$\mathbf{MMH}_{\mathbf{32}}^{*}$ FROM SCRATCH: AN INTRODUCTION TO HASH FUNCTIONS

SPENCER MILLER

Abstract. This paper first provides an introduction to hash functions, both cryptographic and non-cryptographic, based on An Introduction to Cryptography with Coding Theory written by Trappe and Washington[1]. Later, the Carter and Wegman-introduced[4] topic of universal families of hash functions is approached mathematically to show the importance of considering families of functions alongside individual functions. Finally, this paper discusses the computer optimization of a specific Δ-universal family, MMH_{32}^{*}, which was introduced by Halevi and Krawczyk[5], and points the reader toward sources explaining other useful concepts pertaining to hash functions.

Contents

1. Introduction

Hash functions serve as one of the fundamental building blocks in modern computer systems, alongside block ciphers, pseudorandom number generators, and others. In non-cryptographic circumstances, hash functions may be used for purposes such as highly efficient database organization. However, the cryptographic usage of hash functions is where the concept shines. In support roles, hash functions are regularly used to standardize the testing of other cryptographic components to ensure safety and to serve as the source of randomness in cryptographically safe pseudorandom number generators. On their own, hash functions are often used to verify the integrity of data after it is sent through both noisy (randomly data-altering) and insecure (adversary-controlled) communication channels.

This paper focuses on the concept of universal families, sometimes known as classes, of hash functions. These families extend the capacities of hash functions in two ways. Firstly, applications capable of using a random function from the family in each instance gain benefits in efficiency and security proven to be superior to any possible single hash function. Secondly, though the aforementioned benefits

are not *fully* retained when randomly choosing a single function from the, usually large, family and using it, there is a very high likelihood of that function's retaining efficiency and security near the level of the family as a whole.

2. Preliminaries

This section will introduce modular arithmetic and dot products, two tools necessary for understanding the contents of this paper.

Definition 2.1. Let x, y be integers and let m be a nonzero integer called a *modulus*. Then

$$x \equiv y \pmod{m} \tag{2.2}$$

when $(x - y)$ is a multiple of m. (This *congruence* is read as "x is congruent to y, mod m".)

Example 2.3.

$$35 \equiv 0 \pmod{7} \qquad 17 \equiv 15 \pmod{2} \qquad 3 \times 8 \equiv 24 \equiv 4 \pmod{5} \tag{2.4}$$

Modular arithmetic in this paper will consist entirely of *modular reduction*. Modular reduction is the use of some positive modulus m to reduce some large integer x to an element y of $\mathbb{Z}_m = [0, m-1] \subset \mathbb{Z}$, where $x \equiv y \pmod{m}$.

Definition 2.5. For two k-vectors $x = \langle x_1, x_2, \ldots, x_k \rangle, y = \langle y_1, y_2, \ldots, y_k \rangle \in \mathbb{Z}^k$, the *dot product* of x and y is defined as

$$x \cdot y = \sum_{n=1}^{k} x_n y_n \tag{2.6}$$

3. Important Notions for Hash Functions

Definition 3.1. A *hash function* is a function $h : A \to B$ where $A = \{a \in \{0,1\}^j : j \in \mathbb{N}\}$ is the set of all bit sequences of arbitrary length and $B = \{0,1\}^k$ the set of all bit sequences of a specific, generally short, length k. Inputs to hash functions are called *messages* and outputs are called *digests*.

Hash functions' implementations should ideally be fast to run on a computer so as to enable, with as minimal a computational cost as possible, the use of hash functions in the verification of data. Why this is important and how one may go about it is discussed in Section 4. To achieve this efficiency, many hash functions are constructed by combining a fast function which works on a small, finite domain A and a method by which said function may be extended to accept inputs of arbitrary input length. In one such method, called "tree-hashing", we can split a long message into blocks of the right length, find the hashes of the blocks, connect those hashes together to form a new message, and repeat until desired size is reached. This strategy generally allows us to treat hash functions with finite A just the same as those with infinite A, a freedom which will be regularly used in this paper.

