**Automatic Search for**

**Differential Trails in SHA-2**

Mitchell Grout

Supervised by Ryan Ko, Cameron Brown, Aleksey Ladur

Date: 12/03/18

**Summary:** We propose an automatic tool for finding and improving differential trails in SHA-2, which can be extended for other cipher based hash functions. This tool will randomly generates random input differentials, and generates valid trails for SHA-2. These trails can then be optomized by use of optomization algorithms.

The SHA-2 family of hash functions are used widely in security to ensure the integrity of files, generate digital signatures, and securely store passwords. These functions take some variable-length input, and provide a fixed-length output, which is assumed to be unique to that input. We often regard these hash values as fingerprints for the input. If we have two distinct inputs, m and m\*, and a hash function h, then we call h(m) = h(m\*) a collision; meaning the fingerprints of m and m\* are the same. Since h maps an infinite domain onto a codomain, we will always be able to find at least two distinct m and m\* such that their hashes collide. However, at this time, there is no feasible way of finding m and m\*. Current best attacks require 2^60 operations to get halfway.

We propose an automated tool for the discovery of differential trails for SHA-2. This tool will take a user-supplied input differential, using the generalized bit conditions format, and proceed to find any valid trails that hold with a high probability. We will then take the valid trails, and determine sufficient conditions on the message which allows the trail to hold with a 100% certainty. Finally, we will determine if the trail contains any contradictory conditions, and if so, discard.

To fully automate this tool, we propose an extension which wil try random input differentials of a specific form. [1] proposes that input differentials should have a sparse and dense section; the first 8 message words should be freely chosen, and the final section should be constrained. We will analyse this by trying various different constraints on the input differential, changing the sizes of various sections, increasing and decreasing the size of the dense section.

We also seek to determine whether or not input differentials can be optomized by classic optomization techniques, specifically genetic algorithms and simualted annealing. We will take a sample of good differentials generated by our previous tool, and use these to build new differentials, and gauge their fitness in comparison to their ancestors. In order to make this computation more feasible, we will distribute it over a number of computers.

To prove feasibility, we will begin by implementing the aforementioned tools for a custom, reduced SHA-2 named MAW32. The feasibility of implementing our tools for MAW32 will serve as an indicator for feasibility for SHA-2.

References:

**Figure 1:** Generalized conditions on a pair of bits

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | (x,x\*) | (0,0) | (1,0) | (0,1) | (1,1) | | ? | ✓ | ✓ | ✓ | ✓ | | - | ✓ | - | - | ✓ | | x | - | ✓ | ✓ | - | | 0 | ✓ | - | - | - | | u | - | ✓ | - | - | | n | - | - | ✓ | - | | 1 | - | - | - | ✓ | | # | - | - | - | - | | |  |  |  |  |  | | --- | --- | --- | --- | --- | | (x,x\*) | (0,0) | (1,0) | (0,1) | (1,1) | | 3 | ✓ | ✓ | - | - | | 5 | ✓ | - | ✓ | - | | 7 | ✓ | ✓ | ✓ | - | | A | - | ✓ | - | ✓ | | B | ✓ | ✓ | - | ✓ | | C | - | - | ✓ | ✓ | | D | ✓ | - | ✓ | ✓ | | E | - | ✓ | ✓ | ✓ | |

**Figure 2:** MAW32

[ TODO: FIPS-style definition of MAW32 ]

**Figure 3:** MAW32 Reference Implementation

[ TODO: Reference implementation of MAW32 ]