Modern Hash Function Construction

B. Denton[1] **and R. Adhami**[1]
[1]Dept. of Electrical & Computer Engineering, *The University of Alabama in Huntsville*, Huntsville, AL, USA

Abstract- *This paper discusses modern hash function construction using the NIST SHA-3 competition as a survey of modern hash function construction properties. Three primary hash function designs are identified based on the designs of SHA-3 candidates submitted as part of the NIST SHA-3 competition. These designs are Wide-pipe, Sponge, and HAsh Iterated FrAmework (HAIFA).*

Keywords- cryptography; hashing; hash function

1 Introduction

Modern secure hashing algorithms are critically important to the integrity and non-repudiation of information and data in many different computer systems. The most widely used cryptographic hash functions, MD5 and SHA-1, have considerable weaknesses [1,2]. The National Institute of Standards and Technology (NIST) is currently holding an international competition to select the next generation secure hashing algorithm, called SHA-3. This paper covers the construction properties of modern cryptographic hash function as well as the security requirements that motivate these construction properties. After an overview of cryptographic hash function security properties and attacks, we will discuss three primary classifications of modern hash function construction: Wide-pipe, Sponge function, and the HAsh Iterated FrAmework (HAIFA).

2 Hash function security properties and attacks

Cryptographic hash functions play a fundamental role in modern cryptography, specifically in the areas of message authentication, data integrity, digital signatures, and password schemes. In general, a hash function

$$\mathrm{h} : \{0,1\}^{*} \rightarrow \{0,1\}^{\mathrm{n}} \qquad (1)$$

maps an arbitrary finite sized binary input message m to a fixed sized, n-bits, binary output called the *hash value*, *message digest*, or simply *hash*. (Figure 1) For a cryptographic hash functions, the hash value serves as a unique and fixed-sized representation of the unique message input. Unfortunately we have one big problem. Given a domain D and range R with $h : D \rightarrow R$ and $|D| > |R|$ implies that *collisions* are inevitable, where multiple inputs map to the same output. [3]

An acceptable solution is to design a cryptographic hash function (simply hash function from here forth) in such a way that a collision is difficult to find. In other words, finding a collision should be computationally infeasible. In the requirements for NISTS's SHA-3 competition, NIST notes that 2^{80} work is considered too small of a security lower boundary and requires all SHA-3 candidates to have a higher security boundary. [4] This measure is simply the number of calls to the hash function an attacker would have to make in order to find a collision. Although not an ideal definition, computational infeasible today means that more than 2^{80} work is required to find a collision in a hash function.

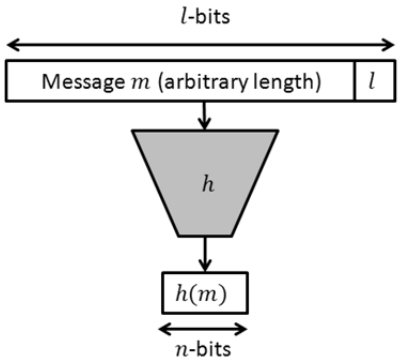

Figure 1 Hash Function

Three distinct attack methods exist for finding a collision in a hash function based on the goals of the attacker. These methods form the fundamental security properties of a hash function. The basic properties of a hash function are:

1. *Preimage resistance* - given an output $h(m)$, it is difficult to find m. See Figure 2.
2. *Second-preimage resistance* - given a specific input m_1 with an output $h(m_1)$, it is difficult to find another input m_2 such that $h(m_1) = h(m_2)$. See Figure 3.
3. *Collision resistance* - it is difficult to find two messages m_1 and m_2 such that $h(m_1) = h(m_2)$. (Note: There is free choice for both messages.) See Figure 4.

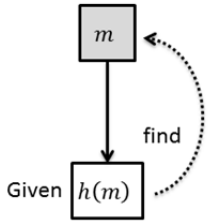

Figure 2 Preimage resistance

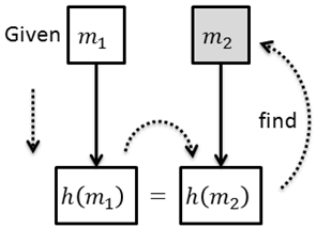

Figure 3 Second-preimage resistance

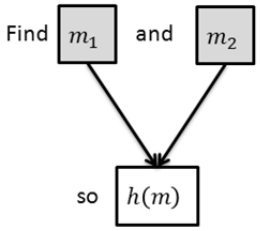

Figure 4 Collision resistance

Preimage resistance has a fixed output value, *Second-preimage resistance* has a fixed input value, and the attacker seeks to find another colliding input message for either of these values. For collision resistance, the attacker does not care what the two input message are, only that they "hash" to the same output.