In cryptographic circumstances, there exist 3 basic security notions which hash functions are designed to meet. There do exist other notions, but these 3 are the most essential. In all of the following definitions, we use the standard that $m \in A$ is a message to be hashed and $d \in B$ is a digest. All 3 involve *infeasibility*, where a

task's being *infeasible* means that an adversary attempting it would require unrealistic computing power, unrealistic computing memory, or unrealistic mathematical insight in order to accomplish the task.

Definition 3.2. *First preimage resistance*, sometimes shortened to *preimage resistance* or FPR, is a property of a hash function $h : A \to B$ which holds when, for any given $d \in B$, it is infeasible to determine any $m \in A$ such that $h(m) = d$.

Intuitively, FPR means that the function is effectively *one-way*; the process of finding the output from the input is easy while finding an input from an output is infeasible.

Definition 3.3. *Second preimage resistance*, sometimes referred to as *weak collision resistance* or SPR, is a property of a hash function $h : A \to B$ which holds when, for any given $m \in A$, it is infeasible to determine any $m' \in A$ such that $h(m) = h(m')$ and $m \neq m'$.

The task made infeasible by the presence of SPR and that made infeasible by FPR both involve the determination of a message m' which hashes to a given digest. However, SPR's task is altered by the attacker's having an example message m which also does so. This property largely indicates the absence of predictable, exploitable patterns in the set of messages which all hash to the same digest. For clarity, we note that while second preimage resistance does not always imply first preimage resistance, it does imply the decreasing feasibility of first preimage resistance's task when the domain A is of much greater cardinality than the range B. In particular, the cited paper includes a proof not covered here showing that a fully-fledged hash function with infinite A and finite B does indeed have first preimage resistance any time it has second preimage resistance.[2]

Definition 3.4. *Strong collision resistance* is a property of a hash function $h : A \to B$ which holds when it is infeasible to determine any $m \neq m' \in A$ such that $h(m) = h(m')$.

Strong collision resistance very generally states that given only the definition of the hash function h, it is infeasible to find *any* pair of messages which hash to the same digest. While second preimage resistance only guarantees security for a randomly chosen $d \in B$, strong collision resistance guarantees it for any choice of d at the attacker's discretion.

Strong collision resistance always implies SPR, hence SPR's alternate name: weak collision resistance. This implication exists because, if SPR were broken with a feasible method, then one could immediately break strong collision resistance by choosing an arbitrary m, calculating $h(m)$, and using the aforementioned method to find a distinct m' where $h(m) = h(m')$.

Before we continue into the specific notions discussed by this paper, we ought to give a short, informal description of the *random oracle*, the theoretically ideal hash function.

The random oracle hash function behaves as follows. When the random oracle function $R : A \to B$ is called for message $m \in A$, it first checks whether that specific m has ever been inputted into the function before; the function is not defined for an m until the $R(m)$ is called. If not, the function chooses a perfectly random $d \in B$ and stores the information that m has been inputted and that the generated digest was d. If the function has seen m before, it retrieves the same d which it output

the first time and outputs it. In other words, the function's outputs are perfectly random except that the same input will always give the same output. The random oracle function has all three of the above desired properties alongside a wide variety of less common but still useful security guarantees.[3] Though this function serves as an excellent ideal function in terms of security and is used extensively in theoretical and testing circumstances, the function is extremely computationally intensive and virtually impossible to secretly share between individuals. Additionally, it has been proven that no procedural hash function can perfectly mimic the random and secure behavior of the random oracle.[3]

4. Universal Families of Hash Functions

The rest of the paper will discuss collections of hash functions called *families* which are analyzed collectively. Every function in a family of hash function has the same domain and the same range. The functions often share a basic structure and are differentiated only by constants. From here on, any family H of hash functions will be referred to as $H : A \to B$ where A is the domain and B is the range of all member functions, as these are the only necessarily shared attributes between all members of a family. Analysis of hash functions as families and especially as *universal* families was introduced by mathematicians Carter and Wegman.[4]

