Branching Heuristics in Differential Collision Search with Applications to SHA-512*

Maria Eichlseder, Florian Mendel, and Martin Schläffer

IAIK, Graz University of Technology, Graz, Austria
maria.eichlseder@iaik.tugraz.at

Abstract. In this work, we present practical semi-free-start collisions for SHA-512 on up to 38 (out of 80) steps with complexity $2^{40.5}$. The best previously published result was on 24 steps. The attack is based on extending local collisions as proposed by Mendel et al. in their Eurocrypt 2013 attack on SHA-256. However, for SHA-512, the search space is too large for direct application of these techniques. We achieve our result by improving the branching heuristic of the guess-and-determine approach to find differential characteristics and conforming message pairs. Experiments show that for smaller problems like 27 steps of SHA-512, the heuristic can also speed up the collision search by a factor of 2^{20}.

Keywords: hash functions, cryptanalysis, SHA-512, collision attack, guess-and-determine attack, branching heuristic

1 Introduction

Since 2005, many collision attacks have been shown for commonly used and standardized hash functions. In particular, the collision attacks of Wang et al. [37,38] on MD5 and SHA-1 have convinced many cryptographers that these widely deployed hash functions can no longer be considered secure. As a consequence, NIST has proposed the transition from SHA-1 to the SHA-2 family. Many companies and organization follow this advice and have already migrated to SHA-2. Even more might do so, since Keccak [29] has not been standardized as SHA-3 yet and SHA-2 is faster on several platforms. In particular, SHA-512 is much faster than both SHA-256 and Keccak on most 64-bit platforms [2]. For this reason, it has been suggested to use a truncated version of SHA-512 even for 256-bit hash values [34]. NIST also defines this variant, called SHA-512/256, in FIPS 180-4 [28].

Nevertheless, not many cryptanalytic results on SHA-512 have been published in the last few years. The security of SHA-512 against preimage attacks was first studied by Aoki et al. in [1]. They presented a preimage attack on 46 out of 80 steps. This was later extended to 50 steps by Khovratovich et al. in [15]. Recently, Li et al. showed that particular preimage attacks can also be used to construct a free-start collision attack for up to 57 steps of SHA-512 in [20]. However, all attacks are only slightly faster than the respective generic attack complexities.

* © IACR 2014. This article is the final version submitted by the authors to the IACR and to Springer-Verlag on 2014-04-30, which appears in the proceedings of FSE 2014.

The currently best known practical collision attack on both the SHA-512 hash and compression function is for 24 steps. It has been published independently by Indesteege et al. [13] and by Sanadhya and Sarkar [32]. Both attacks are trivial extensions of the attack strategy of Nikolić and Biryukov [30] which applies to both SHA-256 and SHA-512. Recently, Mendel et al. [23, 25] demonstrated how to extend these attacks to get collisions for the SHA-256 compression function on up to 38 steps with practical complexity.

The attacks by Mendel et al. use a guess-and-determine based automatic search tool to find differential characteristics and conforming message pairs for reduced SHA-256. Since the first publication by De Cannière and Rechberger on SHA-1 in [5], such tools have been constantly improved [17, 18, 23, 25]. Nevertheless, the increased search space of SHA-512 (due to larger word sizes) prevented successful attacks without the application of new ideas.

To handle the larger search space of SHA-512, we propose a new branching heuristic for the guess-and-determine strategy used in these attacks. Our approach is inspired from related ideas in SAT solvers [12, 19]. The heuristic performs a randomized look-ahead selection of candidates which should be guessed first. Using this approach, we can detect contradictions earlier and reduce the search space faster. More specifically, we are able to speed up the search on SHA-512 by a factor of about 2^{20} (for 27 steps), which allows us to construct practical collisions for 38 steps with a complexity of $2^{40.5}$.

The remainder of this paper is structured as follows. We first give a high-level overview of our attack strategy and related work in Section 2. In Section 3, we discuss branching heuristics used in SAT solvers and propose our new look-ahead branching heuristic for differential cryptanalysis tools. In Section 4, we demonstrate the application of the heuristic to SHA-512 and present a practical semi-free-start collision for 38 steps. Finally, we conclude in Section 5.

2 Motivation

In this section, we give a brief overview on the differential cryptanalysis of hash functions and how the guess-and-determine approach is used to search for differential characteristics. Furthermore, we provide a high-level view on optimization options to improve this search.

2.1 The Search for Differential Characteristics

A differential attack consists of two main parts: constructing a differential characteristic and finding a confirming message pair. Since the attacks by Wang et al. [36–38], these parts are further divided to improve the overall attack complexity as follows:

- **Find a differential characteristic**
  1. Construct the high-probability part of a characteristic
  2. Determine the low-probability part of a characteristic

- **Find a conforming message pair**
  3. Use message modification in low-probability part
  4. Perform random trials in high-probability part

We provide significant improvements in finding dense low-probability differential characteristics. To motivate our work, we first provide an overview of previously published methods and show how we improve upon these methods using improvements in the guess-and-determine approach.

