A Collection of Thoughts on the Keccak/SHA-3 Construction

Alexander Scheel

Iowa State University

MATH 490 | Advisor: Dr. Clifford Bergman | SHA-3 | hash_framework

1. A Series of Introductions

This paper presents the work of the author towards the completion of the requirements for the MATH 490 Independent Study course under Dr. Clifford Bergman at Iowa State University of Science and Technology. This work was an extension of the author's Honors Project under Dr. Eric Bergman and utilized the resulting hash_framework project. Additional artifacts related to this project can be seen in the keccak-attacks repository. A permanent location for this document is in the papers repository.

Within this section are a series of introductions which provide necessary background on the topics of cryptographic hash functions, the development of Keccak/SHA-3, and its structure. While certain sections can be skipped if the reader has the prerequisite knowledge, hopefully all readers find the material engaging and useful. Following these introductions, this paper presents the analysis of the Keccak/SHA-3 hash function before concluding with a final evaluation and further work.

The remaining sections outside of the introductions discuss 1) various mathematical properties of the core round functions, 2) exhaustive collision searches on small instances, 3) correlation matrices, and 4) fixed point attacks.

Introduction on the Topics of Cryptographic Hash Functions. While there are many applications of pure mathematics, few are as demanding and shrouded in secrecy as cryptography. Cryptography exists because of the fundamental need of civilizations, governments, and individuals to keep secrets secure from devoted adversaries. While modern cryptography combines the disciplines of mathematics, computer science, and computer engineering, prior to the turn of the 20th century, cryptography lacked much of its modern rigor.

Within cryptography's collection of algorithms, few are as useful as hash functions have proven to be to cryptographers and non-cryptographers alike. A hash function maps arbitrary length binary strings to binary strings of a fixed length. Because the inputs are of arbitrary size, by the pigeonhole principle, there must exist at least one collision. However, to be considered cryptographically secure, finding such a collision must be hard (on the order of $2^{\frac{b}{2}}$ for b the number of bits in the output).

Further, a cryptographic hash function must be resistant finding a preimage: for a given output value v, finding an input x such that $f(x) = v$ should be roughly on the order of 2^b. Given such

E-mail: alexander.m.scheelgmail.com

an attack, it is possible to find a collision: choose a random input, compute its hash, then use the preimage attack to find a different input for that hash; these two inputs form a collision pair. Thus a preimage attack is stronger than a collision attack.

However, if the attacker is allowed to choose only one message (that is, the challenger chooses the other), this forms a second preimage attack. More formally, for a given fixed message x, the attacker seeks a $y \neq x$ such that $h(x) = h(y)$. While a collision attack is only valid for some such messages, x, a preimage attack has to be valid for all such x; thus it is a stronger attack than collision attacks, but weaker than a preimage attack as it presupposes an existing message.

This independent study focused on a single hash function, Keccak, which was chosen by NIST to be the next standardized algorithm, SHA-3 [1]. This hash function differs from previously standardized algorithms in two major ways: one, it is not based on the Merkel-Damgård construct and is rather based on the Sponge construction, and two, the internal round function is not a compression function and is instead a permutation. Outside of a few theorems done by hand and verifying the lack of low-order correlations, we focused on using automated tools to learn more about the structure of SHA-3.

In the late 1990s, F. Massacci started analyzing cryptographic constructions using Boolean Satisfability [2]; however, applying this to hash functions is a largely understudied field. E. Homsirikamol tried brute forcing collisions in SHA-3 to limited results [3], and P. Morawiecki performed similar work for preimages [4]. However, neither contributed useful, general cryptanalysis techniques to scale attacks against reduced rounds (and reduced sizes) to larger numbers of rounds. Our work begins exploring new techniques and several consequences in the following sections.

Introduction to Terminology. This section contains a collection terminology useful for discussing SHA-3 and Boolean Satisfiability.

We define $\Sigma = \{0, 1\}$ to be the alphabet.

We define the set $W = \{1, 2, 4, 8, 16, 32, 64\}$ to be the powers of two typically used in constructing the Keccak widths; $w = 64$ is standardized for use in SHA-3, though any power of two can be used.