To define a *universal* family of hash functions we must first describe the probability notation over families of hash functions which we will be using in many of the definitions and theorems moving forward. Consider the family of hash functions $H : A \to B$, a fixed $x \in A$ and $y \in B$, and the equation $Pr_{h\in H}[h(x) = y] \leq z$. The inequality can be read as "for a randomly chosen $h \in H$, the probability that $h(x) = y$ is less than or equal to z." More formally, if we let $|h(x) = y| = 1$ when $h(x) = y$ and $|h(x) = y| = 0$ otherwise, and let $|H|$ be the cardinality of the family H, then our inequality is equivalent to the statement

$$\frac{\sum_{h\in H} |h(x) = y|}{|H|} \leq z.$$

Definition 4.1. $H : A \to B$ is a *universal* family of hash functions if, for all $x \neq y \in A$, $Pr_{h\in H}[h(x) = h(y)] = \frac{1}{|B|}$.

$H : A \to B$ is an ϵ-*almost-universal* (ϵ-AU) family of hash functions if, for all $x \neq y \in A$, $Pr_{h\in H}[h(x) = h(y)] \leq \epsilon$.

Intuitively, universality and ϵ-almost-universality mean that, on average, a pair of specific inputs to our family of hash functions collide in at most some low fraction of the functions in the family. The pure, first version of universality has this fraction as $\frac{1}{|B|}$, the lowest possible value for the average, which can be found by investigating the ideal random oracle function discussed earlier.

Definition 4.2. Assume B is an Abelian group and let '$-$' be the group subtraction operation. $H : A \to B$ is a Δ-*universal family of hash functions* if, for all $x \neq y \in A$ and all $b \in B$, we have that $Pr_{h\in H}[h(x) - h(y) = b] = \frac{1}{|B|}$.

$H : A \to B$ is an ϵ-*almost*-Δ-*universal* (ϵ-AΔU) family of hash functions if, for all $x \neq y \in A$ and all $b \in B$, we have that $Pr_{h\in H}[h(x) - h(y) = b] \leq \epsilon$.

Δ-universality means that the domain of the family of hash functions has no pair of messages that disproportionally (above a minimal or given value) *distance-collide*. Distance-collision occurs where two messages have an uneven probability

distribution in the distances between their digests across a family of functions. We note that since collision is the special case of this, with distance $b = 0$, Δ-universality is a strictly stronger version of the universality of the last definition.

Definition 4.3. $H : A \to B$ is a *strongly universal* family of hash functions if, for all $x \neq y \in A$ and all $r, s \in B$, we have that $Pr_{h \in H}[h(x) = r, h(y) = s] = \frac{1}{|B|^2}$.

$H : A \to B$ is an ϵ *-almost-strongly universal* (ϵ-ASU) family of hash functions if, for all $x \neq y \in A$ and all $r, s \in B$, we have that $Pr_{h \in H}[h(x) = r, h(y) = s] \leq \frac{\epsilon}{|B|}$.

Strong universality generalizes universality beyond specific relationships between pairs of messages. Whereas regular universality ensures no inordinately frequent collisions between pairs of messages and Δ-universality ensures no inordinately frequent distances between those pairs, strong universality ensures that no pairs of messages have any relationship in their digests whatsoever. Specifically, a message x hashing to r gives no evidence whatsoever that a message y will hash to any specific s. ϵ-ASU is similarly general, but establishes an upper bound on how *much* evidence is provided, rather than establishing there will be none at all.

Now that we have established the basic definitions and concepts pertaining to both hash functions and families of hash functions, we move on to show how families of hash functions might be leveraged beyond single hash functions. To do this, we first set a bound on the capacities of any lone hash function.

