c, d, e) as the intermediate variables. In each step a function F_t is applied to three of these words, and it changes every 20 steps:

$$F_t(b,c,d) = \begin{cases} Ch(b,c,d) = (b \wedge c) \oplus (\neg b \wedge d) & 0 \leq t \leq 19 \\ Parity(b,c,d) = b \oplus c \oplus d & 20 \leq t \leq 39 \\ Maj(b,c,d) = (b \wedge c) \oplus (b \wedge d) \oplus (c \wedge d) & 40 \leq t \leq 59 \\ Parity(b,c,d) = b \oplus c \oplus d & 60 \leq t \leq 79 \end{cases} \tag{2}$$

The step process would be like:

$$(a_{t+1}, b_{t+1}, c_{t+1}, d_{t+1}, e_{t+1}) \leftarrow (F_t(b_t, c_t, d_t) \boxplus e_t \boxplus (a_t \lll 5) \boxplus W_t \boxplus K_t, a_t, b_t \lll 30, c_t, d_t) \tag{3}$$

where $\lll$ is left rotation, $\boxplus$ is modulo-2^{32} addition and K_t is the round constant. This is repeated for next 512-bit message block in the Merkel-Damagard chain.

Step-Reduced Version: Usually inverting or finding collision for full version of a hash function is very hard. Thus, cryptanalysts work on a relaxed version of those functions like step-reduced versions, which means the function under attack is the same, except that the number of steps is reduced.

3 Architecture of MapleCrypt

We seek to perform a preimage or inversion attack on a step-reduced version of SHA-1, with one block input (512 bits). Although the current work is focused on SHA-1, our approach is applicable to other iterative hash functions. The two main contributions in our design are adaptive restart and a CEGAR-based approach. The adaptive restart is not directly dealing

with the structure of the function and therefore could be used in solving other SAT instances. The CEGAR approach is abstracting and refining step functions. Thus it could be mounted on the other hash functions that have a repeated use of a step function (e.g. MD4, MD5, SHA-2).

3.1 SAT Encoding

The encoding we use in this paper is based on the given in [44]. Most of the operations are encoded using Tseitin transformation, but some operations are described using high level relations. There is a 5-operand addition in each step of SHA-1 (refer to equation (3)). The main contribution of the encoding in [44] is the encoding of this multi-operand addition (instead of encoding of multiple two-operand additions). Current SHA-1 instances in SAT competition are generated using this tool [45]. We have made minor modifications, namely to the encoding of round-dependent logical function (F_t in equation (2)), replacing XOR operations with inclusive-OR to simplify the corresponding clauses.

3.2 CEGAR Loop Design

The SHA-1 SAT instances of up to 20 steps are very easy to solve (less than a second using most modern solvers), but the level of difficulty rapidly rises from 20 steps to 23 steps (needs 2 to 3 days to solve). To the best of our knowledge, preimage for more than 23 steps cannot be constructed in a reasonable amount of time even with the latest techniques and hardware [44, 32, 31].

For the instances of more than 20 steps (e.g. 22 steps), we abstract away initial step functions and keep the last 20 steps intact (abstracting first 2 steps), but we do not abstract away the message words and the message expansion relations. We solve the simplified instance, and find a solution for the message words. However, the resultant solution may be spurious, i.e., may not actually hold for the specific hash target. In order to verify the solution, we run the hash function in the forward direction and check the result with the target, and record values of intermediate variables throughout the hashing. If the computed hash does not match the target (i.e., the solution produced by the solver is spurious), we refine back those parts of abstracted steps of the hash function that are unsatisfiable under the spurious solution. Finally we also add a subset (all except last 8 steps) of intermediate values (computed during the forward run of the hash function on spurious solutions) as blocking clauses.

The intuition behind this procedure is that, first of all, 20 steps are very easy to solve, and it is the highest number of steps that we are better off solving directly, rather than using an abstraction. Secondly, the first few intermediate variables have the most degree of freedom when searching for a preimage or collision. Lastly, blocking a subset of intermediate values, although might block some legitimate solutions, but also blocks many spurious solutions. We can divide our main procedure into two main functions, listed in Algorithm 1.