Constructing the differential characteristic for the low-probability part is one of the most difficult tasks in a differential attack. The main reason is that such low-probability characteristics are usually very dense and have many (hidden) relations which need to be taken into account. Wang et al. found the dense low-probability characteristics for the attacks on MD4, MD5, RIPEMD, SHA-0 and SHA-1 mostly by hand [36–38]. However, for more complex hash functions, such an approach is infeasible. Therefore, (semi-)automatic approaches have been published soon afterwards [5, 33]. These approaches have then been refined in a number of publications. Recently, more sophisticated approaches have been proposed that enable attacks on more complex hash functions such as SHA-256 [23, 25] among many others [16, 18, 22, 24]. All these approaches (including the search by hand) follow the guess-and-determine strategy.

2.2 The Guess-and-Determine Approach

The basic idea of the search algorithm is to pick and guess previously unrestricted bits. After each guess, the information gained from these restrictions is propagated to other bits. If an inconsistency occurs, the algorithm backtracks to an earlier state of the search and tries to correct it. Similar to [23], we denote these three parts of the search by decision (guessing), deduction (propagation), and backtracking (correction). Then, the search algorithm proceeds as in Algorithm 1 given below.

Algorithm 1 Guess-and-Determine Search Algorithm

```
Let $U$ be a set of undetermined bits
while $U$ contains undetermined bits do
  Decision (Guessing)
    1. pick an undetermined bit (randomly or heuristically)
    2. impose new constraints on this bit
  Deduction (Propagation)
    3. propagate the new information to other variables and equations
    4. if an inconsistency is detected, start backtracking,
       else continue with step 1
  Backtracking (Correction)
    5. try a different choice for the decision bit and continue with step 3.
    6. if all choices result in inconsistencies,
         undo guesses until this critical bit can be resolved
```

This procedure can also be visualized by a search tree, which is traversed by depth first search. The branching strategy decides on which variable to split the tree next and thus defines the tree's shape. Typically, the complete tree is much too large for complete traversal, so it is crucial that more promising branches are visited first. In addition, the backtracking algorithm can skip parts of the tree in favor of exploring more distant parts. This makes the search incomplete, but in practice greatly improves the performance.

The challenge in finding a long differential characteristic lies in the fine-tuning of the search algorithm. There are many possible variations, and details can determine whether the search succeeds or fails.

2.3 Improving the Guess-and-Determine Approach

Basically, a guess-and-determine is just a repetition of two steps: first, guess the value of some unknowns and second, determine the value of as many unknowns as possible. However, in practice more details need to be considered to mount successful guess-and-determine attacks on complex hash functions. The most important points to consider are given as follows:

1. **Problem Description:** The complexity of a guess-and-determine attack can be significantly improved if we first optimize the problem description. For example, first constructing a characteristic and then searching for a message pair is already such an optimization. Additionally, the choice of intermediate variables and a good starting point are crucial for a guess-and-determine attack to succeed.
2. **Guessing Strategy:** Instead of randomly guessing variables, using high-level information can lead to much better guesses. For example, by preferring bits (or even words) with no differences, characteristics tend to get sparser, have a higher probability, and conforming message pairs are more likely to exist.
3. **Branching Rules:** In every iteration, the guess-and-determine algorithm needs to decide which branch of the search tree to follow. Using a good branching heuristic, contradictions can be found faster and the search space can be reduced more quickly.
4. **Propagation:** Every time a variable is guessed, we need to check whether the guess is invalid, or new information on other variables can be determined. There is a trade-off between the effort we spend in this step and simply guessing more bits. Different propagation methods for ARX-based hash functions are covered in detail in [7, 17, 18, 23].
5. **Backtracking:** To recover from bad search spaces which do not contain many solutions anymore, we need to backtrack. Two extreme options are performing a complete restart or examining the complete search space. A successful backtracking strategy for SHA-2 has been published in [23].

The first two points are very specific to a given problem and cannot be solved in general. In our attacks on SHA-512, a good starting point is constructed using improved local collisions, similar as in the attack on SHA-256 in [25]. The last two points have already been covered in a number of publications. Additional efforts in these points did not improve the guess-and-determine attack on SHA-512. This leaves the branching rules which have not been optimized yet. In the following, we show that a good branching heuristic can significantly improve the efficiency of a guess-and-determine attack.