Theorem 4.4. *Given any family $H : A \to B$ of hash functions with finite A, there exist $x, y \in A$ such that $Pr_{h \in H}[h(x) = h(y)] > \frac{1}{|B|} - \frac{1}{|A|}$.*

Proof. This proof will first describe the behavior of a single function in the collection H and then extend this behavior to the collection as a whole.

Consider some fixed $h \in H$. For each $b \in B$, let A_b be the preimage of $\{b\} \subseteq B$. Note that for $b \neq b' \in B$, we have that $Pr_{a_b \in A_b, a_{b'} \in A_{b'}}[h(a_b) = h(a_{b'})] = 0$ and also that $Pr_{a_{b_0} \neq a_{b_1} \in A_b}[h(a_{b_0}) = h(a_{b_1})] = 1$. This second finding means that every element in some A_b collides with every other element in A_b by definition. Because of this, we can say that the number of ordered colliding pairs landing on $b \in B$ is $|A_b|(|A_b| - 1) = |A_b|^2 - |A_b|$. Summing these values over all $b \in B$, we find that the number of ordered colliding pairs over all of A is equal to $\Sigma_{b \in B}\left[|A_b|^2 - |A_b|\right]$.

The value of this summation is minimized in the case where all collision-sets A_b are of equal cardinality: $\frac{|A|}{|B|}$. This is because the sum of the squares of n numbers is always greater than or equal to n times their average's square; this can be shown by induction from $n = 1$. As our summation is primarily such a sum of squares, we know that it is minimized in the case where all A_b are as near as possible to their average, $\frac{|A|}{|B|}$. In this minimized-value case, then, the number of ordered colliding pairs in A is equal to $|B|\left[\frac{|A|^2}{|B|^2} - \frac{|A|}{|B|}\right] = |A|^2\left[\frac{1}{|B|} - \frac{1}{|A|}\right]$. As an aside, this statement can be rewritten as the fact that $Pr_{x,y \in A}[h(x) = h(y)] > \left[\frac{1}{|B|} - \frac{1}{|A|}\right]$, which we will utilize later in the proof of Theorem 4.9.

Continuing with this proof, however, we instead multiply our lower bound on the number of total collisions in our single function by the cardinality of H to find the lower bound $|H||A|^2\left[\frac{1}{|B|} - \frac{1}{|A|}\right]$ on the total number of collisions in the entire collection of functions $H : A \to B$.

Consider the number of ordered pairs of values $a \neq a' \in A$, the maximum number of pairs which can possibly collide in any function in H. As $a \neq a'$, this number is at

most $|A|(|A|-1)$. As this is less than the $|A|^2$ term in our count of minimum total collisions, the pigeonhole principle gives that there exists at least one ordered pair such that the number of collisions between those terms is at least $|H|\left(\frac{1}{|B|}-\frac{1}{|A|}\right)$.

Therefore, $Pr_{h\in H}\left[h(a)=h(a')\right] > \left(\frac{1}{|B|}-\frac{1}{|A|}\right)$. □

Intuitively, what this theorem shows is that a family of hash functions will always have at least one pair of messages that collides at a frequency determined by the cardinalities of the domain and range in question. Usefully, this particular theorem does not require universality in the family H and is instead true of any family of hash functions whatsoever. We may note, though we do not prove, that this theorem behaves as would be expected when A is infinite, reducing the $\frac{1}{|A|}$ term to zero.

The theorems for the remainder of this section all rely on an example system which uses a theoretical hash function for the storage of associative data. Associative data storage is data storage where the information is stored in the form (k, j), where the later input of a specific k will output the corresponding j. (k, j) in this type of system is called a key-value pair.

The theoretical system, based on some hash function h, which we will use for our analysis works as follows: construct an array with each value in the array's being a sub-array. The outer array should be of cardinality $|B|$. When a key-value pair (k, j) is stored, it is stored, as (k, j), within the sub-array indexed by $h(k)$. When the key k is later queried, $h(k)$ is calculated and the corresponding sub-array is linearly searched for the correct key-value pair, from which the value is retrieved.