In the listing of algorithm 1, *interValues* refers to the collection of intermediate variables across working steps that will be negated and added as a conflict clause to a CCDB (conflict clause database). The *Refine* function evaluates the clauses of the original formula with the found solution and checks which variables are in the UNSAT clauses and add them back to the current abstracted instance.

3.3 Multi-Armed Bandit Restart

Many restart policies have been proposed in the SAT literature [5, 4, 34, 6], in particular we focus on the uniform, linear, luby, and geometric restart policies [10]. For a given preimage

Algorithm 1 Finding Preimage using a CEGAR loop

Require: W: 512-bit found preimage, H: Hash target, *nsteps*: number of hash steps
Ensure: *true* if W is a valid preimage, *false* otherwise
1: **function** CHECK($W, H, nsteps$)
2: (H', interValues) ← SHA-1(W, nsteps)
3: **if** $H = H'$ **then**
4: **return** true
5: **else**
6: CCDB.add(interValues)
7: **return** false
8:
Require: H: Hash target, *nsteps*: number of hash steps
Ensure: W: 512-bit preimage of H
9: **function** FINDPREIMAGE($H, nsteps$)
10: InstanceSteps ← AbsInstGen(nsteps) ▷ Abstracted set of step functions
11: InstanceW ← MsgInstGen(nsteps) ▷ All of input and expanded message words
12: **while** true **do**
13: W[0..15] ← SATSolver(InstanceSteps, InstanceW, CCDB)
14: **if** Check(W, H, nsteps) = true **then**
15: **return** W
16: InstanceSteps ← Refine(InstanceSteps, W, nsteps)

attack instance, we can not know a priori which of the 4 restart policy will perform the best. To compensate for this, we use multi-armed bandits (MAB) [50], a special case of reinforcement learning, to switch between the 4 policies dynamically during the run of the solver. We chose to use discounted UCB algorithm [24] from MAB literature, as it accounts for the nonstationary environment of the CDCL solver, in particular changes in the learnt clause database over time. Discounted UCB has 4 actions to choose from corresponding to the uniform, linear, luby, and geometric restart policies. Once the action is selected, the solver will proceed to perform the CDCL backtracking search until the chosen restart policy decides to restart. The algorithm computes the average LBD (Literals Block Distance [3]) of the learnt clauses generated since the action was selected, and the reciprocal of the average is the reward given to the selected action. Intuitively, a restart policy which generates small LBDs will receive larger rewards and UCB will increase the probability of selecting that restart policy in the future. Over time, this will bias UCB towards restart policies that generate small LBDs.

4 Experimental Results

4.1 Experimental Setup

Our baseline benchmark consists of instances of preimage of step-reduced SHA-1, from 21 to 23 steps. Instances for less than 21 steps were trivial for every solver we tried. For each step we generated 25 random targets and encoded them as fixed value for the hash output. All jobs were run on AMD opteron CPUs at 2.2 GHz and 8GB RAM. The timeout for solving a single instance was 72 hours, with 4GB of memory allocated for each process. We used 5 SAT solvers, CryptoMiniSat-4.5.3 [47], Minisat_BLBD [12], Lingeling-ayv [8], Minisat-2.2 [18] and MapleSAT [33]. We also tried 4 SMT solvers to take advantage of their BitVector theory solvers.

