Status Update: End of January MATH 490 - Independent Study

Alexander Scheel[a]

[a] Iowa State University

End of January status update for MATH 490: Independent Study at Iowa State University. Proposes to study SHA-3 and outlines current and future areas of study and completed work.

MATH 490 | Status Report | SHA-3 | hash_framework

1. Introduction

The purpose of this independent study is to study cryptography from a mathematical background. The target object is the Keccak hash function as standardized by NIST as SHA-3 and its collision resistance. A cryptographic hash function maps arbitrary length input strings to a fixed-length digest; finding two elements of the input domain which map to the same output should require on the order of $2^{\frac{l}{2}}$ amount of work, where l is the output length of the hash function. SHA-3 provides an excellent study due to its relatively simple construction as compared to previous standardized hash functions. The goal of this independent study is to better understand cryptanalysis techniques such as differential cryptanalysis and computer-aided proof techniques.

The rest of this status report is organized as follows: section 2 details the construction of SHA-3, section 3 details some existing literature in the field, section 4 details the completed work, section 5 details current tasks, and section 6 concludes the status report with a list of future tasks and deliverables.

2. Overview of SHA-3

SHA-3 is the latest generation of hash function standardized by NIST in FIPS-202, published in 2015 [1]. The team behind this function includes Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer [2], a European team. This hash function differs significantly from the previously accepted construction of MD4, MD5, SHA-1, and SHA-2: while the previous hash functions relied on the Merkel-Damgård construction, SHA-3 uses a sponge-function design.

The Merkel-Damgård construction for cryptographic hash functions follows what has been termed the ARX construction: arithmetic-rotation-xor. This construction transforms an initial state to an ouput state according to a predefined series of round functions. Each round function is typically different from the next, and each operates on a different portion of the block. Assume a 32-bit hash function. Typically there are between 4 and 8 state variables, in this case, 32-bit unsigned integers. The input block is split into 32-bit chunks and expanded according to the defined block schedule. This block schedule can either be simple (in the case of MD4 and MD5, simply permutations of the

ordering of the chunks) or complex (in the case of SHA-1 and SHA-2, which involve xor-ing various chunks together to produce additional chunks).

Then, starting from the initial state, each round applies the corresponding round function to the current state and the current block chunk, producing a new state variable. The round function is made up of several additions, at least one rotation and an xor; hence the term ARX construction. This state variable is then used to replace an element of the current state (directly in the case of MD4 and MD5, else in some fixed pattern in SHA-1 and SHA-2). Lastly, after the last block chunk has been processed, the initial state and the final state are added together to produce the output state. Multiple blocks are then handled by assuming this output state as input to the next block, and some suitable padding scheme is applied to the input before splitting into blocks. In the case of MD4, MD5, SHA-1, SHA-2 256 and SHA-2 512, all of the state variables are directly outputted. Therefore, an extension attack is possible: given any input B, which produced output O, it holds that all C produce $h(O, C) = h(IV, B||C)$, where IV is the default initial state value.

In contrast, Keccak uses a construction they call a sponge construction. Rather than several independent state variables, Keccak uses a single array as state, of size 25×2^l for $0 \leq l \leq 6$. This is conceptualized as a $5 \times 5 \times 2^l$ cuboid. Blocks are mixed in once, via a simple xor, and the initial state is the all zero array. Then, a round function is permuted 24 times. This round function is composed of five functions: $\theta, \rho, \pi, \chi, \iota$. Each function is a bijection on the set of all state arrays. Thus, after 24 rounds are applied, a portion of the resulting state is specified as output. If multiple blocks are given, the entire state is used as input, with the next block xored in, before applying the round function.

SHA-3 standardized $l = 6$, producing a state array of 1600 bits. Further, since SHA-2 standardized several different security margins, (224, 256, 384, and 512), SHA-3 did as well. Take the security margin to be b bits. Then the input block size is $1600 - 2 \times b$ and b bits of the resulting state is taken as output. While this prevents length extension attacks on every block, this opens the possibility for a specialized extension attack with lower possibility. Suppose S_0 is the initial state after mixing in a block. If keccak-$p(S_0)$ has the same lower $2 \times b$ bits as S_0, then $h(S_0 \oplus C) = h(S_0||C)$, for all messages C.

3. Review of Literature

The canonical reference for SHA-3 is FIPS-202 [1]. However, there are several useful supplementary materials to use in conjunction with FIPS-202. The first is the core Keccak Reference: published by the Keccak team, this contains extended information underlying their choice of functions, properties of these functions, documentation on inverse functions, and external references [3]. Supplementing this is the Keccak Code Package, which contains many implementations of Keccak and test cases for several bit widths [4]. Lastly, the PhD thesis of J. Daemen seems to have heavily influenced the construction of Keccak, and provides extensive reference material [5].

