A POLYNOMIAL BASED HASHING ALGORITHM

V. Kumar Murty
Department of Mathematics, University of Toronto, 40 St. George Street, Toronto, M5S 3G3, Canada

Nikolajs Volkovs
GANITA Lab, Department of Mathematical and Computational Sciences
University of Toronto - Mississauga, 3359 Mississauga Road, Mississauga, Canada

Keywords: Hash function, data integrity, polynomials over finite fields.

Abstract: The aim of this article is to describe a new hash algorithm using polynomials over finite fields. It runs at speeds comparable to SHA-3. Hardware implementations seem to run at significantly faster speeds, namely at 1.8 Gb/sec on an FPGA. Unlike most other existing hash algorithms, our construction does not follow the Damgaard-Merkle philosophy. The hash has several attractive features in terms of its flexibility. In particular, the length of the hash is a parameter that can be set at the outset. Moreover, the estimated degree of collision resistance is measured in terms of another parameter whose value can be varied.

1 INTRODUCTION

There is much discussion now about how to construct a good hash algorithm. The main difficulty stems from the fact that the design principles of a hash function are not completely understood. However, some desirable features of a future hash function have been enumerated (NIST,2006). In particular, the algorithm should provide for

1. a changeable length of hash
2. collision resistance measured in terms of a tunable parameter
3. different functions for online and offline usage.

There are some methods that have been studied in the literature to produce new hash functions from old functions. For example, one might consider the concatenation of two existing hash functions. Or, one might increase the number of rounds in an existing function. It is not clear what effect these methods have on collision resistance. In general, it seems that the design principles of hash functions are not well understood.

The aim of this article is to describe a new hash algorithm, which incorporates some of the features specified in (NIST,2006). In particular, the output length of the function is a parameter that can be set at the beginning. Moreover, the degree of collision resistance is expected to depend on another parameter that can also be specified at the outset.

The performance of the algorithm is comparable to that of the SHA-family. In particular, for a 384 bit hash, the speed is comparable to SHA-384. For a 512 bit hash, the speed is 5% faster than SHA-512.

The hardware implementation of the algorithm is especially effective. Modelling shows that the speed of the function may reach 1.8 Gbits/sec (on an FPGA V running at 299 MHz). Analysis also suggests that on an ASIC, we may obtain speeds of approximately 4 Gbits/sec (600 MHz). We note that the structure of the algorithm is such that the performance improves for longer files and larger hash sizes. The algorithm has two phases, the second of which does not depend on the length of the message being hashed.

The collision resistance of the function is dependent on the difficulty of solving a family of systems of iterated exponential equations in a finite field. This is not a problem that has been studied in the literature. However, it does not seem to be tractable by standard methods of analytic number theory.

Summarizing, the hash function that we construct has the following important attributes. Firstly, the length of the output can be changed simply by changing a few steps of the calculation. Secondly, an aspect of the construction, namely the use of “bit strings” allows us to choose the degree of collision resistance for a fixed output length. Thirdly, the computation is a bit-stream procedure as opposed to a block procedure. Finally, we note that creating a collision requires the solution of a system of non-linear iterated exponential equations. As far as we are aware, such equations cannot be solved by standard methods of analytic number theory. Moreover, we are not aware of any other hash function in the literature whose collision resistance involves such iterated exponential equations.

For its performance characteristics, design features and dependence on what appears to be an intractible mathematical problem, we believe this hash function is worthy of further attention, and so we present this preliminary report.

Our construction uses polynomials over finite fields. We note that earlier works have used polynomials over finite fields in the construction of hash algorithms. See for example, the work of Krovetz and Rogoway (Krovetz and Rogoway, 2000). However, our use of polynomials is very different. Most other existing hash algorithms are based on the Damgaard-Merkle (Damgaard,1989), (Merkle,1989) approach. The reader can find a description of such algorithms in the book of Menezes, van Oorschot and Vanstone (Menezes et al,1997).

Following the recent ground-breaking work of Wang (Wang et al, 2005), the Damgaard-Merkle design methodology has come under close scrutiny. Our approach, however, is not based on the Damgaard-Merkle methodology.