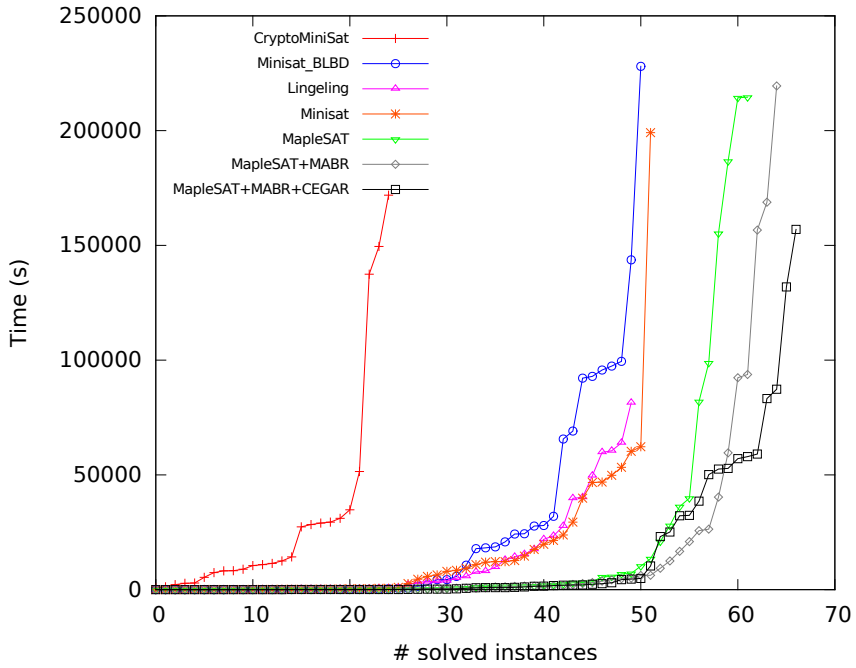

Figure 1: The performance of various SAT solvers against MapleSAT with adaptive MAB-restart and CEGAR.

We used, STP-2.0/CryptoMinisat4, Boolector-2.0.7/Lingeling, CVC4-1.5, Z3-4.4. However, all of these SMT solvers performed surprisingly poorly. Hence, we did not include them in our comparisons.

4.2 CEGAR and MABR

State-of-the-art results for automated and practical preimage of SHA-1 that construct a result rather than presenting an upper bound for attack complexity, propose a SAT encoding of the preimage attack and solve it using modern SAT solvers. Therefore we picked the best existing encoding method for SHA-1 [44] and applied our solving techniques on them. We are comparing our runtimes with other SAT solvers, given the same instances. Figure 1 shows the cactus plot of solving times where each data point shows how many instances could be solved in the corresponding time. Curves more toward bottom are faster and more toward right are solving more instances. It can be seen that MapleSAT with MAB restart dominates in terms of runtime and number of instances solved, and after employing the CEGAR technique, we are able to solve faster and more instances. MapleCrypt is the CEGAR architecture that uses MapleSAT+MABR as backend solver.

4.3 Partial Preimage

This is the kind of attack where the attacker knows some bits of the input message and wants to find out the rest. Our experiments on SHA-1 show that knowing parts of the message does not necessarily make the problem easier, as it might force the solver to find a specific input that

matches those bits and reduces the possibilities. Our results mostly confirm the observations on hardness of partial preimage of SHA-1 in [45]. We could invert up to 27 steps of SHA-1 when having 40 bits of the input message unknown, which matches the best results known in this setting for SHA-1 [42].

5 Related Work

We review the related work on inversion attacks, and touch upon collision attacks as appropriate. Note that for inversion attacks, every additional round of a hash function inverted is considered significant improvement over previous work.

SAT-based Constructive Methods. Since 2005, several hash functions have been shown to be prone to collision attacks [52, 53]. In 2006, Mironov et.al [41] automated parts of collision attack of Wang et al. [52] using SAT solvers. Not many of the collision attack methods use SAT solvers, as the collision finding problem is studied very well and most cryptanalysts use direct implementation of mathematical analyses (e.g., differential cryptanalysis) for this problem.