We define the cost function $C(h, T)$ as the number of key-value pairs accessed during the lifetime of a storage system organized by hash function $h : A \to B$ and containing keys $T \subset A$.

Theorem 4.5. *Let x be any element of A and S any subset of A. Let h be a randomly chosen function from a universal family of functions $H : A \to B$. Then $Pr_{h\in H}\left[h(x) \in h(S)\right] \leq \frac{|S|}{|B|}$ where $h(S)$ is the image of S through function h.*

Proof. First, we break the probabilistic statement in this theorem as far into its constituent specifications as possible:

(4.6)
$$Pr_{h\in H}\left[h(x) \in h(S)\right] = \Sigma_{h\in H} Pr\left[h(x) \in h(S)\right] \leq \Sigma_{h\in H, y\in S} Pr\left[h(x) = h(y)\right].$$

As H is *universal*, we have that $Pr_{h\in H}\left[h(x) = h(y)\right] = \frac{1}{|B|}$ for any $y \in S \subseteq B$. As such, the prior chain of relations can be extended with

$$\Sigma_{h\in H, y\in S} Pr\left[h(x) = h(y)\right] = \Sigma_{y\in S} \frac{1}{|B|} = \frac{|S|}{|B|}. \tag{4.7}$$

□

This theorem puts an upper bound on the collisions of a fixed value in A with any random subset of A. In terms of our data-storage system, this can be interpreted where x is the key being queried and searched for and where S is the set of all keys already inputted other than x. In this interpretation, this theorem puts an upper bound on the probability that the sub-array corresponding to $h(x)$ will have any key-value pairs other than x's. Finally, it is important to remember that this upper

bound on the probability is averaging not only over $x \in A$ and $S \subset A$, but also all of the functions $h \in H$.

The following theorem extends this upper bound's implications to the long-term operation of our data-storage system where R represents the entire sequence of insertions into the system or requests for data from it and k is the number of these which are insertions; another way to think of k is as the maximum number of data pairs stored in the system at any given time.

Theorem 4.8. *For the R, k, C as defined above, $Pr_{h\in H}[C(h,R)] \leq |R|(1+\frac{|k|}{|B|})$.*

Proof. Theorem 4.5 gave us an upper bound on the expected number of collisions in our data system during any given search: $\frac{k}{|B|}$. As the searched-for data will always be in the sub-array we search, the expected number of pairs to be searched there has an upper bound of $(1+\frac{k}{|B|})$. By multiplying this upper bound by the total number of searches to the system, $|R|$, we finalize the above expression for an upper bound on the total cost of our data-storage system over its lifetime. □

By leveraging the prior theorem's upper bound on the likelihood of multiple key-value pairs' being stored in the same sub-array, this theorem puts an upper bound on the possible average search-duration in the lifetime of our data storage system.

Theorem 4.9. *Given any single hash function $f : A \to B$, not necessarily in a family, let E_f be the expected cost with respect to that function of a random request after k random insertions into the storage system have been made. Let E_H be the expected cost (averaging over a universal family $H : A \to B$ of hash functions) of any request after any k insertions into the storage system have been made. Then $E_f \geq (1-\epsilon)E_H$ where $\epsilon = \frac{|B|}{|A|}$.*

Proof. Let S be the subset of domain A which were inserted into the data-storage system prior to a request on the element $x \in A$. Theorem 4.5 implies that $E_H \leq 1+\frac{|S|}{|B|}$ where we have a cost of 1 to read the element we look for and a cost of $\frac{|S|}{|B|}$ to read any collisions. We will next show that $E_f \geq 1+|S|\left(\frac{1}{|B|}-\frac{1}{|A|}\right)$.

