Hashing with Polynomials

Vladimir Shpilrain

Department of Mathematics, The City College of New York, New York, NY 10031
shpil@groups.sci.ccny.cuny.edu [*]

Abstract. In this paper, we explore potential mathematical principles and structures that can provide the foundation for cryptographic hash functions, and also present a simple and efficiently computable hash function based on a non-associative operation with polynomials over a finite field of characteristic 2.

1 Introduction

Hash functions are easy-to-compute compression functions that take a variable-length input and convert it to a fixed-length output. Hash functions are used as compact representations, or digital fingerprints, of data and to provide message integrity. Some hash functions in current use have been shown to be vulnerable. In [6], the author argues that their replacements should be based on a mathematical theory, which has yet to be created.

Before such a theory can be created, one has to describe, as detailed as possible, mathematical properties that a "good" hash function should have. Of course, basic requirements are well known:

1. *Preimage resistance* (sometimes called *non-invertibility*): it should be computationally infeasible to find an input which hashes to a specified output;
2. *Second pre-image resistance*: it should be computationally infeasible to find a second input that hashes to the same output as a specified input;
3. *Collision resistance*: it should be computationally infeasible to find two different inputs that hash to the same output.

Now the problem is to determine mathematical properties of a hash function that would ensure (or at least, make it likely) that the requirements above are met.

Early suggestions (especially the SHA family) did not really use any mathematical ideas apart from the Merkle-Damgard construction for producing collision-resistant hash functions from collision-resistant compression functions (see e.g. [7]); the main idea was just to "create a mess" by using complex iterations (this is not meant in a derogatory sense, but just as an opposite of using mathematical structure one way or another). We have to admit that a "mess" might be good for hiding purposes, but only to some extent. In particular, several early suggestions

[*] Partially supported by the NSF grant DMS-0405105 and by an NSA grant.

were successfully attacked, see e.g. [10]. However, the market has its own rules, and a product that has won the market is unlikely to be replaced by something altogether different; it is more likely that it will be slightly adjusted every time the older version becomes outdated for one reason or another. In other words, in 20 years from now, something like SHA-32768 is more likely to be commercially used than any hash function based on mathematical ideas. To be fair, we have to mention an interesting direction, namely, constructing hash functions being provably as secure as underlying assumptions, e.g. as discrete logarithm assumptions; see [4] and references therein. These hash functions however tend to be not very efficient. For a general survey of hash functions we refer to [7].

One especially discouraging example of the trend of ignoring elegant mathematical ideas is the Tillich-Zémor hash function [9], which is quite simple, efficient, and intelligent, and yet it was almost completely ignored; for example, Google search for this hash function produces about 200 results, compared to over 18,000,000 for SHA-1. This, by any standards, is not healthy, especially since at this time, there is no compelling reason to doubt the security of the Tillich-Zémor hash function. To the best of our knowledge, there are just 3 published papers [3], [5], [8] offering "attacks" (i.e., exposing collisions) on this hash function for some very special values of parameters. For "generic" values of parameters, the Tillich-Zémor hash function withstands all known attacks; in particular, it was shown in [1] that it is vulnerable to the attack of [3] with probability of (approximately) 10^{-27}. Of course, it is difficult to *prove* collision resistance of the Tillich-Zémor hash function, but this is true for most other hash functions, too. (We note that recently, a proposal for a hash function has been made [2] where collision resistance follows from the alleged hardness of a mathematical problem related to *expander graphs*; the authors of [2] acknowledge that one of their constructions is similar to that of Tillich and Zemor.)

In this paper, we try to somewhat remedy the situation. First, in Sections 2 and 3, we use the Tillich-Zémor hash function as a model example to analyze mathematical principles and structures behind a secure and efficient hash function. We speculate that a successful hash function should be based on a (finite) *dynamical system*, of which the Tillich-Zémor construction is a nice example.

Then, in Section 4, we present a simple and efficiently computable hash function which is based on essentially the same dynamical system as the Tillich-Zémor hash function is. In particular, it has the advantages of the Tillich-Zémor hash function, but does not have its (potential) weaknesses. We suggest specific parameters for our hash function in Section 5. However, we have to say up front that our proposal is on a conceptual level; in particular, while our hash function is obviously (by comparing the definitions) more efficient than that of Tillich-Zémor, we do not report actual runtimes here.

2 Tillich-Zémor hash function: three useful features

The Tillich-Zémor hash function, unlike functions in the SHA family, is *not* a block hash function, i.e., each bit is hashed individually. More specifically, the

"0" bit is hashed to a particular 2×2 matrix A, and the "1" bit is hashed to another 2×2 matrix B. Then a bit string is hashed simply to the product of matrices A and B corresponding to bits in this string. For example, the bit string 1000110 is hashed to the matrix BA^3B^2A.