Given that the focus of our paper is on inversion attacks, we provide a thorough overview of the related work for the same. In 2007, De et al. [14] used a SAT solver for an inversion attack on MD4 and enhanced number of steps inverted, up to 2 rounds and 7 steps, by encoding Dobbertin's attack model [17] into SAT constraints. In 2008, Srebrny et al. [48] formulated inversion of SHA-1 as a SAT instance and could solve for restricted message size up to 22 steps. Several later works could solve up to 23 rounds of SHA-1 [44, 32, 31]. Lafitte et al. [29] presented a generic way to encode basic cryptographic operations which was an improvement over operator overloading model in [26] and used it in a preimage test on MD4 and finding weak keys in IDEA and MESH ciphers, although other than finding weak keys, preimage results were not better than the best previously published attack. In 2013, Morawiecki et al. [42] used the idea of minimization of SAT instances generated via analysis of cryptographic primitives. They applied their tool CryptLogVer on some hash functions like SHA-1 and Keccak to analyze their preimage resistance. Although they did not increase the number of inverted steps, they showed improvement in solving time. Nossum [45], also presents an encoding for preimage attack of SHA-1 which targets the 5-operand addition operation that is performed in each step of SHA-1 and the generated instances have fewer variables than the work of Srebrny et al. [48] and Morwiecki et al. [42], and in general are easier to solve by modern SAT solvers. It is used to generate SHA-1 instances of SAT competition [45]. The work presented in this paper is using Nossum's instance generator with small tweaks. All of the mentioned techniques use SAT solvers as a black box tool. By contrast our design leverages CEGAR and modifies the restart policies of SAT solvers. Additionally, our method is faster for finding preimages than previous work.

Non SAT Solver based Constructive Attacks. These methods are almost exclusively aimed at collision attacks. In 2006, De Cannière et al. [15] built a non SAT solver based tool for SHA-1 collision attack leveraging the breakthrough of Wang et al. [53]. In 2011, Mendel et al. [38] extended it for SHA-256 which was further improved in their work in 2013 [39]. Eichlseder et al. [19] improved upon the branching heuristics of this tool and applied it to SHA-512. Recently Stevens et al. [49] presented a parallelized search implementation and found free-start collision on full SHA-1.

Non-constructive Theoretical Bounds. Here we review known theoretical bounds on preimage attack on various hash functions. One of the first preimage results on SHA-1 was achieved by using techniques like reversing the inversion problem and mathematical structures like P^3 graphs [16], which could invert 34 and 44 steps with complexity of 2^{80} and 2^{157} respec-

tively. Aoki and Sasaki [1] used meet-in-the-middle to attack SHA-1 (and also MD4 and MD5) and improved the number of steps to 48 with the solving complexity of $2^{159.3}$. Knellwolf et al. [28] improved it in 2012 by providing a differential formulation of MITM model and raised the bar up to 57 steps. Espiatu et al. [20] extended this work further to higher order differentials for preimage attack on SHA-1 and BLAKE2 and went up to 62 steps for SHA-1. Mathematical structure like Biclique [27], allowed extending coverage of MITM over larger number of steps.

Adaptive Restarts. Armin Biere proposed monitoring variable assignment flips in PicoSAT, and delayed restarts when the weighted average of flips is below a predetermined threshold [5]. Audemard and Simon proposed monitoring the LBD of learnt clauses, and a restart is triggered if the short term LBDs exceeds the long term LBDs by a constant factor [4]. Haim and Walsh used machine learning to train a classifier to select from a portfolio of restart policies [25]. Gagliolo and Schmidhuber used bandits to select between luby and uniform restart heuristic [23].

6 Conclusion and Future Work

We presented a tool called MapleCrypt for preimage attack on SHA-1 hash functions, which uses CEGAR and adaptive restart techniques. Our tool is faster than other automated search tools in the literature for the constructive preimage attack. Our results show that SAT solvers and SAT-based techniques are a promising approach for handling the laborious parts of cryptanalysis, and identifying weaknesses in hash function designs. This design can be extended to work on other hash functions like SHA-2 family.

References

[1] Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-step MD5 and more. In *Selected Areas in Cryptography*, pages 103–119. Springer Berlin Heidelberg, 2009.

[2] Gilles Audemard and Laurent Simon. GLUCOSE: A Solver that Predicts Learnt Clauses Quality. *SAT Competition*, pages 7–8, 2009.

[3] Gilles Audemard and Laurent Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *IJCAI*, volume 9, pages 399–404, 2009.

[4] Gilles Audemard and Laurent Simon. Refining Restarts Strategies for SAT and UNSAT. In *Principles and Practice of Constraint Programming*, pages 118–126. Springer, 2012.

