IOWA STATE UNIVERSITY
Department of Computer Science
Department of Mathematics

Undergraduate Honors Research Project
MATH 490 – Independent Study
April 2018
`https://github.com/cipherboy/hash_framework`

Alexander M. Scheel
Advisors: Dr. Eric Davis, Dr. Clifford Bergman

Building a Framework for Logical Cryptanalysis of Hash Functions

1. Introduction

Core to cryptography is the notion of a **cryptographic hash function**. A hash function maps arbitrary length inputs to a fixed digest (see Figure 1). To be considered cryptographically secure, it must be **computationally hard to find** the following:

- **Preimage** – Two input messages which hash to the same output value (for given y, find an x such that h(x) = y).
- **Second Preimage** – An input message different from the given one which hashes to the same value (for a given x, find a y for which h(x) = h(y))
- **Collision** – Any two input messages which hash to the same value (find x and y such that h(x) = h(y))

Much of modern research has focused on finding collisions; however, published literature does not demonstrate the utility of a given attack. Our research seeks to answer the question of, given a collision attack on a hash function, is it possible to use this to attack a specific system, such as Git (source code revision control) or the PKI (public key infrastructure underpinning secure web communication)? In trying to answer this question, we developed new techniques for **logical cryptanalysis**.

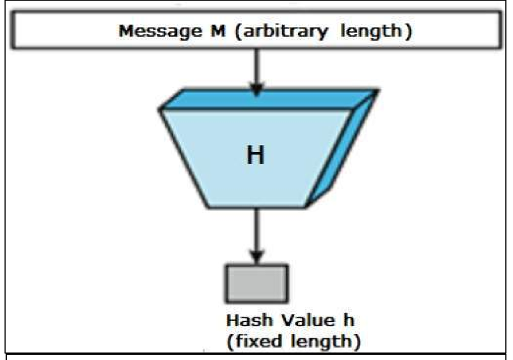

Figure 1: Diagram of a hash function.

2. Framework Architecture

To enable rapid prototyping of new logical cryptanalysis techniques, we built a distributed architecture. This architecture consisted of:

- A PostgreSQL server, Python 3, and a Flask REST API
- Each job was a **Boolean Satisfiability** (3-CNF-SAT) problem in the **circuits** language.
- Techniques were represented as "kernels"
- Most successful techniques was the distance heuristic
  - Strong collisions tended to have immediate neighbors with low distance (creating the ≾ relation)