For a given $w \in W$, we define $S_w = \Sigma^{25w}$, to be the set of all binary strings of length $25w$; these represent the possible states (which are binary strings that map onto an indexible array A described later) and each round permutation maps $S_w \mapsto S_w$.

A binary string $s \in S_w$ can be indexed directly (in a zero-indexed manner, i.e., the starting bit of s is denoted $s[0]$), or via a 3 dimensional structure of size 5 by 5 by w. This is typically done in conjunction with an uppercase letter, $A = s$, and then $A[x, y, z] = s[w(5y + x) + z]$. This is in accordance with FIPS 202 [1].

Introduction on the Development of Keccak/SHA-3. NIST standardized Keccak as SHA-3 in FIPS 202 [1]; Keccak was chosen as the winner of the corresponding hash function design competition started in 2009. Designed by the Keccak Team (G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer [5]), Keccak features a novel construction.

Keccak is built around the concept of a state cube: a 5 by 5 by w cube of bits (where w is a power of 2). By increasing w, a higher security margin can be given, at the expense of increasing the computational intensity. However, because internal round operations on this state cube are bijective, it can be implemented efficiently in hardware with minimal gate and propagation delays. Typically this state is called A when represented as a cube, and s when represented as a (flat) bit string. There are five major functions which compose to form a single round: θ, ρ, π, χ, and ι; each maps bijectively over the entire state space (i.e., elements of 2^{25w}).

θ is an eleven way XOR and is thus a linear map. For all $0 \leq x \leq 4$ and $0 \leq z < w$, define:

$$C[x,z] := \oplus_{0 \leq y \leq 4} A[x,y,z]$$
$$D[x,z] := C[(x-1) \bmod 5, z] \oplus C[(x+1) \bmod 5, (z-1) \bmod w]$$

Then:

$$A'[x,y,z] := A[x,y,z] \oplus D[x,z]$$

ρ is a pure permutation function, permuting the locations of bits in a lane (the z component) of the state cube. For all $0 \leq z < w : A'[0,0,z] := A[0,0,z]$. Then let $x, y = (0,0)$, and for $0 \leq t \leq 23$:

$$\forall 0 \leq z < w : A'[x,y,z] := A[x,y,(z - \frac{(t+1)(t+2)}{2}) \bmod w]$$
$$let x, y = (y, (2x+3y) \bmod 5)$$

π is another pure permutation function, permuting locations along the face of the cube. For all $0 \leq x \leq 4, 0 \leq y \leq 4, 0 \leq z < w$:

$$A'[x,y,z] := A[(x+3y) \bmod 5, x, z].$$

χ is the only non-linear function and is of degree one (with inverse of degree three). For all $0 \leq x \leq 4, 0 \leq y \leq 4, 0 \leq z < w$:

$$A'[x,y,z] := A[x,y,z] \oplus (\neg A[(x+1) \bmod 5, y, z] \wedge A[(x+2) \bmod 5, y, z]).$$

ι is a fixed-value XOR useful for preventing trivial fixed points, and is the only function dependent on the current round number. Refer to page 16 of FIPS 202 ([1]) for its specification; it is not discussed extensively in this paper.

The sponge construction can be described as follows. Fix w as a power of two. To have a security margin of m bits, the output needs to have m bits; however, if the attacker has complete control of all $25w$ bits of state, they would be able to choose the output easily because Keccak is bijective. Thus, restrict the input to $25w - 2m$ bits and set the remaining $2m$ bits to zero. To find a preimage for a m-bit output, the attacker would have to find a $25w - m$ bit suffix such that the preimage of that suffix has $2m$ zeros. Similarly, for a collision attack, an attacker has to find two $25w - 2m$ messages such that their hashes have the same prefix of length m. Under an ideal round permutation, clearly these are non-trivial and hard to find in general.

For a multiple block message the procedure is thus as follows: call the composition of 24 rounds to be KECCAK-f. Start with $s = 0^{25w}$. Splitting the message into chunks of $25w - 2m$ (and padding according to the scheme defined in FIPS 202 [1]), for each block b, extend b to length $25w$ with zeros, then compute $s = f(s \oplus b)$. From s, take the top m bits as the output of the hash function.