From here on we deal with any lone function f and its expected cost E_f. In the proof of Theorem 4.4, we found that $Pr_{x,y\in A}[f(x)=f(y)] \geq \left(\frac{1}{|B|}-\frac{1}{|A|}\right)$ for any arbitrary hash function such as f. Now we extend this lower bound on the probability of a collision of x and y to the collision of x with some arbitrary subset $S \subseteq A$. To do this, we sum the smaller expected values for collisions with specific $y \in S$:

$$\Sigma_{y\in S} Pr[f(x)=f(y)] \geq \Sigma_{y\in S}\left(\frac{1}{|B|}-\frac{1}{|A|}\right) = |S|\left(\frac{1}{|B|}-\frac{1}{|A|}\right). \tag{4.10}$$

Adapting this to our cost metric in our linear search scheme by adding one, then, we can complete the following chain of relations:

$$\begin{aligned} E_f \geq 1+|S|\left(\frac{1}{|B|}-\frac{1}{|A|}\right) &\geq \frac{|A|-|B|}{|A|}+|S|\left(\frac{1}{|B|}-\frac{1}{|A|}\right) \\ &= \left(1-\frac{|B|}{|A|}\right)\left(1+\frac{|S|}{|B|}\right) \geq \left(1-\frac{|B|}{|A|}\right)E_H. \end{aligned} \tag{4.11}$$

Note that this proof assumes that all costs are nonnegative, which is provided by our definition of the cost metric. □

This theorem shows that the average efficiency of a universal family of hash functions can be more efficient than that of any lone hash function, where efficiency is the minimization of the number of reads to the storage system described above. Though this theorem, then, is specific to the storage system established here, we note that a similar benefit exists in virtually all use cases of hash functions.

5. Fast, Secure Message Authentication with MMH_{32}

In this section we approach the problem of finding an ϵ-almost-Δ-universal family of hash functions which can be run very fast on modern computers. The desire for speed is general, but the application which motivates this search in particular is that of message authentication for large multimedia applications, such as video streaming. The message authentication usage of hash functions ensures that a packet of sent data is uncorrupted by the journey between the computers involved. To accomplish this, the sender and receiver share a secret hash function $h : A \to B$ chosen randomly from a family H. When large data packet, or message, m is sent, it is sent as the pair $(m, h(m))$. When information is received as (m', d'), the recipient checks whether $h(m') = d'$. Random data corruption is exceedingly unlikely to generate a pair which fulfills this due to the sheer size of B. Corruption by an attacker would also fail, as the function h must be known in order to feasibly generate a pair for which $h(m') = d'$. As such, when this condition is met, the recipient can, with a high degree of certainly, know that the data is uncorrupted and sent from the correct source.

The family MMH^*_{32}, constructed by Halevi and Krawczyk,[5] which we construct in this section is based upon a Δ-universal family called MMH^*, which was itself constructed by Carter and Wegman.[4] The Δ-universal hash family MMH^* is defined as follows:

Definition 5.1. Let p be some prime, k some positive integer, and $\mathbb{Z}_p^k$ the set of all k-vectors with elements chosen from the finite $\mathbb{Z}_p$. The family $MMH^* : \mathbb{Z}_p^k \to \mathbb{Z}_p$ is defined as $MMH^* = \{g_{\mathbf{x}} : \mathbb{Z}_p^k \to \mathbb{Z} | \mathbf{x} \in \mathbb{Z}_{\mathbf{p}}^{\mathbf{k}}\}$, where functions $g_{\mathbf{x}}$ are themselves defined such that for $\mathbf{x} = \langle \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k} \rangle, \mathbf{m} = \langle \mathbf{m_1}, \mathbf{m_2}, \ldots, \mathbf{m_k} \rangle$, $\mathbf{x}, \mathbf{m} \in \mathbb{Z}_{\mathbf{p}}^{\mathbf{k}}$:

$$g_{\mathbf{x}}(\mathbf{m}) = \mathbf{m} \cdot \mathbf{x} \pmod{\mathbf{p}} = \sum_{\mathbf{i=1}}^{\mathbf{k}} \mathbf{m_i x_i} \pmod{\mathbf{p}}. \tag{5.2}$$

In plainer terms, this family of functions operates by taking the dot-product $\cdot$ of some k-vector over the finite field and our message which is also a k-vector over the same field. The result is taken mod p, then outputted.