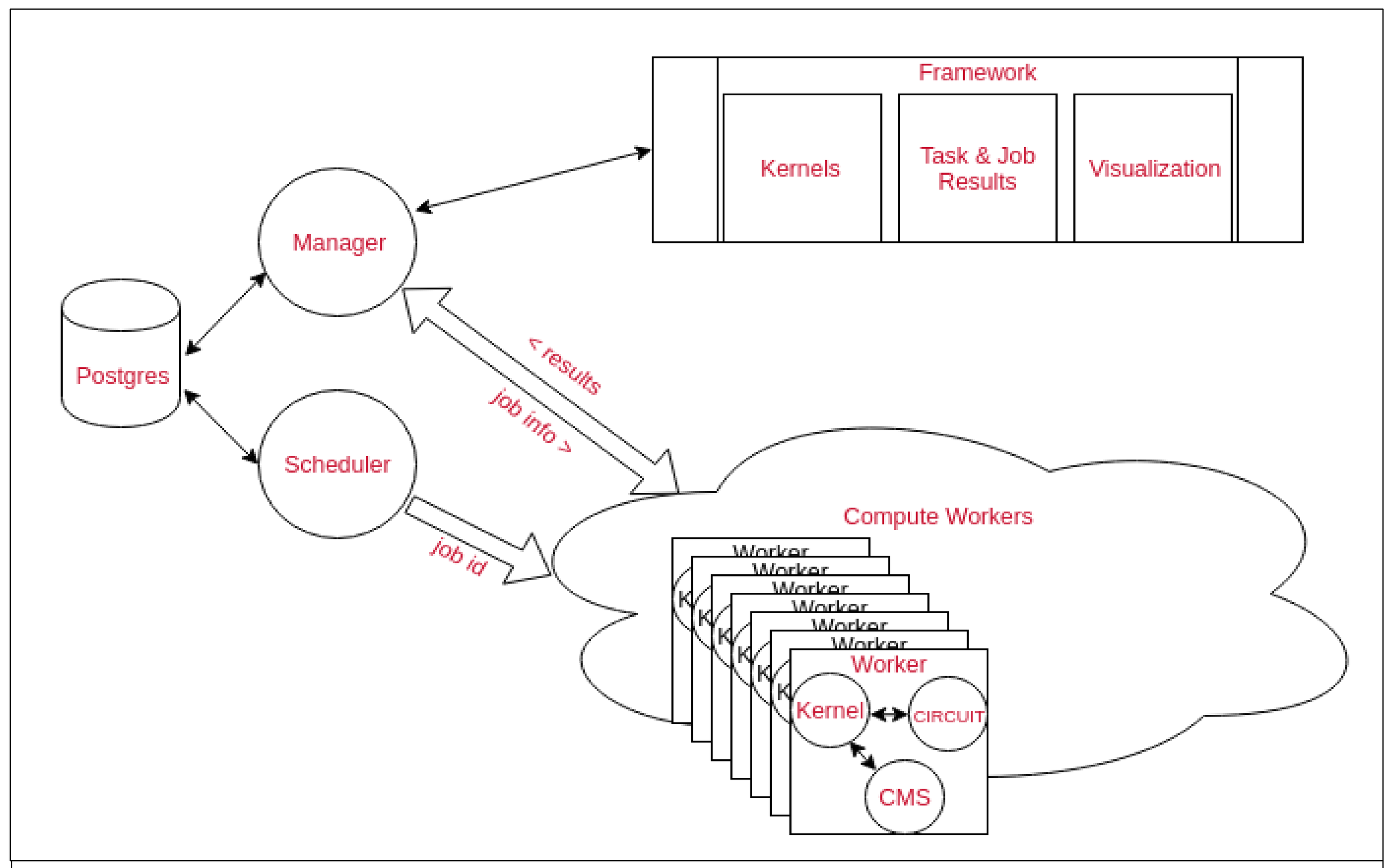

Figure 2: Architecture Overview of `hash_framework` showing communications between workers, scheduler, manager, and framework controller.

3. MD4 Results

Overall, our techniques yielded a 35-fold increase over any previous technique for finding collisions in MD4 (see Figure 3); the work of Martin Schläffer was the second-most productive technique, producing approximately 1,000 collision paths[1]. Figure 4 demonstrates the concept of a **differential path**: the intermediate state variables of two valid collisions form a differential path when xor-ed together. In some instances, a path between two different paths can be formed by changing one round at a time; this is demonstrated by Figure 4 (c).