Of the attacks I've looked at, the work of E. Homsirikamol, et. al, is most relevant to my existing work [6]. They directly applied SAT solvers to find collisions in Keccak. However, they used only the most simple model and were only able to find a two-round collision within 30 hours. Thus it serves more as a benchmark than as useful cryptanalysis. The work of P. Morawiecki and M. Srebrny also looks at applying SAT solvers to Keccak, but explicitly focused on preimage attacks [7]. In this instance, it appears the authors only considered a direct preimage attack with little other work towards splitting the work between multiple servers or doing advanced exploration of possible paths. However, they did appear to build a toolkit, CryptLogVer, for studying this problem.

4. Completed Work

Completed is an initial survey of the literature on the subject of the construction of SHA-3 and attacks against it. Further, an implementation of SHA-3 has been added to my hash_framework package [8]. This has been tested against the vectors in the Keccak Code Project [4], and is shown correct.

Several initial attack vectors have been considered as well. Towards studying the specialized extension attack, an initial pass trying to bound the error in the remaining bits. However, up to an input difference of 5, no alternate paths have been found producing a net-zero difference in the tail. This has yet to be studied on reduced width versions of Keccak.

Towards studying collision attacks, a similar error-bounding property has been studied on front prefixes. Given that there are k bits of difference in the input, and given an input margin of m (where the remaining 1600 - m bits are all zero), what is the largest M such that, after applying only the first round, the difference between the two blocks is zero in the first M bits. This has been exhaustively studied on the $l = 3$ and $l = 4$ bit-width versions of Keccak, and partially on the full width ($l = 6$). The results are startling: for small, even k, M quickly grows large, whereas for odd, small k, M remains fixed at a relatively small number. Thus there is the potential to exploit this. However, as k grows large, the odd and even values of k converge in structure. This can seen in the links attached to this email.

5. Current Work

Currently, I am rewriting my job distribution mechanism to work more efficiently on larger deployments such as an HPC cluster or AWS cloud instances. This will add several desirable features: better fault tolerance, handling job distribution automatically and prioritizing between several tasks, and devoting a set number of cores per task. Each task will then have many jobs and statistics about them can be collected and centralized (such as number of variables, time to solve, etc.). This would provide the basis for more closely analyzing the performance of the entire system.

6. Future Work

After this rewrite, I'll look at exploring a few areas: analyzing the impact of each of the sub-round functions $(\theta, \rho, \pi, \chi, \iota)$ on various differential paths, continuing the above work on analyzing input and output margins on k-bit differences, and looking at total, given k bits of input difference, what possible combinations of output differences are possible.

Further, I'll implement HPC-specific workloads. The most obvious starting point is to analyzing small versions of Keccak ($l = 4$) for two-variable correlations across each of the several rounds. Further, studying the per-round correlation of initial input differential on resulting intermediate round differentials would be an interesting result as well, for small l.

Once these properties have been more closely explored, there is potential for proving theorems based on the results. However, I think it is important to get some initial results from these various areas to see how they all fit together and what avenues are most likely to help attack Keccak.

7. Bibliography

1. (2015) Fips pub 202, sha-3 standard: Permutation-based hash and extendable-output functions. U.S.Department of Commerce/National Institute of Standards and Technology. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.
2. Team K (2008 - 2017) Team keccak | home (online). https://keccak.team/.
3. Bertoni G, Daemen J, Peeters M, Van Assche G, Van Keer R (2011) The keccak reference - version 3.0. https://keccak.team/files/Keccak-reference-3.0.pdf.
4. Team K (2018) Keccak code project github.com. https://github.com/gvanas/KeccakCodePackage.

5. Daemen J (1995) Cipher and hash function design strategies based on linear and differential cryptanalysis. http://jda.noekeon.org/JDA_Thesis_1995.pdf.
6. Homsirikamol E, Morawiecki P, Rogawski M, Srebrny M (2012) Security margin evaluation of sha-3 contest finalists through sat-based attacks (Cryptology ePrint Archive, Report 2012/421). https://eprint.iacr.org/2012/421.pdf.
7. Morawiecki P, Srebrny M (2010) A sat-based preimage analysis of reduced keccak hash functions (Cryptology ePrint Archive, Report 2010/285). https://eprint.iacr.org/2010/285.pdf.
8. Scheel A (2018) hash_framework. https://github.com/cipherboy/hash_framework.