Theorem 5.3. *The family MMH^* is Δ-universal.*

Proof. Fix some $x \in \mathbb{Z}_p^k$, let $m, m' \in \mathbb{Z}_p^k$ be two distinct messages, and let $a \in \mathbb{Z}_p$ be any given possible digest. Assume without loss of generality that $m_1 \neq m'_1$. Then, we have that

$$\begin{aligned} &Pr_{x_1}\left[g_{\mathbf{x}}(\mathbf{m}) - \mathbf{g_x}(\mathbf{m'}) \equiv \mathbf{a} \pmod{\mathbf{p}}\right] \\ &\qquad = Pr_{x_1}\left[(m_1 - m'_1)x_1 \equiv a - \Sigma_{i=2}^{k}(m_i - m'_i)x_i \pmod{p}\right] = \frac{1}{p} \end{aligned} \tag{5.4}$$

This final equality emerges from the fact that each side of the previous term's congruence has an equal probability of being congruent to any specific term in $\mathbb{Z}_p$. As such, the likelihood of their being congruent to one another is $\frac{1}{p}$. □

We might note that $\frac{1}{p}$ isn't actually that low a probability for guessing the digest of a random message, but by hashing with two different functions in the family and putting the digests one after the other, we can get $\frac{1}{p^2}$ as the probability; this is exponentially better.

We now move on to describe the Halevi/Krawczyk variation on MMH^* which they call MMH^*_{32}. The objective of their changes is to utilize the 64-bit nature of modern computer CPUs and a trick called "divisionless modular reduction" to decrease the computational load the original MMH^* imposed on processors. This trick necessitates the use of prime p such that $2^{32} < p < 2^{32} + 2^{16}$ rather than the arbitrarily-chosen primes usable for MMH^*. Here the authors use the smallest such p: $p = 2^{32} + 15$.

Definition 5.5. $MMH^*_{32} = \{g_{\mathbf{x}} : \left(\{0,1\}^{32}\right)^k \to \mathbb{Z}_p | \mathbf{x} \in \left(\{\mathbf{0},\mathbf{1}\}^{\mathbf{32}}\right)^{\mathbf{k}}\}$ where functions $g_{\mathbf{x}}$ are defined such that for $\mathbf{x} = \langle \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_k} \rangle, \mathbf{m} = \langle \mathbf{m_1}, \mathbf{m_2}, \ldots, \mathbf{m_k} \rangle$, we have that

$$g_{\mathbf{x}}(\mathbf{m}) = \mathbf{m} \cdot \mathbf{x} \pmod{2^{32}+15} = \left[\mathbf{\Sigma_{i=1}^{k}} \mathbf{m_i x_i}\right] \pmod{2^{32}+15}. \tag{5.6}$$

Theorem 5.7. *The family MMH^*_{32} is ϵ-$A\Delta U$ with $\epsilon = 2^{-32}$.*

Proof. This proof is identical, except for the condition that there are 2^{32} possible messages rather than p, to that of the proof of MMH^*'s Δ-universality. As a result, the ϵ for security is $\frac{1}{2^{32}}$ rather than $\frac{1}{p}$. □

However, it is not immediately clear how this new family is so much more efficient computationally than the original MMH^*.