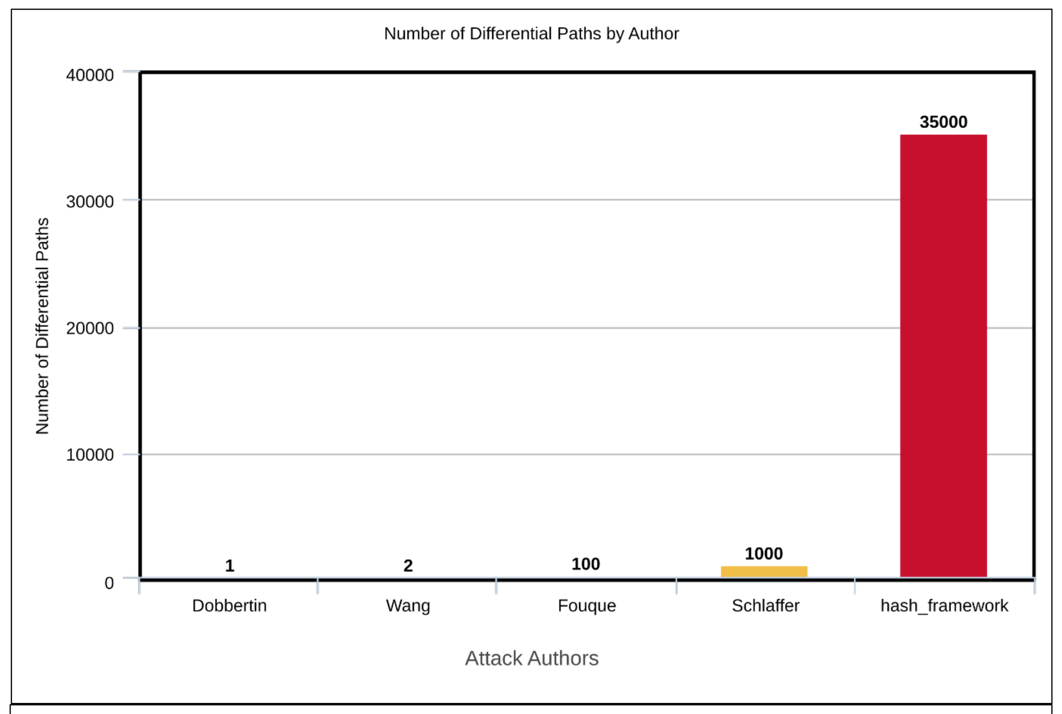

Figure 3: Chart showing the number of differential paths found by various authors; demonstrates a 35-fold increase over previous work.

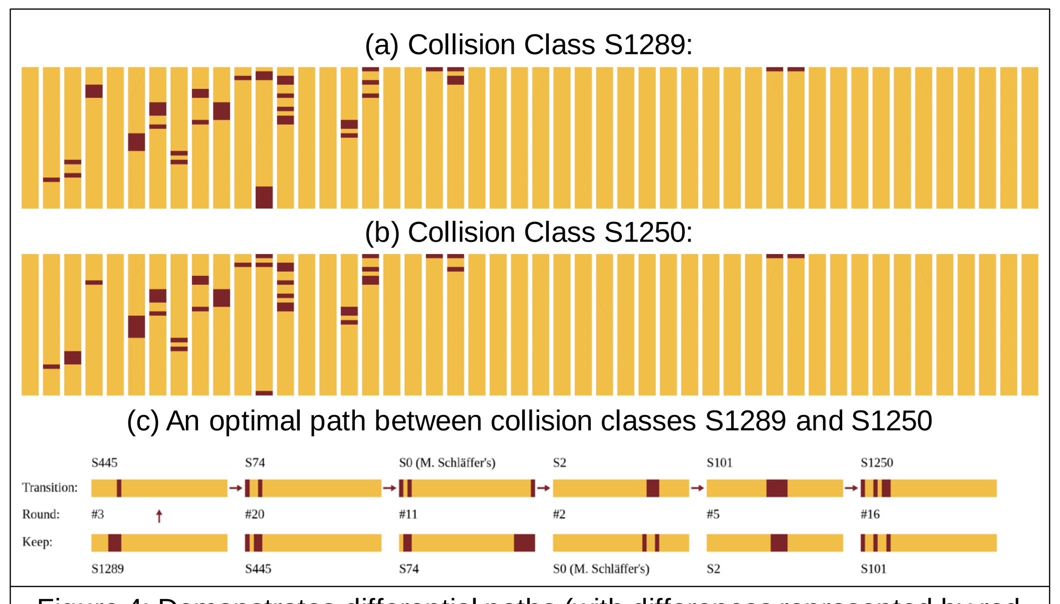

Figure 4: Demonstrates differential paths (with differences represented by red stripes) and a minimal series of one-round changes between them.

4. Order of SHA-3 Permutation Functions

In 2013, NIST published FIPS-202, formalizing Keccak/SHA-3 as the winner of its recent competition. Keccak broke from the tradition of using Merkle-Damgård, and instead uses a sponge construction (See Figure 5) with five permutation functions (θ, ρ, π, χ, ι) composed to form the round function. Due to the introduction of an XOF construct (eXtensible Output Function), the order of these permutations is important: too many cycles of small order will lead to the XOF repeating before its theoretical capacity is reached. Table 1 shows that 24 rounds of SHA-3 has a surprisingly large order with 14 distinct cycle sizes, potentially weakening its security margin.