3 Branching Heuristics

Branching rules are one of the essential ingredients for guess-and-determine attacks. They define how the search algorithm selects the next variable to guess, and which guess values to try first for this variable. The branching rule aims to keep the search runtime as short as possible. Depending on whether the current partial assignment is correct (satisfiable) or contradictory, this means either that a satisfying solution is found as soon as possible, or that the contradiction is detected quickly. In the latter case, this corresponds to identifying a conflicting subset of unassigned variables and branching on these first in order to prune the search tree. The search trees traversed by different branching rules can vary drastically in size, from constant (for unsatisfiable problems) or linear (for satisfiable problems) to exponential in the number of variables [31].

This section first discusses existing branching rules used in general-purpose SAT solvers and for the cryptanalysis of hash functions. Afterwards, we introduce our randomized look-ahead heuristic.

3.1 Branching Heuristics in SAT Solvers

Most general-purpose SAT solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [4], a guess-and-determine approach for satisfiability problems given in conjunctive normal form (CNF). The problem of choosing optimal branching variables and corresponding assignments for DPLL algorithms has been proven to be both NP-hard and coNP-hard [21]. However, there is a variety of commonly implemented branching rules based on different heuristics to evaluate the urgency or relevance of potential branching variables. In addition, meta-rules to select different branching rules depending on the situation and search history have been proposed [10].

Commonly used SAT branching rules can be categorized according to their target heuristic (current properties, look-ahead or history analysis), their output (a single branching variable/literal or a preselection of candidate variables) and their randomness (deterministic or randomized). Popularly used heuristics include the following:

– **Uniformly random**. A random unassigned variable is picked with uniform distribution. This approach is computationally cheapest. Many modern SAT solvers apply this rule with a small probability and otherwise use a more informed choice. In differential cryptanalysis, this is the most typical rule.

- **Small clauses**. The earliest heuristics greedily favor variables that appear in many small clauses. The rationale for this choice is twofold. First, smaller clauses need to be fulfilled “more urgently” since there are fewer options left that avoid contradictions. Second, even if the guessed literal evaluates to false in binary clauses, unit propagation ensues and curtails the search tree. Example heuristics of this category include Böhm’s rule [3], MOM (maximum occurrences on clauses of minimum size) [8], and the Jeroslaw-Wang rules [14]. The latter, for example, assign weights $w(c)$ to clauses c that decrease exponentially with the clause length $|c|$. Each literal (OS-JW) or variable (TW-JW) scores according to the weight sum of all clauses it appears in, and the best literal or variable is picked for guessing.
  More recently, small clauses have been used as a preselection heuristic for more expensive look-ahead rules. In differential guess-and-determine attacks, two-bit conditions [23] play a related role. This preselection heuristic also favors variables with a higher number of closely coupled undetermined variables.
- **Literal count**. These heuristics ignore the clause size and simply count unresolved clauses linked to a variable. Examples include DLCS and DLIS as introduced by the GRASP solver [35]. It makes sense in CNF problems, where satisfying one literal resolves the complete clause. This does not apply, for example, for the xor-chains typically found in hash functions. Instead, this heuristic would create a large amount of (hidden) dependencies and reduce the remaining freedom without the positive effect of immediate propagation.
- **Conflict driven**. A more popular variation of literal counting is VSIDS, first implemented in Chaff [26] and later included in MiniSAT [6] and others. Here, the initial literal score of each variable decays over time via multiplication with a constant $\delta < 1$. However, scores are refreshed (bumped) with occurrences in newly learned clauses from the CDCL process. Effectively, the score keeps track of recent contradictions involving the variable. Critical variables with many recent contradictions are guessed first.
  The BerkMin solver extends this concept to bump not only variables from learned clauses, but from any clauses involved in the resolution process [9]. In differential attacks, the backtracking strategy [23] provides a similar behaviour.
- **Look-ahead**. Instead of judging current properties of the formula or the previous search history, look-ahead heuristics analyse the actual effects of branching in a candidate variable [12, 19]. For example, the Satz solver performs Unit Propagation Look-Ahead: both possible assigments for each free variable are tested for consequences of this decision and the caused unit propagations. If one of two assignments causes a contradiction, the other is fixed; if both are contradictory, backtracking is started; and if both seem valid, the variable v is assigned score

  $$\mathcal{M}(v) = w(\neg v) \cdot w(v) \cdot 1024 + w(\neg v) + w(v),$$

  where $w(\ell)$ is typically the number of new binary clauses caused by the propagation of literal $\ell \in \{v, \neg v\}$.

– **Locality**. To limit the candidates for expensive look-ahead calculations, the candidate variables can be limited to those occurring in recently changed (reduced) clauses, as implemented in the marchdl solver [11].

Not all of these rules are suitable for general Boolean satisfiability problems that are not given in CNF format, as already indicated in the list above. In particular, if the propagation and learning process differs from the standard SAT case, the above rules can be counterproductive. On the positive side, dedicated solvers for specific applications can apply domain-specific knowledge to guide the search process.