The majority of the improvement lies simplifying the modular reduction during or after the calculation of the dot product. General implementations of modular reduction are centered around division; division, not being inbuilt to the hardware of computers like addition and multiplication are, is a comparatively intensive task computationally. To combat this, the choice $p = 2^{32} + 15$ or of a similar p in MMH^*_{32} allows for the use of divisionless modular reduction. Noting that a pair of unsigned 32-bit integer values a, b can represent any unsigned 64-bit integer value via the representation $2^{32}a + b$, we can deduce that

$$2^{32}a + b \equiv (2^{32}a + b) - a(2^{32} + 15) \equiv b - 15a \pmod{2^{32}+15} \tag{5.8}$$

By this method the modular reduction is calculated using the much faster CPU operations multiplication and addition. There exist methods with which the calculation of $b - 15a$ may be sped up optimally so the entire process requires only 10-15 machine instructions, but the core of the speed increase is this wholesale removal of the need for division-based modular reduction in the implementation of a very nearly Δ-universal hash function. Alongside the 64-bit scale of modern processors and the hardware-accelerated dot-product calculation available on newer systems, MMH^*_{32} runs fast enough that it can easily and safely authenticate data streams as large as real-time video streaming without issue.

6. Conclusion and Further Reading

This paper has provided the requisite tools to understanding the internal operations of the family of hash functions MMH_{32}^* as well as the context necessary to appreciate its potential usefulness to data authentication and large-scale tasks.

However, the subject of hash functions is quite wide and extends far beyond this paper's basic mathematical analysis of universal families: the most mathematically-oriented, introductory-level concept in the subject. As such, this section is included to point the reader toward other aspects of the subject, particularly those of more central importance to the use of hash functions generally.

The *birthday attack*, the theoretical basis for many attacks on hash functions has a significant amount of depth to offer. Along the same lines, the reader might look into the *rho attack*, a secondary hash function-breaking attack which requires less memory than the birthday attack. [6]

For the general methods by which many cryptographic hash functions are produced, the reader can look into the idea of Merkle-Damgård Construction, which is the basis of functions such as SHA-1 (Standard Hash Function-1) and SHA-2. The reader can also look into Sponge Functions, a common construction which is the basis of many more recent hash functions including SHA-3. Other methods do exist and have produced non-standard but widely-used functions like BLAKE2.[6]

For an example of a hash function which was once safe and was later broken, see the aforementioned function SHA-1 and the associated website shattered.io, which illustrates a collision between two meaningful and similar PDF-format messages[7].

Lastly, for specific, simple instances of universal families of hash functions, the reader can look into the families H_1 and H_3. Additionally, H_2 is an extension of H_3 which is more memory intensive and less computationally intensive. All three of these families were introduced by Carter and Wegman.[4]

7. Acknowledgments

I would first like to thank Maeve Coates Welsh, my mentor, for her guidance through a topic unfamiliar to us both and for her notes which brought a much needed dose of coherence to this paper. I would also like to thank Peter May for his work organizing the entire UChicago REU program. Finally, I would like to thank my family for all of their support and encouragement during the writing process.

References

[1] Wade Trappe, Lawrence C. Washington. Introduction to Cryptography with Coding Theory. Pearson Education, Ltd. 2006.

[2] Rogaway, P. and T. Shrimpton. Cryptographic hash function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. Fast Software Encryption, Lecture Notes in Computer Science, vol. 3017, eds. B.K. Roy and W. Meier. Springer-Verlag, Berlin, 371–388. 2004.

[3] Ran Canneti, Oded Goldreich, Shai Halevi. The Random Oracle Model, Revisited. In Proceedings of 30th Annual ACM Symposium on the Theory of Computing, pages 209-218. 1998.

[4] J. Lawrence Carter, Mark N. Wegman. Universal Classes of Hash Functions. Journal of Computer and System Sciences, Volume 18 Issue 2. 1979.

[5] Shai Halevi, Hugo Krawczyk. MMH: Software Message Authentication in the Gbit/Second Rates. Biham E. (eds) Fast Software Encryption. 1997.

[6] Jean-Philippe Aumasson. Serious Cryptography: A Practical Introduction to Modern Encryption. No Starch Press, Inc. 2018.

[7] https://shattered.io/