2 BRIEF DESCRIPTION OF THE STEPS

The main steps are stretching, masking, forming a collection of tables with bit strings, forming a knapsack and performing one exponentiation in a group. We briefly describe each of these steps.

2.1 Padding, Splitting and Masking

The message is stretched by appending 4096 bits consisting of a fixed string. This stretching operation is different from the one described in, for example, Aiello, Haber and Venkatesan (Aiello et al, 1998) in which a randomized function is used to perform stretching. In our algorithm, the purpose of stretching is to populate certain auxiliary bit strings. Let us denote by k the length in bits of the padded message M.

The message is then split into overlapping segments which are interpreted as polynomials over $\mathbb{F}_2$ of degree $< n$ where n is chosen such that $3 < n < 11$. More precisely, denote by $M(i,j)$ the substring of M beginning with the i-th bit and ending with the j-th bit. Also, denote by $M[i]$ the i-th bit of M. Let us define $S(M,n)$ to be the set $M(1,n), M(2,n+1), \cdots, M(k-n+1,k), M(k-n+2)M[1], M(k-n+3)M(1,2), \cdots M[k]M(1,n-1)$. Each $M(i,i+n-1)$ may be thought of as a polynomial of degree $< n$ over $\mathbb{F}_2$. Thus, $S(M,n)$ consists of k polynomials of degree $< n$. Note that the construction of the $S(M,n)$ is a stream procedure. We choose c values of n, where c is a variable parameter.

Next, we perform a complicated iterative mathematical procedure which we call masking. It is one-to-one and length preserving. Though one-to-one, the procedure is difficult to invert and it involves finite field arithmetic. Let us denote by $CUR_1, \cdots, CUR_k$ the effect of this procedure. We view them as k polynomials of degree $< n$. At any given time, we need to store 2^n of these polynomials.

For any bit string B, we define $int(B)$ to be the integer whose base 2 expansion is B. The registers CUR_i are constructed as follows. Set

$$d_1 = d_1(i) = i - 2 - int(M_{i-1}), \qquad (1)$$

$$d_2 = d_2(i) = i - 2 - int(CUR_{i-1}).$$

Let $f(x) \in \mathbb{F}_2[x]$ be irreducible of degree n. Thus, there is an isomorphism of fields

$$\mathbb{F}_2[x]/(f(x)) \simeq \mathbb{F}_{2^n}.$$

Denote by ϕ_f the isomorphism of $\mathbb{F}_2$-vector spaces

$$\mathbb{F}_2[x]/(f(x)) \longrightarrow \mathbb{F}_2^n.$$

Let δ and β be generators of $(\mathbb{F}_2[x]/(f(x)))^\times$ (resp. $(\mathbb{F}_2[x]/(g(x)))^\times$) corresponding to polynomials $f(x)$ and $g(x)$ say. We set

$$CUR_1 = M_1 \oplus \phi_f(\delta) \oplus \phi_g(\beta), \qquad (2)$$

$$CUR_i = M_i \oplus$$
$$\phi_f(\delta^{(int(M_{i-1})+int(CUR_{i-2})mod 2^n})\oplus$$
$$\oplus\phi_g(\beta^{(int(CUR_{i-1})+int(CUR_{i-2}))mod 2^n)})$$

for $i = 2, \ldots, 2^n+1$, and

$$CUR_i = M_i \oplus \qquad (3)$$
$$\phi_f(\delta^{int(M_{i-1})+int(CUR_{d_1})mod 2^n})\oplus$$
$$\oplus\phi_g(\beta^{(int(CUR_{i-1})+int(CUR_{d_2})mod 2^n})$$

for $i = 2^n+2, \ldots, k$ with d_1 and d_2 defined by (1). Once again, we stress that the procedure just described for calculating the values CUR_i is a stream procedure. Moreover, as the result below indicates, the values of the CUR_i uniquely determine the original message M.

Proposition 2.1. *Let M and M' be messages of length k with $CUR_i(M) = CUR_i(M')$ for $i = 1, \cdots, k$. Then $M = M'$.*

The proof is given in (Murty and Volkovs,2008).