2. Mathematical Properties of the Five Round Functions

In the following section, we detail various mathematical properties of the five permutation functions which make up the core round function of Keccak. In most cases, we seek to provide mathematical proofs of these properties. In all cases, we rely on external code and Boolean Satisfiability for computerized proofs where proofs are not directly provided. We do note the introduction of a cleaner form of χ^{-1} than in existing literature.

Of θ. In this section, we show that θ is bijective, that XOR distributes through θ, give a method for finding the inverse of θ, and give the order of θ.

Let w be a fixed power of 2. Since S_w is of finite size, it suffices to show that, $\forall x, y \in S_w$, $x \neq y \Rightarrow \theta(x) \neq \theta(y)$. Assume the hypothesis: then $x \oplus y \neq 0^{25w}$.

Lemma 2.1.

$$\theta(a) = 0^{25w} \iff a = 0^{25w}$$

Proof. This follows from the definition of θ: note that $\theta(0^{25w}) = 0^{25w}$. If $a \neq 0^{25w}$, then there exists an index set I_1 such that $\forall i \in I_1$, $a[i] = 1$, and $\forall j \notin I_1$, $a[j] = 0$. Then $\theta(a) \neq 0^{25w}$, which follows from the construction of θ. (Note that each output bit of θ is composed of the XOR of 11 values in a fixed pattern).

□

Lemma 2.2. $\forall a, b \in S_w$,

$$\theta(a \oplus b) = \theta(a) \oplus \theta(b)$$

Proof. This follows from the definition of θ: note that θ is composed entirely of XORs and that XOR is commutative and associative.

□

Combining Lemma 2.1 and Lemma 2.2, we have that:

$$\begin{aligned} x \oplus y = 0^{25w} &\iff \theta(x \oplus y) = \theta(0^{25w}) \\ &\iff \theta(x \oplus y) = 0^{25w} \\ &\iff \theta(x) \oplus \theta(y) = 0^{25w} \end{aligned}$$

and hence θ is bijective.

To construct the inverse of θ, note that $A'[x, y, z] = A[x, y, z] \oplus D[x, z]$; hence, $A[x, y, z] = A'[x, y, z] \oplus D'[x, z]$ for some $D' = D$. Since $D[x, z]$ is composed of several $C[x, z]$, where $C[x, z] = \oplus_{y=0}^{4} A[x, y, z]$, we can similarly define $C'[x, z]$ to be $C'[x, z] = \oplus_{y=0}^{4} A'[x, y, z]$. Then, we expect the inverse of θ to be of a similar form. This reduces to a linear algebra problem over boolean variables. We know that $D'[x, z] = D[x, z]$ in order to recover $A[x, y, z]$. Hence we can represent $D[x, z]$ as strings of length $5 \times w$, where bit $i = z' + 5 \times x'$ is 1 if and only if $C[x', z']$ is used in the construction of $D[x, z]$. Further, we can view each of the $C'[x, z]$ as being the conjunction of three $C[x', z']$ in A; thus these are strings where bit $j = z' + 5 \times x'$ is 1 if and only if $C[x', z']$ is used to construct $C'[x, z]$.

(That is, since $C'[x, z] = \oplus_{y=0}^{4} A'[x, y, z]$, $C'[x, z] = \oplus_{y=0}^{4} (A[x, y, z] \oplus D[x, z])$ and hence $C'[x, z] = \oplus_{y=0}^{4} (A[x, y, z] \oplus C[x', z'] \oplus C[x'', z''])$, and thus $C'[x, z] = C[x, z] \oplus C[x', z'] \oplus C[x'', z'']$, for some x, z, x', z', x'', z'' based on the definition of θ).

Thus, giving each $C'[x, z]$ a constant $c_{x,z}$ for whether it is used in constructing $D'[x, z]$, we can form a system of linear equations and solve for the constants $c_{x,z}$ in each expression. Since there are $5 \times w$ variables and $5 \times w$ equations in each equation for $D'[x, z]$, this can be solved easily, yielding the inverse of θ.