[5] Armin Biere. Adaptive Restart Strategies for Conflict Driven SAT Solvers. In *Theory and Applications of Satisfiability Testing–SAT 2008*, pages 28–33. Springer, 2008.

[6] Armin Biere. PicoSAT Essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008.

[7] Armin Biere. Lingeling, Plingeling, Picosat and Precosat at SAT Race 2010. *FMV Report Series Technical Report*, 10(1), 2010.

[8] Armin Biere. Lingeling ayv. `http://fmv.jku.at/lingeling/`, 2015.

[9] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded Model Checking. *Advances in computers*, 58:117–148, 2003.

[10] Armin Biere and Andreas Fröhlich. Evaluating CDCL Restart Schemes. In *Pragmatics of SAT*, 2015.

[11] Cristian Cadar, Vijay Ganesh, Peter M Pawlowski, David L Dill, and Dawson R Engler. EXE: Automatically Generating Inputs of Death. *ACM Transactions on Information and System Security (TISSEC)*, 12(2):10, 2008.

[12] Jingchao Chen. A bit-encoding phase selection strategy for satisfiability solvers. In *Theory and Applications of Models of Computation*, pages 158–167. Springer, 2014.

[13] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided Abstraction Refinement. In *Computer aided verification*, pages 154–169. Springer, 2000.

[14] Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion Attacks on Secure Hash Functions Using SAT Solvers. *Theory and Applications of Satisfiability Testing–SAT 2007*, pages 377–382, 2007.

[15] Christophe De Canniere and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In *Advances in Cryptology–ASIACRYPT 2006*, pages 1–20. Springer, 2006.

[16] Christophe De Canniere and Christian Rechberger. Preimages for Reduced SHA-0 and SHA-1. In *Advances in Cryptology–CRYPTO 2008*, pages 179–202. Springer, 2008.

[17] Hans Dobbertin. Cryptanalysis of MD4. In *Fast Software Encryption*, pages 53–69. Springer, 1996.

[18] Niklas Eén and Niklas Sörensson. Minisat 2.2. http://minisat.se/.

[19] Maria Eichlseder, Florian Mendel, and Martin Schläffer. Branching Heuristics in Differential Collision Search with Applications to SHA-512. *IACR Cryptology ePrint Archive*, 2014:302, 2014.

[20] Thomas Espitau, Pierre-Alain Fouque, and Pierre Karpman. Higher-Order Differential Meet-in-the-Middle Preimage Attacks on SHA-1 and BLAKE. In *Advances in Cryptology–CRYPTO 2015*, pages 683–701. Springer, 2015.

[21] 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.

[22] PUB FIPS. 180-4. *Federal Information Processing Standards Publication, Secure Hash*, 2011.

[23] Matteo Gagliolo and Jürgen Schmidhuber. Learning Restart Strategies. In *IJCAI*, pages 792–797, 2007.

[24] Aurélien Garivier and Eric Moulines. *Algorithmic Learning Theory: 22nd International Conference, ALT 2011, Espoo, Finland, October 5-7, 2011. Proceedings*, chapter On Upper-Confidence Bound Policies for Switching Bandit Problems, pages 174–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[25] Shai Haim and Toby Walsh. Restart Strategy Selection Using Machine Learning Techniques. In *Theory and Applications of Satisfiability Testing-SAT 2009*, pages 312–325. Springer, 2009.

[26] Dejan Jovanović and Predrag Janičić. *Logical Analysis of Hash Functions*. Springer, 2005.

[27] Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In *Fast Software Encryption*, pages 244–263. Springer, 2012.

[28] Simon Knellwolf and Dmitry Khovratovich. New Preimage Attacks Against Reduced SHA-1. In *Advances in Cryptology–Crypto 2012*, pages 367–383. Springer, 2012.

[29] Frédéric Lafitte, Jorge Nakahara Jr, and Dirk Van Heule. Applications of SAT Solvers in Cryptanalysis: Finding Weak Keys and Preimages. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:1–25, 2014.

