Automated Searching for Differential Characteristics in SHA-2

Mitchell Grout

Supervised by Ryan Ko, Cameron Brown, Aleksey Ladur

12th March 2018

Summary: Successful attacks on modern cryptographic hash functions rely on the existence of good differential characteristics, which allow for the efficient 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 difference can be used to construct a differential characteristic which holds with a high probability, and can randomly trial input differences to automatically search for good differential 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. These functions take some variable-length input of bits, known as the message, and output some fixed-length output of bits, known as the hash. The number of bits in the hash is typically very large, ranging from 224-bits to 512-bits in SHA-2. These functions are assumed to satisfy the following key properties:

- Small differences in messages should result in large differences in hashes
- It is infeasible to generate a message from a hash
- It is infeasible to find two messages with the same hash

These properties, along with the large number of possible hash values, allows us to assume the hash of a message is almost unique, and as such we often regard the hash as a type of fingerprint for the message. However, as the output is not truly unique: it is possible to find two distinct messages m and m^* which share the same hash; we call (m, m^*) a colliding pair of messages.

Being able to efficiently construct colliding pairs in SHA-2 is a significant problem, due to the assumed uniqueness of the hash. SHA-2 hashes are often used to detect if a file has been modified, allowing for quick integrity checks for files transferred over a network. However, if there were an efficient method of constructing colliding pairs in SHA-2, then it would become feasible to construct two files which shared the same hash. This could then be used for malicious purposes, such as intercepting files on the network and replacing them with the colliding file. As a result, it is necessary to cryptanalyze SHA-2, and all other common cryptographic hash functions currently in use, to ensure it is not possible to efficiently construct colliding pairs. If it is indeed possible, then the hash function can be recommended for deprecation, and be replaced with more secure hash functions.

SHA-256 is a commonly used hash function from the SHA-2 family, which has hash sizes of 256-bits. The internal structure of SHA-256 is described specifically in FIPS180-4. In SHA-256, it is possible to construct a colliding pair of messages with 2^{128} hash operations. This is possible through the 'birthday attack', a statistical attack which is bounded by $O(\sqrt{2^n})$ hash operations, where n 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. The current approach to constructing colliding pairs for SHA-2 is with differential cryptanalysis, which was first described by Biham and Shamir (1993) in order to cryptanalyze the Data Encryption Standard.

Differential cryptanalysis involves determining the effect of differences in input on differences in output. A differential characteristic, denoted $\Delta_0 \rightarrow \Delta_1 \rightarrow \cdots \rightarrow \Delta_n$, states that a pair of inputs which differ by Δ_0 will differ by Δ_1 after round 1, then by Δ_2 after round 2, and will eventually differ by Δ_n in output. If this trail holds with probability P, then on average we must test $1/P$ pairs, with each pair differing by Δ_0, to find a collision. Using this, we may restate our problem: efficiently finding a colliding pair in SHA-2 is equivalent to finding a differential trail which holds with a high probability, and $\Delta_n = 0$. This technique has been used successfully to efficiently find colliding pairs for MD4 (Wang, Lai, Feng, Chen, & Yu, 2005), MD5 (Wang & Yu, 2005), and SHA-0 (Naito et al., 2006).

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 by Mendel, Nad, and Schläffer (2011). The process of searching for a characteristic can be separated into four steps:

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

An input difference is a description of the difference between the two values in the colliding pair. To describe this, we will use the generalized conditions on pairs of bits as described by De Cannière and Rechberger (2006). This allows for complex relationships between bits to be written compactly: for example, all pairs of bytes of the form $\{(x, x^*) : x_7 = 1; x_1 \neq x_1^*; x_0 = x_0^*\}$ can be described by $\nabla = [\mathrm{A}?\,?\,?\,?\,?\,\mathrm{x}\,-]$. To build our input difference, we will use a technique similar to Mendel, Nad, and Schläffer (2013). Specifically, we will identify sparse and dense sections in our input difference; the sparse section will indicate points where the two messages are entirely equal, and the dense section will be filled with random constraints on bits. We opt for completely random choice

unlike the backtracking algorithm described in De Cannière and Rechberger (2006), for simplicity of implementation.