Lastly, we have computed the order of the permutation θ for all w. We reproduce them here without proof; they were found by randomized search, and verified with SAT for $w = 1, 2, 4$ and 8. In general, the order is given by the expression $3 \times w$.

w	Order
1	3
2	6
4	12
8	24
16	48
32	96
64	192

Of ρ***.*** Since ρ is a simple permutation of the location of bits, it holds that $\rho(a \oplus b) = \rho(a) \oplus \rho(b)$.

It is obvious that the order of the ρ permutation is w: this follows from the construction of ρ.

Of π***.*** Since π is a simple permutation of the location of bits, it holds trivially that $\pi(a \oplus b) = \pi(a) \oplus \pi(b)$.

It is obvious that the order of the π permutation is 24: this follows from the construction of π.

Of χ***.*** The order of the χ function is 4, and is independent of the width w. This is because χ is independent of both y and z coordinate. We will show algebraically that the order of χ is 4, however, it has also been verified with SAT. We first present three basic lemmas without proof:

Lemma 2.3.

$$\neg(a \oplus b) = \neg a \oplus b$$

Lemma 2.4.

$$(a \oplus b) \wedge (c \oplus d) = (a \wedge c) \oplus (a \wedge d) \oplus (b \wedge c) \oplus (b \wedge d)$$

Lemma 2.5.

$$\neg(a \oplus b) \wedge (c \oplus d) = (\neg a \oplus b) \wedge (c \oplus d)$$

We define a row of the state cube, A to have elements a_1, a_2, a_3, a_4, and a_5. After applying χ to this row, we define the resulting output row to be b_1, b_2, b_3, b_4, b_5, where:

$$\begin{aligned}
b_1 &= a_1 \oplus (\neg a_2 \wedge a_3) \\
b_2 &= a_2 \oplus (\neg a_3 \wedge a_4) \\
b_3 &= a_3 \oplus (\neg a_4 \wedge a_5) \\
b_4 &= a_4 \oplus (\neg a_5 \wedge a_1) \\
b_5 &= a_5 \oplus (\neg a_1 \wedge a_2)
\end{aligned}$$

And similarly for c_1 ... c_5, d_1 ... d_5, e_1 ... e_5. Thus, to show χ has order 4, it suffices to show that $e_1 = a_1$, (since χ has rotational symmetry; if $e_1 = a_1$, then $e_2 = a_2$ ... $e_5 = a_5$).

Lemma 2.6.

$$\begin{aligned}
c_1 =& b_1 \oplus (\neg b_2 \wedge b_3) \\
=& (a_1 \oplus (\neg a_2 \wedge a_3)) \oplus (\neg(a_2 \oplus (\neg a_3 \wedge a_4)) \wedge (a_3 \oplus (\neg a_4 \wedge a_5))) \\
=& a_1 \oplus (\neg a_2 \wedge a_3) \oplus (\neg a_2 \wedge a_3) \oplus (\neg a_2 \wedge \neg a_4 \wedge a_5) \\
& \oplus (a_3 \wedge \neg a_3 \wedge a_4) \oplus (a_3 \wedge a_4 \wedge \neg a_4 \wedge a_5) \text{ (by Lemma 2.5)} \\
=& a_1 \oplus (\neg a_2 \wedge \neg a_4 \wedge a_5)
\end{aligned}$$

And:

$$\begin{aligned}
c_2 =& a_2 \oplus (\neg a_3 \wedge \neg a_5 \wedge a_1)\\
c_3 =& a_3 \oplus (\neg a_4 \wedge \neg a_1 \wedge a_2)\\
c_4 =& a_4 \oplus (\neg a_5 \wedge \neg a_2 \wedge a_3)\\
c_5 =& a_5 \oplus (\neg a_1 \wedge \neg a_3 \wedge a_4)
\end{aligned}$$