A common security model used to describe the ideal hash function is the random oracle. Visualize the random oracle as an elf sitting in a box with a source of physical randomness and some means of storage. A common explanation uses dice and a scroll. The elf will accept queries from anyone and will look in the scroll to see if an entry exists for that query. The elf will answer queries from anyone, both friendly and foe. If the query exists, the elf will respond with the recorded result. If the query does not exist, the elf will throw the dice, record the randomized result, and respond. The elf can only work so fast and thus has a limited amount of queries that can be answered every second. The end result is a "perfect" one-way function. [5]

In order for a hash function to have an acceptable security level it should behave like a random oracle. The minimum amount of work required by an attacker to violate the preimage or second-preimage resistance property for an n-bit output hash function should be 2^n. The minimum amount of work to violate the collision resistance property (due to the birthday paradox [6]) should be $2^{n/2}$. [7] For example, SHA-1 has a 160-bit output, so any attack that finds a preimage or second preimage in less than 2^{160} or a collision in less than 2^{80} demonstrates that SHA-1 provides less security than a random oracle. In fact, a collision attack exists against SHA -1 that only requires 2^{69}work [1] which was one of the largest motivating factors for NIST to form the SHA-3 competition to select a new standard hash function.

Table 1 Minimum Security Requirements for a Hash Function

Attack	Security Boundary
Preimage	2^n
Second-Preimage	2^n
Collision	$2^{n/2}$

The hash functions, MD5 and SHA-1, use a classic Merkle-Damgård construction [8,9] which, in general, splits the input message into r equal sized m-bit message blocks $(m_0 \cdots m_{r-1})$ padding the last block as necessary and appending the message length, then iterates through each message block applying a compression function

$$c : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^n. \quad (2)$$

The input to the compression function is the previous compression function output $CV_{i-1}$and the current message block m_i. The compression function output CV_i is called the Chaining Value and the initial chaining value CV_0 is called the Initialization Vector or IV. This process is show in Figure 5 below.

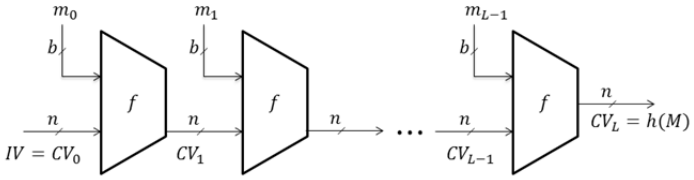

Figure 5 Iterated Hash Function

Unfortunately, due to the iterative nature of classic Merkle-Damgård (MD) construction certain generic attacks exist that differentiate an MD hash function from a random oracle [5] such as the multicollision attack [10] and length extension attacks.

Imagine a collision finding function C that takes an initialization vector $(CV_0 = h_0)$ and outputs two message blocks $m_1 \neq m'_1$ that both hash to the same value

$$h_1 = f(m_0, h_0) = f(m'_0, h_0). \quad (3)$$

The amount of work required to find this collision is $2^{n/2}$. Set $CV_1 = h_1$ and the collision finding function will find two more messages, m_1 and m'_1, that also collide which requires another $2^{n/2}$. Thus for finding k-colliding pairs (Figure 6) of message blocks that all hash to the same final value h_k, the attacker only has to expend $k \times 2^{n/2}$ effort instead of the expected $2^{n(2^k-1)/2^k}$effort if the hash function was truly random. This attack also leads to other serious attacks such as

the long message second-preimage attack [11] and the herding attack [12].

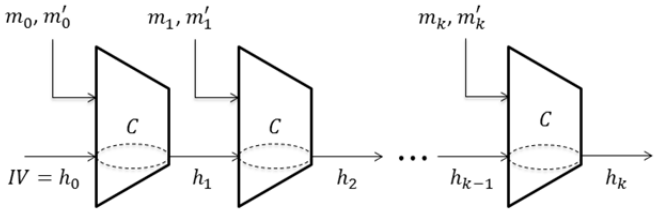

Figure 6 Multicollisions in Merkle-Damgård construction

Another problem with MD based hash functions is length extension. A message m is divided into $m_0, m_1, \cdots, m_k$ blocks and hashed to a value H. Now choose another message m' that divides into $m_0, m_1, \cdots, m_k, m_{k+1}$ blocks. Since m and m' share the same first k blocks, the hash value $h(m)$ is simply the intermediate hash value after k blocks when computing $h(m')$. This is shown in Figure 7 below. This certainly is not a property of the ideal hash function and creates serious problems in practice. [13]