To construct a differential characteristic, we take a given input difference, and propagate it through each component in the hash function, up to a specified number of rounds. By propagating through the individual components, we have more flexibility, as we can take advantage of known qualities for individual components. To determine how an input difference propagates through a component, we either use it's known qualities, or use random sampling to determine the probability of a specific output difference. In the case of random sampling, we will take a random input, and compute a second input which differs by the specified input difference. Then, we compute the value of both inputs through the component, and measure the output difference. This will be repeated for a given number of iterations, and probabilities for output differences computed. Output differences which are below a given threshold probability will be discarded. Finally, we continue the propagation process with every significant output difference computed. This has exponential growth in both memory and time, so it is important we pick an appropriate value for the threshold probability.

Once we have a characteristic which holds with a high probability, we will compute the sufficient conditions for the characteristic to hold. These are conditions on individual differences in state which allow for input differences to propagate with a 100% certainty. In bitwise functions, these conditions can be found readily, so we are concerned with finding conditions for non-linear functions such as addition of differentials. We will approach this by starting with the bit unconstrained, and use random sampling to determine what constraints should be imposed. We can use the results of this sampling to find an exact condition for every individual bit in the state. Although this step can be performed while the characteristic is being constructed, we opt to keep it separate for simplicity.

When we have constructed a characteristic, it is necessary to determine if it is contradictory. This occurs when at least two of its conditions are mutually exclusive. In Mendel et al. (2011), this step is performed in conjunction with the discovery of sufficient conditions, which allows for backtracking and changing conditions all the way up to changing the input difference. We will similarly perform contradiction checks as we compute sufficient conditions. To determine if a contradiction is present, we will look at the set of conditions imposed on a variable before it is used in a component, and the set of conditions imposed after, and determine if any two conditions are mutually exclusive.

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. We will use these to generate new input differences from an existing pool of known-good differentials. The reason these algorithms were selected is due to the nature of hash functions: small changes to input result in large changes to the output. However, we similarly have that a complex differential will cause a small change to the output. As such, by taking a complex differential generated by our previous process, and improving it using our optimization algorithms, we hope to produce another complex differential which is even better than the input. To help accelerate this process, we will distribute the work of optimization over several computers.

Planned Approach

The progress for this project is separated into two major sections; the feasibility trial, and actual implementation. To gauge if our proposed implementation is feasible, and if it is effective at searching for characteristics, we have decided to implement it for a simple, custom variant of SHA-2 named MAW32. This algorithm operates on smaller messages, and produces smaller output, but has a similar internal structure to SHA-256. The first six 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 ten 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 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 final report.

Evaluation

A colliding pair for SHA-256 can be constructed in 2^{128} time by use of the birthday attack, so an attack which can find a colliding pair faster than this can be considered a success. For our tool, we will regard it as a success if it is able to, in a reasonable amount of time, repeatedly produce differential characteristics which allow for the creation of colliding pairs in better than 2^{128} time. The level of success is determined by the quality of these characteristics, as well as the quality of characteristics found by classic optimization. Depending on the success of this tool, we will be able to comment on the current security of the SHA-2 family of functions, and whether they should be suggested for deprecation in favour for more recent hash functions such as SHA-3.

References

Biham, E., & Shamir, A. (1993). *Differential Cryptanalysis of the Data Encryption Standard*: Springer-Verlag New York.

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

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

Mendel, F., Nad, T., & Schläffer, M. (2013). *Improving Local Collisions: New Attacks on Reduced SHA-256*, Berlin, Heidelberg.

Naito, Y., Sasaki, Y., Shimoyama, T., Yajima, J., Kunihiro, N., & Ohta, K. (2006). *Improved Collision Search for SHA-0*, Berlin, Heidelberg.

Wang, X., Lai, X., Feng, D., Chen, H., & Yu, X. (2005). *Cryptanalysis of the Hash Functions MD4 and RIPEMD*, Berlin, Heidelberg.

Wang, X., & Yu, H. (2005). *How to Break MD5 and Other Hash Functions*, Berlin, Heidelberg.