Thus, to show $e_1 = a_1$:

$$\begin{aligned}
e_1 =& d_1 \oplus (\neg d_2 \wedge d_3)\\
=& c_1 \oplus (\neg c_2 \wedge \neg c_4 \wedge c_5)\\
=& (a_1 \oplus (\neg a_2 \wedge \neg a_4 \wedge a_5)) \oplus (\neg(a_2 \oplus (\neg a_3 \wedge \neg a_5 \wedge a_1))\\
& \wedge \neg(a_4 \oplus (\neg a_5 \wedge \neg a_2 \wedge a_3))\\
& \wedge (a_5 \oplus (\neg a_1 \wedge \neg a_3 \wedge a_4)))\\
=& a_1
\end{aligned}$$

Hence χ has order 4 and $\chi^{-1} = \chi^3$. The Keccak reference states: "We refer to [20, Section 6.6.2] for an algorithm for computing the inverse of χ." [6]; this points a reference in J. Daemen's thesis [7]. However, given χ, χ^3 is easy to compute and thus χ^{-1} is also easy to compute. Alternatively, using the above, it is easy to see that:

$$d_1 = a_1 \oplus (\neg a_2 \wedge a_3) \oplus (\neg a_2 \wedge \neg a_4 \wedge a_5)$$

Lastly, note that XOR does not distribute over χ due to the introduction of the and.

Of ι. Note that since ι is an XOR with a fixed value, it is obvious that ι is a bijection: for any w, for any i, and for all $x \in S_w$, $\iota(\iota(x, i), i) = x$, since $x \oplus \iota_i \oplus \iota_i = x$. Hence, ι is its own inverse and hence ι is bijective since the inverse is well defined for all $x \in S_w$.

Lastly, note that it is trivial that the order of the ι permutation is 2 by construction (due to the XOR).

Evaluation of the Orders of Composition of Permutations. In this section, we discuss how the above five permutation functions compose, and the orders of the resulting compositions. We limit our discussion to $w = 1$ for the interests of exhaustive search: 2^{25} is possible on commodity hardware, 2^{50} would require additional resources. In general, we find that these permutations interact non-trivially, resulting in cycles of mixed size. We reproduce these results in the table below:

Function	Order	Fixed Points	Number of Cycles	List of Cycles
θ	3	2097152	2	1, 3
ρ	1	33554432	1	1
π	24	4	8	1, 2, 3, 4, 6, 8, 12, 24
χ	4	32	3	1, 2, 4
$\chi \circ \pi$	17360392635484575518934418947500880 $\approx 2^{113.741}$	3	28	Not Reproduced
$\pi \circ \rho \circ \theta$	24	4	8	1, 2, 3, 4, 6, 8, 12, 24
$\chi \circ \pi \circ \rho \circ \theta$	418144575651966378899040573720 $\approx 2^{98.399}$	3	27	Not Reproduced
r_1	320185339723133697023127516600 $\approx 2^{98.014}$	0	14	Not Reproduced
r_2	$\approx 2^{130.726}$	0	12	Not Reproduced
r_3	$\approx 2^{164.242}$	1	16	Not Reproduced
r_4	$\approx 2^{211.609}$	0	16	Not Reproduced
r_5	$\approx 2^{131.878}$	2	16	Not Reproduced
r_6	$\approx 2^{190.743}$	3	18	Not Reproduced
r_7	$\approx 2^{218.017}$	0	18	Not Reproduced
r_8	$\approx 2^{190.137}$	0	18	Not Reproduced
r_9	$\approx 2^{188.483}$	0	14	Not Reproduced
r_{10}	$\approx 2^{154.432}$	0	19	Not Reproduced
r_{11}	$\approx 2^{223.948}$	1	18	Not Reproduced
r_{12}	$\approx 2^{108.579}$	0	12	Not Reproduced
r_{13}	$\approx 2^{209.785}$	1	18	Not Reproduced
r_{14}	$\approx 2^{170.621}$	0	14	Not Reproduced
r_{15}	$\approx 2^{196.507}$	2	22	Not Reproduced
r_{16}	$\approx 2^{172.643}$	1	16	Not Reproduced
r_{17}	$\approx 2^{221.160}$	1	22	Not Reproduced
r_{18}	$\approx 2^{203.510}$	0	20	Not Reproduced
r_{19}	$\approx 2^{250.748}$	3	20	Not Reproduced
r_{20}	$\approx 2^{183.937}$	1	19	Not Reproduced
r_{21}	$\approx 2^{158.852}$	2	15	Not Reproduced
r_{22}	$\approx 2^{111.874}$	1	12	Not Reproduced
r_{23}	$\approx 2^{230.807}$	0	24	Not Reproduced
r_{24}	$\approx 2^{140.355}$	1	14	Not Reproduced

Note that, for $w = 1$, ρ is the identity function and is thus omitted from the tables above. Note that r_n denotes the first n rounds of SHA-3.