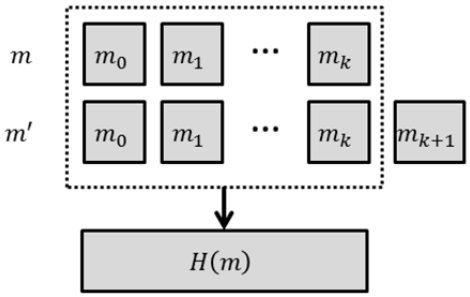

Figure 7 Length Extension

Merkle and Damgård [8,9] proved that a collision resistant compression function implies a collision resistant hash function but due to internal collisions caused by the iterative nature of MD construction, these additional attacks are possible. A random oracle does not have these weaknesses. These generic attacks have motivated cryptographic researchers to find new methods of designing hash functions.

3 State-of-the-art hash function design

The NIST SHA-3 competition began in November 2008 with the first round consisting of 51 candidate hash functions. In August 2009, the second round cut the competition down to 12 candidates. The final round began in December 2010 with 5 finalists. Nearly all these candidate hash functions can be broken down into one of three general categories: wide-pipe, sponge functions, and HAIFA.

3.1 Wide-pipe

Wide-pipe hash function construction is designed to overcome the multicollision attack and length extension attack by trying to prevent internal collisions. [14] In simple terms, this is accomplished by increasing the size of internal state of the hash function. For an n-bit output hash function with a w-bit CV, $w > n$. (Figure 8) In order to obtain the n-bit output, an output transformation is performed on the final CV_{r-1}. This could be just discarding bits or some other more complicated function.

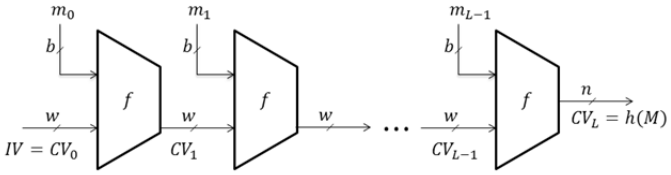

Figure 8 Wide-pipe Construction

3.2 Sponge Functions

The next class of hash function construction, sponge functions [15], was designed in such a way to mimic a random oracle. In general terms, a random oracle

$$R : \{0,1\}^* \rightarrow \{0,1\}^\infty \quad (4)$$

maps a variable-length input message to an infinite output string. It is also completely random; the output bits are uniformly and independently distributed. The requirement of a random oracle that makes it suitable for a proper hash function security model is that identical input messages generate identical output strings. A secure hash function with an n-bit hash value should behave exactly like a random oracle whose output is truncated to n-bits.

Again, due to the iterative nature of classic Merkle-Damgård construction, internal state collisions exist in the chaining values, CV_i. State collisions introduce properties that do not exist for a random oracle such as the length extension attack. For example, consider that M_1 and M_2 are two messages that form a state collision. For any suffix N, the messages $M_1|N$ and $M_2|N$ will have identical hash values. State collisions alone are not a problem but they do lead to a differentiator from a random oracle. The random oracle security model is an unreachable goal for an iterated hash function. Instead of abandoning iterated hash functions, Bertoni et al. [15] designed the sponge function construction as a new security model that is only distinguishable from a random oracle by the presence of internal state collisions.

Sponge construction consists of a fixed-length permutation on a fixed number of bits, $r + c$, shown in Figure 9 below [15]. It can be used to create a function with variable-sized input and arbitrary length output.

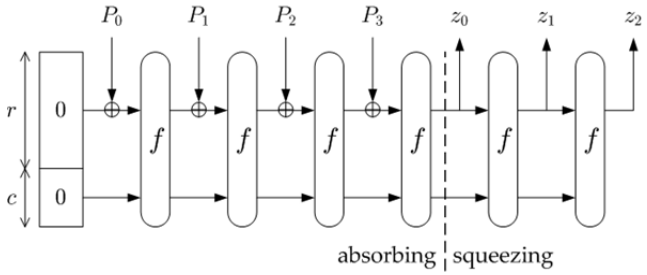

Figure 9 Sponge Construction

The value r is called the bitrate and the value c is the capacity. Both are initially set to zero. The input message is first padded and split into r-bit length message blocks. The sponge construction consists of two stages, absorbing and squeezing. In the absorbing stage, r-bit sized blocks of the input message (P_i in Figure 9) are XORed with the first r-bits of the state, followed by an application of the permutation function f. This is repeated until all message blocks are processed. In the squeezing stage, the first r-bits are returned as output of the sponge construction, again followed by applications of the permutation function f. The length of the squeezing phase is user-specified depending on the desired output length.