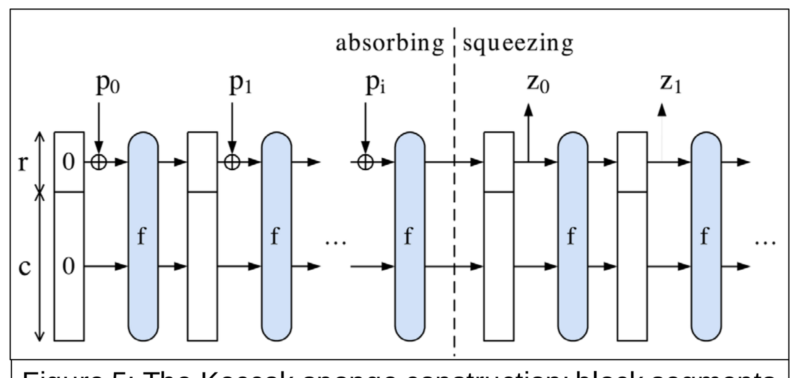

Figure 5: The Keccak sponge construction: block segments ($p_0 \ldots p_i$) are xor-ed in during the absorbing phase and output segments ($z_0 \ldots z_j$) are returned in the squeezing phase.

Function	Order	Cycle Sizes	Fixed Points
θ	3	2	2^{21}
ρ	1	1	2^{25}
π	24	8	4
χ	4	3	32
ι	1	1	0
1 round	$\sim 2^{98}$	14	0
24 rounds	$\sim 2^{140}$	14	1

Table 1: Orders of internal permutations, number of distinct cycle sizes, and number of fixed-points for w=1.

5. Differential and Marginal Properties of SHA-3

To gauge collision resistance, there are three essential properties of SHA-3: the effect of the **input margin** on choices of resulting state differences, the effect of **state differences** on other state differences, and the effect of state differences on the **output margin**. Combining these three allows us to gauge the security margin of SHA-3. Under an ideal permutation function, non-zero input differences would result in non-zero output differences, and increasing the margin would not significantly impact collision resistance. However, the data presents a different story: ρ and π are simple bit location permutations, and θ satisfies that $\theta(x \oplus y) = \theta(x) \oplus \theta(y)$ (note the symmetry in Figure 6 for θ). This symmetry weakens SHA-3 by ultimately improving the chances of attacks: note how, in Figure 8, even-numbers of differences have a larger margin than odd number of differences for low difference counts, and for high input difference counts, the margin is significantly increased. Further, Figure 7 demonstrates that the input margin affects θ by limiting viable output differences.

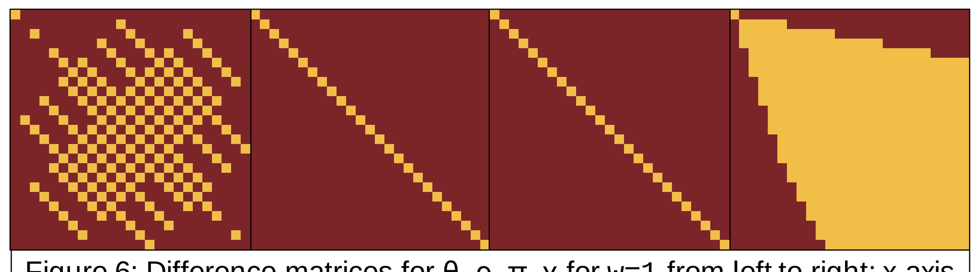

Figure 6: Difference matrices for θ, ρ, π, χ for w=1 from left to right; x-axis is input difference, y-axis is output difference; yellow is satisfiable, red is unsatisfiable.