3.2 The Look-Ahead Branching Heuristic

The branching strategy is one of the most promising areas for optimization in differential cryptanalysis tools based on tree search. Ideally, the branching strategy quickly navigates towards a valid assignment of variables and avoids subtrees without solutions. For detecting invalid subtrees, the branching strategy relies on the propagation method to detect contradictions as soon as possible. However, the propagation procedure can not only be used to decide whether previous guesses were contradictory. In addition, we also want to apply it to guide the branching strategy. The goal of this interaction is to minimize the size of the search tree in order to find solutions faster.

The basic principles of our implementation of the look-ahead branching heuristic are given by the following two observations:

– **Productive propagation is good.** Guessing a variable where propagation of the value determines (many) other variables can have multiple advantages compared to variables with less propagation. The most immediate effect is that the remaining search space is reduced. If more variables are determined right now, they will not create unnecessary subtrees for guessing later. The overall tree size and thus the complexity of the remaining search is reduced.
– **Contradictions are even better.** Of course, the overall search aims to find non-contradictory assignments. Nevertheless, discovering contradictory value assignments in the current subtree is consistently helpful for the remaining search. If only one of two possible value assignments is contradictory, the variable certainly needs to be fixed to the other value. If both values are contradictory, we must already have made an error with a previous guess and need to backtrack immediately. In both cases, it is clearly better to address the conflicting bit sooner rather than later.

Note that the first criterion is not beyond controversy. In particular, limiting the search space at the same time reduces the remaining degrees of freedom. If one value assigned to a specific bit propagates better than the second possible value, then, intuitively speaking, the probability for a solution in the remaining search space for the first option is lower than for the second value.

3.3 Implementation of the Look-Ahead Branching Heuristic

In order to implement the criteria above in a practical branching heuristic, we use a look-ahead approach related to the Unit-Propagation Look-Ahead (UPLA) used in some SAT solvers. When the branching rule needs to select the next variable to guess, each candidate is in turn evaluated.

In more detail, for each candidate, a value is tentatively assigned and the propagation method is applied to determine the consequences of this assignment. If a contradiction occurs, this candidate is selected immediately. Otherwise, the number of propagated variables is calculated. If it is better than the previously favorite candidate, this variable becomes the new favorite.

There are two performance-related problems with this basic approach. First, performing the look-ahead propagation for all free variables is very costly. Second, the basic UPLA approach includes no randomization. However, we need randomization since a complete search of the tree is typically computationally infeasible in differential cryptanalysis. Instead, large tree parts are skipped and the search is restarted regularly. To avoid becoming lost in the same search branches over and over again, it is essential that the branching strategy is sufficiently randomized.

We address both problems at once by selecting only a random subset of variables for closer evaluation. Our branching heuristic is summarized in Algorithm 2.

Algorithm 2 Look-ahead branching heuristic for differential cryptanalysis

```
Let $U$ be a set of undetermined bits and $s_{\max}$ the limit of look-ahead candidates.
repeat
  Guessing
    1. pick a bit $v \in U$ randomly and increment $s$
    2. impose new constraints on this bit $v$
  Propagation
    3. propagate the new information to other variables and equations
    4. if an inconsistency is detected, return $v$ as the decision bit
       else count the number $m$ of additional variables that were assigned due to
       this guess and save the pair $(v, m)$ in a list $L$.
  Update
    5. remove all variables that were assigned due to the guess $v$ from the set $U$
    6. undo all changes to restore the original assignment
until $U$ is empty or $s \geq s_{\max}$
return $v^*$ from $L$ with the highest score $m$ as the decision bit
```

The size of the randomly selected subset is an essential parameter for the success of the heuristic. To limit the look-ahead costs, we limit the maximum subset size by a constant number that is chosen in the beginning of the search procedure, depending on the specific problem instance. In order to also provide sufficient randomization, we additionally bound the size relative to the current number of unguessed variables.

Beside the subset size, the decision which individual variables to select for look-ahead plays a role. UPLA-based solvers use a pre-selection of interesting candidates, for example by locality criteria. In our case, the search performance can be greatly improved by only guessing bits of specific hash function words and favoring bits with more two-bit conditions or bits involved in recent conflicts. However, the selection must remain sufficiently randomized.

Additionally, we do not explicitly evaluate variables that were already determined by the propagation procedure of one of the previous candidates. We mark these as evaluated without calculating a separate look-ahead and without considering them as favorite candidates, since their score is at most as good as the bit that triggered their propagation (at least with respect to one of the assignment options).

4 Application to SHA-512

In this section, we discuss the application of our look-ahead branching heuristics to SHA-512. As a result, we are able to construct the first practical collision on the reduced SHA-512 compression function for 38 out of 80 steps. The best previously published result was on 24 steps.