[30] Xucjia Lai and James L Massey. Hash Functions Based on Block Ciphers. In *Advances in CryptologyEUROCRYPT92*, pages 55–70. Springer, 1993.

[31] Florian Legendre, Gilles Dequen, and Michaël Krajecki. Encoding Hash Functions as a SAT Problem. In *Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on*, volume 1, pages 916–921. IEEE, 2012.

[32] Florian Legendre, Gilles Dequen, and Michaël Krajecki. Logical Reasoning to Detect Weaknesses About SHA-1 and MD4/5. *IACR Cryptology ePrint Archive*, 2014:239, 2014.

[33] Jia Hui Liang, Vijay Ganesh, Poupart Pascal, and Krzysztof Czarnecki. Learning Rate Based Branching Heuristic for SAT Solvers. *Submitted to Theory and Applications of Satisfiability Testing-SAT 2016*, 2016.

[34] Michael Luby, Alistair Sinclair, and David Zuckerman. Optimal Speedup of Las Vegas Algorithms. In *Theory and Computing Systems, 1993., Proceedings of the 2nd Israel Symposium on the*, pages 128–133. IEEE, 1993.

[35] João P Marques-Silva and Karem A Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *Computers, IEEE Transactions on*, 48(5):506–521, 1999.

[36] Fabio Massacci. Using Walk-SAT and Rel-SAT for Cryptographic Key Search. In *IJCAI*, volume 1999, pages 290–295, 1999.

[37] Fabio Massacci and Laura Marraro. Logical Cryptanalysis as a SAT Problem. *Journal of Automated Reasoning*, 24(1-2):165–203, 2000.

[38] Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding SHA-2 Characteristics: Searching Through a Minefield of Contradictions. *Advances in Cryptology–ASIACRYPT 2011*, pages 288–307, 2011.

[39] Florian Mendel, Tomislav Nad, and Martin Schläffer. Improving Local Collisions: New Attacks on Reduced SHA-256. In *EUROCRYPT*, volume 7881, pages 262–278, 2013.

[40] Ralph C Merkle. One Way Hash Functions and DES. In *Advances in CryptologyCRYPTO89 Proceedings*, pages 428–446. Springer, 1989.

[41] Ilya Mironov and Lintao Zhang. Applications of SAT Solvers to Cryptanalysis of Hash Functions. *Theory and Applications of Satisfiability Testing-SAT 2006*, pages 102–115, 2006.

[42] Paweł Morawiecki and Marian Srebrny. A SAT-based Preimage Analysis of Reduced KECCAK Hash Functions. *Information Processing Letters*, 113(10):392–397, 2013.

[43] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th annual Design Automation Conference*, pages 530–535. ACM, 2001.

[44] Vegard Nossum. SAT-based Preimage Attacks on SHA-1. 2012.

[45] Vegard Nossum. Instance Generator for Encoding Preimage, Second-Preimage, and Collision Attacks on SHA-1. *Proceedings of the SAT competition*, pages 119–120, 2013.

[46] Jussi Rintanen. Planning and SAT. *Handbook of Satisfiability*, 185:483–504, 2009.

[47] Mate Soos. CryptoMiniSat 4.5.3. http://www.msoos.org/cryptominisat4/, 2015.

[48] Marian Srebrny, Mateusz Srebrny, and Lidia Stepien. SAT as a Programming Environment for Linear Algebra and Cryptanalysis. In *ISAIM*, 2008.

[49] Marc Stevens, Pierre Karpman, and Thomas Peyrin. Freestart Collision for Full SHA-1. *Cryptology ePrint Archive*, (2015/967):1–21, 2015.

[50] Richard S Sutton and Andrew G Barto. *Introduction to Reinforcement Learning*, volume 135. MIT Press Cambridge, 1998.

[51] Aaron Tomb. Applying Satisfiability to the Analysis of Cryptography. https://github.com/GaloisInc/sat2015-crypto/blob/master/slides/talk.pdf, 2015.

[52] Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. *Advances in Cryptology–EUROCRYPT 2005*, pages 19–35, 2005.

[53] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology–CRYPTO 2005*, pages 1–16. Springer, 2005.