While on first glance, large orders make it appear that the hash function is strong, SHA-3 also defines an XOF (or eXtensible Output Function) construct. This allows SHA-3 to act as a pseudo-random number generator: after hashing an input, the internal state of SHA-3 has reached some value S. Suppose we want to request m bits of state. If k exceeds our security margin $k/2$, we take $k/2$ bits, permute $S \to S'$ according to KECCAK-f, and repeat until all m bits have been retrieved.

An ideal permutation function for an XOF would ensure that there exists one random cycle through all possible states (and thus contain a cycle of 2^{25}, in this case). However, the order of the r_n rounds of SHA-3 far exceed 2^{25} and all of them have several cycles. This suggests that the actual security margin of the XOF construct is far less than the theoretical value. However, this analysis needs to be extended to at least $w = 4$ (performing 2^{100} iterations of the hash function core which is not feasible) to fully verify this.

3. Exhaustive Collision Searches

By modeling the problem with SAT, we were able to recreate the work of prior authors ([3], [4]). However, like previous authors, our results do not scale to useful results. For $w = 1$, we have exhaustively searched the collision space; the results of this search are reproduced below:

w	r	Number of collisions
1	1	1024
1	2	502
1	3	543
1	4	525
1	5	488
1	6	518
1	7	532
1	8	498
1	9	506
1	10	522
1	11	506
1	12	503
1	13	495
1	14	522
1	15	485
1	16	540
1	17	467
1	18	490
2	1	$> 2^{20}$
2	2	$> 2^{14}$
2	3	> 74

Note that the results for $w = 2$ are incomplete; the search was terminated after one week. Further, all of these used an effective margin of 512-bits. In our limited time, we were unable to find any useful patterns in this data.

4. Correlation Matrices

Another area of study was the resistance of Keccak to correlation attacks. That is, given some structured input (in this case, a valid block where the remaining bits are all zeros), does SHA-3 emit any useful two bit correlations across rounds? Here, we constructed a program to generate matricies of correlation for all two-variable boolean functions given some state. We then performed Monte-Carlo simulation; for $w = 8$ and $r = 24$, we found no useful correlations under sufficiently large starting states ($>$ 16 million). Thus, Keccak is secure from cross-round first and second order correlations; all distinguisher attacks must thus rely on at least third-order correlations. For $w = 8$, checking all third-order correlations across 24 rounds becomes computationally infeasible and thus was not performed, since it is not likely to yeild useful results.

5. Fixed Point Attacks

In this section, we introduce two potential attacks against Keccak/SHA-3 using fixed points in the underlying round function. However, neither of these attacks have been proven possible with current analysis except for small values of w and small numbers of rounds. The first is a possible attack using full fixed points in the core round function, while the latter describes a category of partial fixed points.

Full Fixed Points. One theoretical attack against SHA-3 is by using a fixed point. If there existed a state x such that $h(x) = x$, for h the round function, then for all additional blocks, $h(x|b) = h(x \oplus b)$. While fixed points can and do occur (see 24-rounds in Table 2), using a fixed-point attack is more subtle in practice: it is unlikely that a fixed point is a valid block (that is, x contains a suffix that is 0^m for the current margin, m). Thus one must find a series of blocks b_1, b_2 such that $h(b_1) \oplus b_2 = x$;

this in turn extends the attack from being $h(x|b) = h(x \oplus b)$ to:

$$\begin{aligned} x = h(h(b_1) \oplus b_2) &\Rightarrow \\ h(x|b) = h(b_1|b_2|b) &= h(b_1|h_2 \oplus b) \\ &= h(b_1|h_2|0^{25w}|b) \end{aligned}$$

However finding b_1, b_2 amounts to a preimage attack and is thus unlikely to occur.