4.1 Brief Description of SHA-512

SHA-512 is an iterated hash function that processes 1024-bit input message blocks and produces a 512-bit hash value. In the following, we briefly describe the hash function. It basically consists of two parts: the message expansion and the state update transformation. A detailed description of the hash function is given in [27].

Message Expansion. The message expansion of SHA-512 splits the 1024-bit message block into 16 64-bit words M_i and expands them into 80 expanded message words W_i as follows:

$$W_i = \begin{cases} M_i & 0 \le i < 16 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & 16 \le i < 80 \end{cases}$$

The functions $\sigma_0(x)$ and $\sigma_1(x)$ are given by

$$\begin{aligned} \sigma_0(x) &= (x \ggg 1) \oplus (x \ggg 8) \oplus (x \gg 7) \\ \sigma_1(x) &= (x \ggg 19) \oplus (x \ggg 61) \oplus (x \gg 6). \end{aligned}$$

State Update Transformation. The state update transformation starts from a (fixed) initial value IV of 8 64-bit words $A_{-4}, \ldots, A_{-1}, E_{-4}, \ldots, E_{-1}$ and updates them in 80 steps. In each step one expanded message word W_i is used to compute the two state variables E_i and A_i as follows:

$$\begin{aligned} E_i &= A_{i-4} + E_{i-4} + \Sigma_1(E_{i-1}) + \mathrm{IF}(E_{i-1}, E_{i-2}, E_{i-3}) + K_i + W_i \\ A_i &= E_i - A_{i-4} + \Sigma_0(A_{i-1}) + \mathrm{MAJ}(A_{i-1}, A_{i-2}, A_{i-3}). \end{aligned}$$

For the definition of the step constants K_i we refer to [27]. The bitwise Boolean functions IF and MAJ used in each step are defined by

$$\begin{aligned} \mathrm{IF}(x, y, z) &= x \wedge y \oplus x \wedge z \oplus z \\ \mathrm{MAJ}(x, y, z) &= x \wedge y \oplus y \wedge z \oplus x \wedge z, \end{aligned}$$

and the linear functions Σ_0 and Σ_1 are defined as follows:

$$\begin{aligned} \Sigma_0(x) &= (x \ggg 28) \oplus (x \ggg 34) \oplus (x \ggg 39) \\ \Sigma_1(x) &= (x \ggg 14) \oplus (x \ggg 18) \oplus (x \ggg 41). \end{aligned}$$

After the last step of the state update transformation, the initial values are added to the output values of the last step (Davies-Meyer construction). The result is the final hash value or the initial value for the next message block.

4.2 Extending the Attacks on SHA-256 to SHA-512

For our collision attacks on SHA-512, we use the same strategy as in the attack on SHA-256 in [25]. Since the message expansion and state update transformation is the same (except for larger word sizes and different rotation values in Σ_i, σ_i), we can use similar local collisions (with differences in the same message words) to construct semi-free-start collisions for the compression function on up to 38 steps.

The starting point for 38 steps uses a local collision which spans 18 steps, with differences in 6 expanded message words $(W_7, W_8, W_{10}, W_{15}, W_{23}, W_{24})$. For more details on how to select the starting point, we refer to [25]. Once the starting point is fixed, the main task is to find a differential characteristic and confirming message pair for this 18-step local collision.

By using the same guessing, backtracking and propagation strategy, we did not find any results for 38 steps of SHA-512. Due to the large word size and thus, larger search space, contradictions are detected much later in SHA-512. We have tried different approaches on every level, but did not succeed in finding any valid differential characteristics. The solution was to optimize the branching strategy to detect on one hand contradictions earlier and on the other side to reduce the search space faster.

4.3 Improving the Search using Look-Ahead Branching

To improve the search algorithm, we use the look-ahead branching heuristic proposed in Section 3.3. As discussed there, the choice of the subset size $s_{\max}$ is critical for the behaviour of the heuristic. We have evaluated different variants of the heuristic and get the best results for a limit of $s_{\max} = 16$. Larger values of $s_{\max}$ further reduce the tree depth, but due to the additional cost for evaluating more candidates, this does not improve the overall runtime.

Additionally, with larger subset sizes, the search tends to visit very similar subtrees again and again after each restart. This is particularly critical if the search space is limited to a few words, as in the focused search strategy described below. For other hash functions with larger states sizes or less focused search strategies, the optimal value for $s_{\max}$ may be very different.

Similar to [25], the guess-and-determine attack is separated into three stages. The rules of the guessing strategy are given in Table 1 and the three stages are summarized as follows:

Stage 1:
We first search for a consistent differential characteristic in the message expansion. Hence, we only add unconstrained bits ('`?`') and difference bits ('`x`') of W to the set U.