If we choose c values $n_1, \cdots, n_c$ of n, then we calculate the values of registers CUR^{n_j} for all the elements of the sequences

$$S(M,n_1), S(M,n_2), ..., S(M,n_c).$$

Of course, for different n_j we use different fields $\mathbb{F}_{2^{n_j}}$ with the corresponding generators. Thus, we have constructed the following set of collections of polynomials

$$CUR_i^{n_1}, CUR_i^{n_2}, ..., CUR_i^{n_c}, \tag{4}$$

2.2 Construction of Tables

We construct a number (we considered 200) of tables containing 2^n entries each. These tables are initially set to zero and then individual entries of the table are incremented according to a simple rule that depends on the bits of the stretched message and the values of the CUR_i. At the end of this operation, we produce tables the average entry of which is $k/(2^n * 200)$. In particular, for a message of 1MB and using $n = 4$, the average table entry will be an integer of size 2500. By construction, the sum of the entries in the tables is the same as the message length, namely k.

Choose an integer $r > 2^n$. We will construct r tables $Tb_1, \cdots, Tb_r$ as follows. Let $q > 0$ and $g > 0$ be chosen so that $q + g < n$. For example, we may take $q = g = 1$. For any integer m, denote by $i = int(m \pmod r)$ the integer with $0 \le i < r$ and $m = i \pmod r$. Define the function

$$h = h(M) : \{1, 2, \cdots, k\} \longrightarrow \{1, 2, \cdots, r\}$$

as follows:

$$h(1) = \begin{cases} int(q \pmod r) + 1 & \text{if } M[1] = 1 \\ int(q + g \pmod r) + 1 & \text{if } M[1] = 0. \end{cases}$$

For $i > 1$,

$$h(i) =$$

$$\begin{cases} h(i-1) + int(q \pmod r) + 1 & \text{if } M[i] = 1 \\ h(i-1) + int(q + g \pmod r) + 1 & \text{if } M[i] = 0. \end{cases}$$

If on the i-th step of the distribution, $M[i]$ is 1, we assign polynomial CUR_i to the table $h(i)$. By "assigning" an element CUR_i to a table with index $h(i)$ we mean that we increment the counter $int(CUR_i)$ - th element of the $h(i)$-th table by 1. We stress that r, q and g are chosen so that it is impossible to send any two neighboring polynomials from any sequence CUR to one and the same table.

In practice, we have more than one value of n. Thus, the above construction requires the choice of $r_1, \cdots, r_c$ corresponding to $n_1, \cdots n_c$. The construction produces tables $Tb_i^{(j)}$ for $1 \le j \le c$ and $1 \le i \le r_j$.

2.3 Bit Strings

This is a technical construction which involves associating a number of strings to selected entries of each table. The length of these strings is a parameter that can be chosen. The number of strings per table is another parameter that can be chosen, with fewer strings corresponding to greater loss of information. In our tests, we associated up to 5 bit string to each table. These strings are constructed so as to capture information about the order in which the entries of the table are updated. These strings require an additional memory of about 5KB, assuming one string of length 200 bits per table. Details of this construction will be presented in (Murty and Volkovs,2008).

2.4 The Spectrum

From these tables, we apply a certain linear transformation to construct a "spectrum". The maximum value of a spectrum entry is approximately $k/10$ and the average value is $k/(2^{n+1} * 10)$. In particular, for a message of length 1MB, the spectrum from each table contains about 25 bits. It is not necessary to store more than one spectrum at a time.

2.5 Knapsack

Using the spectrum, we perform a Cantor enumeration which computes one single number for each table. This number is of the order $(k/10)^{16}$. In particular, for a 1MB file, this is about 40 bytes. To the integer obtained by enumeration, we add the integer corresponding to the bit string. This produces, for each table t, an integer I_t. Now we form the sum

$$I = \sum tI_t$$

which is an integer of length 320 bits.

2.6 Exponentiation and the Hash Value

Now fix a group G and P an element of this group. We compute the point multiple $I.P$. For the group, we can choose, for example, the group of points on an elliptic curve over a finite field of 2^τ elements, where τ is a parameter that can be chosen. Finally, the hash value if the x-coordinate of the point $I.P$. It belongs to a finite field of 2^τ elements and can be interpreted as a bit string of length τ.