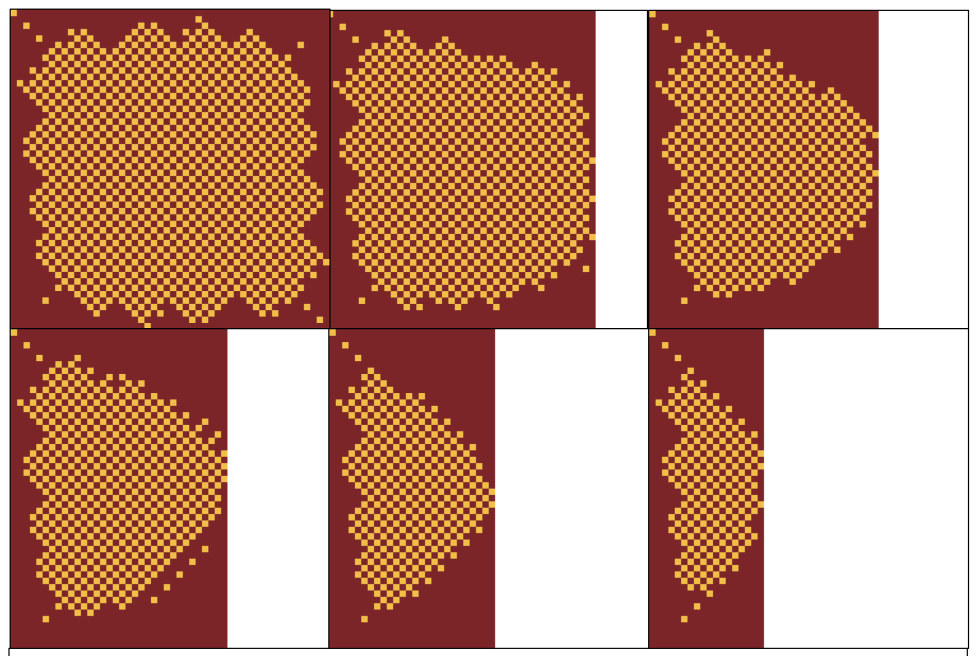

Figure 7: The effect of effective margin on differences for the θ function at w=2: x-axis is input differences, y-axis is output difference; yellow is satisfiable, red is unsatisfiable.

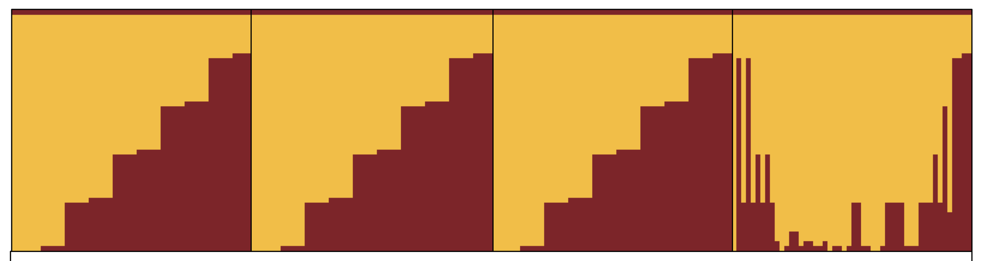

Figure 8: The effect of successive composition on the output margin (χ, π∘χ, ρ∘π∘χ, θ∘ρ∘π∘χ) at w=2: x-axis is input differences, y-axis is size of output margin. Note that for even differences,

6. Conclusion, Future Work, and Open Access

Our works shows that logical cryptanalysis provides a useful foundation for studying hash functions provided that efficient techniques can be created. By splitting up problems into smaller portions, work can be distributed across a heterogeneous compute infrastructure, enabling searching of much larger problem spaces. This enabled us to find a 35-fold increase in MD4 collisions, and begin to study advanced properties of the Keccak/SHA-3 hash function useful for evaluating its collision resistance.

Future work includes developing new techniques for studying other hash functions, meet in the middle techniques for merging partial collisions, and extending this to other problem spaces such as symmetric ciphers and **HMACs** (Hash-Based Message Authentication Codes).

All of our work is publicly available online as a contribution to open source and open access research. Please see the following repositories under the `cipherboy` GitHub account: `hash_framework`, `keccak-attacks`, `papers`, and `talks`.

References/Acknowledgements: Nathan Mulbrook
Figure 1: tutorialspoint.com; Figure 5: wikipedia.com
Footnote 1: According to claims in "Searching for Differential Paths in MD4" – M. Schläffer and E. Oswald