Stage 2:
We continue with the search for a differential characteristic in the state update. Hence, we add all unconstrained bits ('`?`') and difference bits ('`x`') of A and E to the set U. We pick decision bits more often from A, since this results in sparser characteristics for A. Experiments have shown that in this case, confirming message pairs are easier to find in the last stage.

Stage 3:
In the last stage, we search for confirming message pairs by guessing bits without difference ('`-`'). We only pick decision bits of A, E and W which are constrained by two-bit conditions, similar as in [23]. This serves as a preselection heuristic for the branching look-ahead.

Table 1. Decision rules in different search stages.

stage	decision bit	decision rule		
		probability	choice 1	choice 2
1–2	`?`	1	`-`	`x`
	`x`	$\left\{\begin{array}{l}1/2\\1/2\end{array}\right.$	`u` / `n`	`n` / `u`
3	`-`	$\left\{\begin{array}{l}1/2\\1/2\end{array}\right.$	`0` / `1`	`1` / `0`

4.4 Results

Using the improvements in the branching heuristic proposed in the previous section, we are able to find semi-free-start collisions for SHA-512 on up to 38 steps. Finding a differential characteristic together with a conforming message pair took 5441 seconds ($\approx$ 1.5h) on a cluster with 40 CPUs. This corresponds to a complexity of about $2^{40.5}$ evaluations of the SHA-512 compression function. The colliding message pair is given in Table 2 and the differential characteristic is shown in Table 3 in the appendix.

Table 2. Example of a semi-free-start collision for 38 steps of SHA-512.

h_0	e8626f53a3771964	2ae427b8c5065790	c8fd5a1628fc3337	0f362d297f82f987
	89166a0c022ffc40	c2c49c30e629239f	d1fa8bd692843025	ad4bba64c797e6ec
m	610519a88f0d2809	3addc83f01c8b179	84afa7a2772c6141	ad539854e64c9cce
	85450b73549b2085	7296b5291f31c0d9	fc978d9624e2c2cc	fffffffffffffffe
	92114cb9d2f4cd9b	34a3198b79871212	cca7f43154e38081	ac0598a589168fe1
	f32ae6a0070a8d2e	755aa5cada87e894	4b9bd7df3c94b667	65291f2b80cc8c51
m^*	610519a88f0d2809	3addc83f01c8b179	84afa7a2772c6141	ad539854e64c9cce
	85450b73549b2085	7296b5291f31c0d9	fc978d9624e2c2cc	0000000000000001
	92114cb9d2f4cd9c	34a3198b79871212	cca8143154e38079	ac0598a589168fe1
	f32ae6a0070a8d2e	755aa5cada87e894	4b9bd7df3c94b667	65291f2b80cc8c50
Δm	0000000000000000	0000000000000000	0000000000000000	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	ffffffffffffffff
	0000000000000007	0000000000000000	000fe000000000f8	0000000000000000
	0000000000000000	0000000000000000	0000000000000000	0000000000000001
h_1	946a28eedc3b2ff6	c4573d0a13ea6268	11f07b04b06900dd	897c606e4053bbe4
	2406aae9d58504b4	89b237932b061ba8	663402cb4bb1972c	d99c062dce945423

To show the benefit of our new look-ahead branching heuristic, we have performed some comparisons. Without look-ahead branching, we were able to find a semi-free-start collision for 27 steps of SHA-512 using 4 days on a cluster with 40 nodes, which corresponds to a complexity of about $2^{46.5}$. Using look-ahead branching with $s_{\max} = 16$ we can find differential characteristics with conforming message pairs within seconds on a standard PC (complexity $2^{26.5}$).

The heuristic can also be used to improve the search complexity for primitives with a smaller state to a certain extent. For example, experiments show a speedup of more than an order of magnitude for attacks on 27 or 38 steps of SHA-256. However, due to the heuristic nature of the improvement and the general sensitivity of the search procedure to different parameters, the effects are hard to quantify.

Unfortunately, we were not able to extend the semi-free-start collisions to collision attacks on the hash function. The main reason is that the resulting differential characteristics are quite dense and we do not have enough freedom to match the IV with practical complexity.

5 Conclusions

In this work, we have improved the best semi-free-start collisions on SHA-512 from 24 to 38 steps. Our attack has a practical complexity of $2^{40.5}$ and we have shown a colliding message pair. We get this result by applying the semi-free-start collision attack on 38 steps of SHA-256 to SHA-512. However, due to the increased word size, and hence increased search space, a straight-forward extension was not possible.

To get these results we have analyzed possible improvements in the guess-and-determine approach to find differential characteristics and conforming message pairs. We got the best results by optimizing the branching heuristic using ideas from SAT solvers. Our heuristic performs a randomized look-ahead selection of candidates which should be guessed first.

Future work includes to apply the look-ahead heuristic to more complex designs. Also, other techniques from SAT solvers may improve guess-and-determine attacks in differential cryptanalysis. However, a direct application of SAT solver techniques without taking high-level information on differential cryptanalysis into account is usually not successful. Finally, an open question is how to use our new results to improve the collision attacks on the SHA-512 hash function.