For $w = 1, 2$, we have verified that no fixed points occur which are valid blocks for $1 \leq r \leq 24$, where r is the number of rounds, at margins of 4 and 8 bits.

Partial Fixed Points. Another theoretical attack is using a partial fixed point. Suppose there exists a block x such that $h(x) = y$ is also a block (that is, y has a suffix of 0^m). In this case, the following is a valid collision, for all blocks b:

$$h(x \oplus b) = h(y|b)$$

We have verified the existance of these partial fixed points for small w and number of rounds. Furthermore, they are more readily found using SAT than finding a fixed point is.

For $w = 1$, we reproduce a table of quantities of partial fixed points per round below for the first 8 rounds:

Rounds	Quantity	Input	Output
1	512	FFFFFFFFFFFFFFFFFFFFFFFFF	TFFFFFFFFFFFFFFFFFFFFFFFF
2	567	FTFTTTTFFTTTFFTFTFFFFFFFF	TTTFTFFTFTTTFTFFTFFFFFFFF
3	527	FFTFFFFTFTFFFFFFFFFFFFFFF	TTTFTTTTFTFTFTFTFFFFFFFFF
4	545	TTTFFTTTFFTTFTTFFFFFFFFFF	FFFTTTTFTTFFFTFTFFFFFFFFF
5	488	FFFFFFFFTFFFTFFFFFFFFFFFF	TFFTFFTFFFFFFTFFTFFFFFFFF
6	533	FFTFFFFTTFFFTFFFFFFFFFFFF	TTTFFFFTFFFFTTFFTFFFFFFFF
7	510	TFFFTTTFTTTTFFTFFFFFFFFFF	FTFFFFFFFTFTTTFTTFFFFFFFF

Note that the above table is only for an effective margin of 256-bits (4 bits when $w = 1$). For an effective margin of 512-bits (8 bits), we found no such partial fixed points.

However, brute forcing a partial fixed point for larger values of w and r quickly grows impossible. A similar technique to the aforementioned differential matrices could extend these techniques to larger values of w and r. Instead of building a model to check for a specific number of differences, build a model to check for a specific number of zeros in the inputs and outputs. This could provide a reduction in number of search paths for the SAT solver, and allow for more explicit parallelism.

6. Conclusions and Further Work

Overall, Keccak/SHA-3 remains secure to many attacks from SAT solvers. The majority of the techniques discussed here do not scale to the full $w = 64$, rendering them ineffective for attacking SHA-3 in general. More efficient techniques are thus necessary for finding collisions in SHA-3; this likely involves finding ways in which structure in small values of w translate to structure in larger values of w.

Future work includes scaling the partial fixed point attacks via intermediate models, evaluating weaknesses in alternate constructions, and improving the run time of marginal and differential analysis. However, without significant breakthroughs in logical cryptanalysis, most of these attacks are unlikely. Further, it is not immediately obvious what effect low permutation order has on the possibility of collisions; introducing permutation-based cryptanalysis could result in new techniques applicable to SHA-3. Additional work could include automatic lemmaizing of found constraints and including them in future searches.

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. Massacci F (year?) Logical cryptanalysis. http://disi.unitn.it/~massacci/CryptoSAT/.
3. Homsirikamol E, Morawiecki P, Rogawski M, Srebrny M (2012) *Security Margin Evaluation of SHA-3 Contest Finalists through SAT-Based Attacks*, eds. Cortesi A, Chaki N, Saeed K, Wierzchoń S. (Springer Berlin Heidelberg, Berlin, Heidelberg), pp. 56–67. https://eprint.iacr.org/2012/421.pdf.
4. 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.
5. Team K (2008 - 2017) Team keccak | home (online). https://keccak.team/.
6. 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.
7. Daemen J (1995) Cipher and hash function design strategies based on linear and differential cryptanalysis. http://jda.noekeon.org/JDA_Thesis_1995.pdf.