2.7 Outline of the Algorithm

Our algorithm, then, can be described in brief as follows.

```
PARAMETERS: $c$, $n_1, \cdots, n_c$, $\{r_j, s_j, g_j, q_j\}$, $\tau$
INPUT: Message $M$ of length $k$
OUTPUT: Hash value $H$ of $M$ of $\tau$ bits
1. Compute the stretching and splitting $S(M, n_j)$ $(1 \leq j \leq c)$
2. Compute the masking $CUR_i^{n_j}$ for $1 \leq j \leq c$ and $1 \leq i \leq k$.
3. Compute the tables $Tb_i^{(n_j)}$ for $1 \leq j \leq c$ and $1 \leq i \leq r_j$. Each table has $2^{n_j}$ entries and each entry has $s_j$ bits.
4. Compute bit strings and their associated integers $BS_i^{n_j}$.
5. From the tables, compute the spectra and their associated integers $E_i^{n_j}$.
6. Use both sets of integers to compute an integer $I_j$ (for $1 \leq j \leq c$).
7. Compute $H_j$ in the group.
8. Put the $H_j$ together to form final hash value $H$.
```

3 PRELIMINARY ANALYSIS

We tested the algorithm on the randomness tests provided by NIST. The results were as follows. For all five output lengths, namely 160, 224, 256, 384 and 512 bits, all the tests were successfully passed. Moreover, all of the results we obtained during running the NIST tests (NIST2,2006) show that we passed the tests with certain reserve. For instance, for the Frequency Test, in accordance with which “the number of 1’s must be between 9,654 and 10,346”, we got that the interval of number of 1’s for the 256-th bit of the output was between 9817 and 10225.

The collision resistance of the algorithm is dependent on the difficulty of solving several problems including the problem of iterated exponential equations over a finite field. Details will be given in the extended paper (Murty and Volkovs,2008) in preparation.

4 SUMMARY

The hash function that has been briefly described in this announcement offers several attractive features such as tunable output length, a tunable measure of collision resistance. Moreover, in hardware it is running at 1.8 Gb/sec on an FPGA and it is expected to run even faster on an ASIC. The design methodology is not based on the Damgaard-Merkle approach and is a bit-stream procedure.

REFERENCES

Aiello, W., Haber, S., and Venkatesan, R. (1998). New constructions for secure hash functions (Extended abstract). In *Fast Software Encryption, LNCS vol.1372*, pages 150-167. Springer Verlag, Berlin.

Damgaard, I. (1989). A design principle for hash functions. In *Advances in Cryptology, LNCS 435*, pages 416-427. Springer Verlag, Berlin.

Hankerson, D., Menezes, A., and Vanstone, S. (2004). *Guide to Elliptic curve cryptography*. Springer-Verlag, New York.

Krovetz,T., and Rogoway,P. (2000). Fast universal hashing with small keys and no preprocessing: the PolyR construction. In *Information Security and Cryptology ICICS 2000, LNCS vol. 2015*, pages73-89. Springer-Verlag, Berlin.

Mal’cev, A.I. (1970) *Algorithms and recursive functions*. Wolters-Noordhoff Pub.Co.

Menezes, A., van Oorschot, P.C., and Vanstone, S. (1997) *Handbook of Applied Cryptography*. CRC Press.

Merkle, R. (1989). A Certified Digital Signature. In *Advances in Cryptology, LNCS 435*, pages 218-238. Springer Verlag, Berlin.

Murty, V. Kumar, and Volkovs, N. (2008). ERINDALE: A polynomial based hashing algorithm. In preparation.

National Institute of Standards and Technology (2006). *Second NIST Workshop on Hash functions*. http://csrc.nist.gov/groups/ST/hash/second workshop.html, August 24-25, 2006.

National Institute of Standards and Technology (2006). Computer Security Division, Computer Security Resource Center. http://csrc.nist.gov/groups/STM/cavp/standards.html

Wang, X., Yin, Y., and Yu, H. (2005). Finding collisions in the full SHA-1. In *Advances in Cryptology, LNCS 3621*, pages 17-36. Springer Verlag, Berlin.