Acknowledgments. The work has been supported in part by the Secure Information Technology Center-Austria (A-SIT), by the Austrian Government through the research program FIT-IT Trust in IT Systems (Project SePAG, Project Number 835919), and by the European Commission through the FP7 Joint Technology Initiatives (Call ARTEMIS-2012-1, Project Arrowhead, Grant Agreement Number 332987).

References

1. K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for Step-Reduced SHA-2. In M. Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 578–597. Springer, 2009.
2. D. J. Bernstein and T. Lange. eBASH: ECRYPT Benchmarking of All Submitted Hashes, January 2011. Available online: `http://bench.cr.yp.to/ebash.html`.
3. M. Buro and H. Kleine-Büning. Report on a SAT competition. *Bulletin of the European Association for Theoretical Computer Science*, 49:143–151, 1993.
4. M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.
5. C. De Cannière and C. Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
6. N. Eén and N. Sörensson. An Extensible SAT-solver. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.
7. M. Eichlseder, F. Mendel, T. Nad, V. Rijmen, and M. Schläffer. Linear Propagation in Efficient Guess-and-Determine Attacks. In L. Budaghyan, T. Helleseth, and M. G. Parker, editors, *WCC*, 2013. `http://www.selmer.uib.no/WCC2013/`.

8. J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, Departement of computer and Information science, University of Pennsylvania, Philadelphia, 1995.
9. E. I. Goldberg and Y. Novikov. BerkMin: A Fast and Robust Sat-Solver. In *DATE*, pages 142–149. IEEE Computer Society, 2002.
10. M. Herbstritt and B. Becker. Conflict-Based Selection of Branching Rules. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 441–451. Springer, 2003.
11. M. Heule and H. van Maaren. March_dl: Adding Adaptive Heuristics and a New Branching Strategy. *JSAT*, 2(1-4):47–59, 2006.
12. M. Heule and H. van Maaren. Look-Ahead Based SAT Solvers. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 155–184. IOS Press, 2009.
13. S. Indesteege, F. Mendel, B. Preneel, and C. Rechberger. Collisions and Other Non-random Properties for Step-Reduced SHA-256. In R. M. Avanzi, L. Keliher, and F. Sica, editors, *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 276–293. Springer, 2008.
14. R. G. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. *Ann. Math. Artif. Intell.*, 1:167–187, 1990.
15. D. Khovratovich, C. Rechberger, and A. Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In A. Canteaut, editor, *FSE*, volume 7549 of *LNCS*, pages 244–263. Springer, 2012.
16. F. Landelle and T. Peyrin. Cryptanalysis of Full RIPEMD-128. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *LNCS*, pages 228–244. Springer, 2013.
17. G. Leurent. Analysis of Differential Attacks in ARX Constructions. In X. Wang and K. Sako, editors, *ASIACRYPT*, volume 7658 of *LNCS*, pages 226–243. Springer, 2012.
18. G. Leurent. Construction of Differential Characteristics in ARX Designs Application to Skein. In R. Canetti and J. A. Garay, editors, *CRYPTO (1)*, volume 8042 of *LNCS*, pages 241–258. Springer, 2013.
19. C. M. Li and Anbulagan. Heuristics Based on Unit Propagation for Satisfiability Problems. In *IJCAI (1)*, pages 366–371. Morgan Kaufmann, 1997.
20. J. Li, T. Isobe, and K. Shibutani. Converting Meet-In-The-Middle Preimage Attack into Pseudo Collision Attack: Application to SHA-2. In A. Canteaut, editor, *FSE*, volume 7549 of *LNCS*, pages 264–286. Springer, 2012.
21. P. Liberatore. On the complexity of choosing the branching literal in DPLL. *Artif. Intell.*, 116(1-2):315–326, 2000.
22. F. Mendel, T. Nad, S. Scherz, and M. Schläffer. Differential Attacks on Reduced RIPEMD-160. In D. Gollmann and F. C. Freiling, editors, *ISC*, volume 7483 of *LNCS*, pages 23–38. Springer, 2012.
23. F. Mendel, T. Nad, and M. Schläffer. Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In D. H. Lee and X. Wang, editors, *ASIACRYPT*, volume 7073 of *LNCS*, pages 288–307. Springer, 2011.
24. F. Mendel, T. Nad, and M. Schläffer. Finding Collisions for Round-Reduced SM3. In E. Dawson, editor, *CT-RSA*, volume 7779 of *LNCS*, pages 174–188. Springer, 2013.
25. F. Mendel, T. Nad, and M. Schläffer. Improving Local Collisions: New Attacks on Reduced SHA-256. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *LNCS*, pages 262–278. Springer, 2013.

26. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *DAC*, pages 530–535. ACM, 2001.
27. National Institute of Standards and Technology. FIPS PUB 180-3: Secure Hash Standard. Federal Information Processing Standards Publication 180-3, U.S. Department of Commerce, October 2008. Available online: `http://www.itl.nist.gov/fipspubs`.
28. National Institute of Standards and Technology. FIPS PUB 180-4: Secure Hash Standard. Federal Information Processing Standards Publication 180-4, U.S. Department of Commerce, March 2012. Available online: `http://www.itl.nist.gov/fipspubs`.
29. National Institute of Standards and Technology. SHA-3 Selection Announcement, October 2012. Available online: `http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_selection_announcement.pdf`.
30. I. Nikolić and A. Biryukov. Collisions for Step-Reduced SHA-256. In K. Nyberg, editor, *FSE*, volume 5086 of *LNCS*, pages 1–15. Springer, 2008.
31. M. Ouyang. How Good Are Branching Rules in DPLL? *Discrete Applied Mathematics*, 89(1-3):281–286, 1998.
32. S. K. Sanadhya and P. Sarkar. New Collision Attacks against Up to 24-Step SHA-2. In D. R. Chowdhury, V. Rijmen, and A. Das, editors, *INDOCRYPT*, volume 5365 of *LNCS*, pages 91–103. Springer, 2008.
33. M. Schläffer and E. Oswald. Searching for Differential Paths in MD4. In M. J. B. Robshaw, editor, *FSE*, volume 4047 of *LNCS*, pages 242–261. Springer, 2006.
34. J. W. Shay Gueron, Simon Johnson. SHA-512/256. Cryptology ePrint Archive, Report 2010/548, 2010. `http://eprint.iacr.org/`.
35. J. P. M. Silva. The Impact of Branching Heuristics in Propositional Satisfiability Algorithms. In P. Barahona and J. J. Alferes, editors, *EPIA*, volume 1695 of *LNCS*, pages 62–74. Springer, 1999.
36. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 1–18. Springer, 2005.
37. X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
38. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In R. Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.

Table 3. Differential characteristic for a semi-free-start-collision of SHA-512 reduced to 38 steps (bits with two-bit conditions highlighted).

i	A_i	E_i	W_i
-4	--	--	
-3	--	--	
-2	--	--	
-1	--	--	
0	--	--	--
1	--	--	--
2	--	--	--
3	--	--	--
4	--	--	--
5	--	---1-0	--
6	--	--------1----1101--0-1	--
7	---nun	-0-------00--00010---01---00------01--1------------1--1--1---n-u	uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuun
8	-----------uuun-u----------n-----u------n----------------------u	-100-0-1unnn00nnun0-111-0111---00-111-0uu-1-0-1-00000011-11--u00	----00---nuu
9	un--n------u---nu----u---n--nn-nuunn---n-nn-n-n-n-u-u--n--------	-10010-00uu10u00n0101u0--0nn1u11--nn--n0---00-1-001n11uuu11--001	--
10	--	-u-u-0-u10nun-1u1uu-01uu-u0u011uu-1n1u1-0nu1n1n-0100n1u10n11-uu0	------------nuuuuuu-------------------------------------unnnn---
11	--	-11110100110011u01011n01110uu0101nun001uu10nn0u0unu111-u01010n0u	---1
12	---n------	n1100--0--0u--n0-n0-0n1101uuu11---1--110u01101-100n0110u0n--1unn	---0
13	--	un--n1101-1u0000nunnn0110n-00u01-10u01n0u-0u1u1uu0u-u100nn0-0011	--
14	--	10--0-10----101110111010010100----11--1-11-1-1-11-100--unn---001	--
15	u------u--	n1--1-0u---101011010011111110100--00001-00-1-1-00-0-11-0-u--11-1	---u
16	---u	n-010-nu0-01nuuuuuuuuuuuuuuuu100-nuu1110uu111-u010-111-00n-00-un	--
17	--	11----01----010--100nuuuuuuunuuuuunuuuu110-11101100010011011111n	--
18	--	0-----00----0---------------01111101111-01----1000001010000000nu	--
19	--	u----unn--------------------------------11----nuuuuuuuuuuuuuuuuu	--
20	--	00001000100010101010-------------1111unnnnnnnnnnnnnnnnnnnnnnnnnn	---1
21	--	1----111-----------------------------111111111-11111111111111111	--
22	--	-------------------------------------000000000000000000000000000	--
23	--	--	unuuuuuu--
24	--	--	---n
25	--	--	--
26	--	--	--
27	--	--	--
28	--	--	--
29	--	--	--
30	--	--	--
31	--	--	--
32	--	--	--
33	--	--	--
34	--	--	--
35	--	--	--
36	--	--	--
37	--	--	--