**Automated Searching for**

**Differential Characteristics in SHA-2**

Mitchell Grout

Supervised by Ryan Ko, Cameron Brown, Aleksey Ladur

Date: 12/03/18

**Summary:** Successful attacks on modern cryptographic hash functions rely on the existence of good differential characteristics, which allow for the construction of a colliding pair of messages. The construction of these characteristics was historically done by hand, but has become a more automated process in recent years. Automated tools have been proposed for finding good characteristics for SHA-2, and have been used to extend existing attacks. We propose an automatic tool for finding differential characteristics in SHA-2, and improving existing characteristics. This tool will be able to determine if a given input differential admits any differential characteristics with a high probability, and can randomly trial input differentials to automatically search for good characteristics. It will also be able to refine existing good characteristics using classic optimization algorithms. It may also be run in a distributed system to increase overall performance and reduce individual resource requirements.

**Introduction**

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. The most common of these functions is SHA-256; all other members of the SHA-2 family can be regarded as a variant of SHA-256, with differing initial values and output lengths. These functions take some variable-length input, and map it to some fixed-length output. In the case of SHA-256, the output is 256-bits long. Due to the size of the output, we often assume that the output of a hash function is unique; as a result, we can regard the output of a hash function as a type of fingerprint for the input. If we have two distinct inputs, and , such that they share the same hash, then we call a colliding pair. Since SHA-2 hash functions map an infinite domain onto a finite codomain, we are guaranteed the existence of infinitely many colliding pairs.

Being able to efficiently construct colliding pairs in SHA-2 is a massive security problem, due to the assumed uniqueness of the output. Since SHA-2 is used to ensure the integrity of files, being able to efficiently construct colliding pairs would mean it would be possible to take an existing file, and produce a new file which shared the same SHA-2 hash. This could then be used for malicious purposes, such as intercepting a file being transferred over a network, and sending the new file instead. If the sender communicated only the SHA-2 hash of the transferred file, then the recipient would not be able to tell that they had received the wrong file. As a result, it is necessary to cryptanalyze SHA-2 and other common cryptographic hash functions currently in use, to determine if it is indeed feasible to efficiently construct colliding pairs, and if so suggest their deprecation in favour of more secure hash functions.

In SHA-256, it is possible to construct a colliding pair of messages in time. This is possible through the ‘birthday attack’, a statistical attack which is bounded by , where is the number of bits in the output. Although this is infeasible as a method to generate colliding pairs, it serves as an upper bound for the complexity of attacks. A promising approach to finding a method of constructing colliding pairs for SHA-2 is with differential cryptanalysis[1]. This technique was first described by Biham and Shamir, to cryptanalyze the Data Encryption Standard (DES), and involves analysing the effect of differences in input pairs on the differences of output pairs. This has been used successfully to find colliding pairs for MD4[2], MD5[3], and SHA-0.

**Design**

We propose a tool which allows for automated searching for differential characteristics in SHA-2. We will use a process similar to that described in [4]. The process of searching for a characteristic can be separated into four steps:

* Construction of an input differential
* Construction of differential characteristics using the input differential
* Computing sufficient conditions to make the characteristic hold with 100% certainty
* Determining if the sufficient conditions are non-contradictory

An input differential is a description of the difference between the two values in the colliding pair. To describe this differential, we use the generalized conditions on pairs of bits[5]. To build our input differential, we will use a technique similar to what is described in [6]. We will identify sparse and dense sections in our input differential, and randomly fill the dense section with constraints on bits. We opt for guessing-and-checking the dense section instead of the backtracking algorithm in [5] for potential speedup and simplicity of implementation.

To construct a differential characteristic, we must propagate our input difference throughout the hash function, up to a specified number of rounds. This will be done component-wise, as this allows for more flexibility in the characteristic chosen, and allows us to take advantage of known qualities in the component functions. To determine how a differential propagates through a function, we either use known qualities of the function to propagate the differential, or we use random sample of inputs to determine the probability of a specific output differential. We will retain all probabilities which are above a given threshold probability, and propagate each of these individually. This has exponential growth in both memory and time, so it is important we pick an appropriate value for the threshold probability.

To compute sufficient conditions for a characteristic to hold, we do nothing.

When we have constructed a characteristic, it is necessary to determine if it is contradictory. This occurs when two of the sufficient conditions are mutually exclusive. To perform this, we will use a technique similar

Once we have constructed a set of differential characteristics, we will try to use classic optimization techniques to improve them further. The techniques that we will focus on are simulated annealing, and genetic algorithms. To help accelerate this process, we will distribute the work over several computers.

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.

**Planned Approach**

The progress for this project is separated into two major sections; the feasibility trial, and actual implementation. In order to gauge how effective our tool is, we have designed a simple variant of SHA-2, named MAW32. The definition for MAW32 is in figure 2, and reference implementation in figure 3. The first 6 weeks will be spent focusing on implementing our tool specifically for MAW32, and running tests to determine if the produced characteristics are suitable. The next 10 weeks will then be spent on implementing our tool for SHA-256. This will also involve running tests over a longer period to determine if we can find characteristics that are similar to the current best-known characteristics for SHA-256. After this, the remaining time will be spent collecting good characteristics from the tool, and writing up the report.

**Evaluation**

The measure of success for this project is if we can produce differential characteristics which admit a colliding pair with complexity better than .

**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 ]

**References**

1. Biham, E.S., Adi, *Differential Cryptanalysis of the Data Encryption Standard*. 1993: Springer-Verlag New York.

2. Wang, X., et al. *Cryptanalysis of the Hash Functions MD4 and RIPEMD*. 2005. Berlin, Heidelberg: Springer Berlin Heidelberg.

3. Wang, X. and H. Yu. *How to Break MD5 and Other Hash Functions*. 2005. Berlin, Heidelberg: Springer Berlin Heidelberg.

4. Mendel, F., T. Nad, and M. Schläffer. *Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions*. 2011. Berlin, Heidelberg: Springer Berlin Heidelberg.

5. De Cannière, C. and C. Rechberger. *Finding SHA-1 Characteristics: General Results and Applications*. 2006. Berlin, Heidelberg: Springer Berlin Heidelberg.

6. Mendel, F., T. Nad, and M. Schläffer. *Improving Local Collisions: New Attacks on Reduced SHA-256*. 2013. Berlin, Heidelberg: Springer Berlin Heidelberg.