3.3 HAIFA

The HAsh Iterative FrAmework (HAIFA) [16] design solves many of the internal collision problems associated with the classic MD construction design by adding a fixed (optional) salt of s-bits along with a (mandatory) counter C_i of t-bits to every message block in the iteration i of the hash function. Wide-pipe and HAIFA are very similar designs. The counter C_i keeps track of the number of message bits hashed so far. The HAIFA design is both prefix- and suffix-free and as a consequence is collision resistant and indifferentiable from a random oracle. [17] This design is also proven secure against 2nd-preimage attacks if the underlying compression function behaves like an ideal primitive. [17] Figure 10 below shows the construction of the compression function. This can also be expressed as

$$CV_i = C(CV_{i-1}, m_i, C_i, S). \quad (5)$$

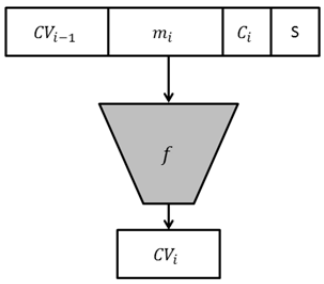

Figure 10 HAIFA Construction

4 Discussion

This paper covered three general secure hash function construction methods represented by various hash functions specifications in the NIST SHA-3 competition. All of these are general descriptions and do not represent an actual hash function. Instead, these construction methods serve the purpose of describing general solutions to some serious attacks against classic Merkle-Damgård hash function construction.

5 References

[1] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu, "Finding Collisions in the Full SHA-1," in *Proceeding of CRYPTO'05*, vol. 3621, 2005, pp. 17-36.

[2] Marc Bevand, "MD5 Chosen-Prefix Collisions on GPUs," in *Black Hat USA*, Las Vegas, 2009.

[3] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, *Handbook of Applied Cryptography*.: CRC Press, 1996.

[4] National Institute of Standards and Technology, "Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family," *Federal Register*, vol. 72, no. 212, pp. 62212-62220, November 2007.

[5] Mihir Bellare and Phillip Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and Communications Security*, 1993, pp. 62-73.

[6] Gideon Yuval, "How to Swindle Rabin," *Cryptologia*, vol. 3, no. 3, pp. 187 - 191, 1979.

[7] William Stallings, *Cryptography and Network Security: Principles and Practices 5th Ed.*, 4th ed.: Prentice Hall, 2010.

[8] Ivan Damgård, "A Design Principal for Hash Functions," in *Proceedings of CRYPTO '89*, vol. 435, 1989, pp. 416-427.

[9] Ralph Merkle, "One Way Hash Functions and DES," in *Proceedings of CRYPTO'89*, vol. 435, 1989, pp. 428-446.

[10] Antoine Joux, "Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions," in *Proceedings of CRYPTO'04*, vol. 3152, 2004, pp. 306-316.

[11] John Kelsey and Bruce Schneier, "Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work," in *Proceedings of EUROCRYPT'05*, vol. 3494, 2005, pp. 474-490.

[12] John Kelsey and Tadayoshi Kohno, "Herding Hash Functions and the Nostradamus Attack," in *Proceedings of EUROCRYPT'06*, vol. 4404, 2006, pp. 183-200.

[13] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno, *Cryptographic Engineering: Design Principles and Practical Applications*.: Wiley Publishing, Inc., 2010.

[14] Steven Lucks, "Design Principles for Iterated Hash Functions," Cryptology ePrint Archive, 2004. [Online]. http://eprint.iacr.org/2004/253

[15] Guido Bertoni, Joan Daemon, Michael Peeters, and Gilles Van Assche, "Sponge Functions," in *ECRYPT Hash Function Workshop*, 2007.

[16] Eli Biham and Orr Dunkelman, "A Framework for Iterative Hash Functions - HAIFA," Cryptology ePrint Archive, 2007. [Online]. http://eprint.iacr.org/2007/278

[17] Andreeva, Elena; Mennink, Bart; Preneel, Bart, "Security Reductions of the Second Round SHA-3 Candidates," Cryptology ePrint Archive, 2010. [Online]. http://eprint.iacr.org/2010/381