Obviously, this kind of arrangement is possible with *any* pair of elements A, B of *any* semigroup S. The question is: what choice of semigroup S and elements A, B makes the corresponding hash function secure? We argue here that the choice made by Tillich and Zémor in [9] has three useful features which are, in our opinion, significant for cryptographic security in general and for the security of a hash function in particular.

First we recall that Tillich and Zémor use matrices A, B from the group $SL_2(R)$, where R is a commutative ring (actually, a field) defined as $R = \mathbf{F}_2[x]/(p(x))$. Here $\mathbf{F}_2$ is the field with two elements, $\mathbf{F}_2[x]$ is the ring of polynomials over $\mathbf{F}_2$, and $(p(x))$ is the ideal of $\mathbf{F}_2[x]$ generated by an irreducible polynomial $p(x)$ of degree n (typically, n is a prime, $127 \leq n \leq 170$); for example, $p(x) = x^{131} + x^7 + x^6 + x^5 + x^4 + x + 1$. Thus, $R = \mathbf{F}_2[x]/(p(x))$ is isomorphic to $\mathbf{F}_{2^n}$, the field with 2^n elements.

Then, the matrices A and B are:

$$A = \begin{pmatrix} \alpha & 1 \\ 1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} \alpha & \alpha + 1 \\ 1 & 1 \end{pmatrix},$$

where α is a root of $p(x)$.

Now the three useful features of the Tillich-Zémor hash function are:

1. The *commutativity* of the ring R is good for "diffusion", i.e., for hiding occurrences of A or B in a product.
2. The *periodicity* of the ring R, too, is good for "diffusion". (Periodicity means that for any $u \in R$, there is a positive integer m such that $u^m = u$.)
3. The non-commutativity of matrix multiplication prevents from obvious collisions. For example, were the multiplication commutative, the bit strings "01" and "10" would hash to the same thing.

We emphasize once again that

$$COMMUTATIVITY \qquad and \qquad PERIODICITY$$

are two major tools for hiding factors in a product; their importance for cryptographic security in general and for the security of a hash function in particular cannot be overestimated. Furthermore, a commutative and periodic platform gives rise to a *dynamical system*, where two functions (corresponding to the "0" and the "1" bits) act in a rather complex way. Dynamical systems with a large number of states, even "innocent-looking" ones, usually exhibit very complex behavior; it is sufficient to mention the notorious "3x+1" problem. In particular, any instances of re-occurring state are usually very difficult to predict (again, recall the "3x+1" problem). In the context of hash functions, those correspond to collisions, thus making the latter difficult to detect.

At the same time, for better security, commutativity might be reinforced by non-commutativity pretty much the same way as concrete is reinforced by steel to produce ferroconcrete. Thus,

$$COMMUTATIVITY \quad \text{in the corset of} \quad NON-COMMUTATIVITY$$

is another important ingredient of cryptographic security. It prevents the attacker from using obvious "relations", such as $ab = ba$, to simplify a product.

To conclude this section, we say a few words about the efficiency of the Tillich-Zémor hash function. Computing this function involves the following operations with polynomials over $\mathbf{F}_2$:

1. Multiplication and addition of polynomials in α of degrees bounded by that of the polynomial $p(x)$.
2. Division of a polynomial in α whose degree is at most twice the degree of $p(x)$, by the polynomial $p(\alpha)$.

These operations are quite efficient; in fact, their time complexity is bounded by a constant which depends only on (the degree of) the polynomial $p(x)$. Since the suggested degree of $p(x)$ is fairly small (see above), the constant in question is small, too.

3 Tillich-Zémor hash function: two potential weaknesses

In this section, we discuss two features of the Tillich-Zémor hash function which may, in our opinion, yield undesirable trapdoors.

(i) The matrices A and B are *invertible*.
(ii) The operation (matrix multiplication) used in hashing is *associative*, i.e., $a(bc) = (ab)c$ for any $a, b, c \in SL_2(R)$.

These two features together may not be good from the security point of view, for the following reasons. First, suppose the intruder knows part of a hashed bitstring $\mathcal{S}$: say, $\mathcal{S} = \mathcal{S}_1\mathcal{S}_2$, and assume the intruder knows $\mathcal{S}_1$. Then, because of the property (ii) above, we have for the hashes:

$$H(\mathcal{S}) = H(\mathcal{S}_1) \cdot H(\mathcal{S}_2). \tag{1}$$

Since we assume that the intruder knows $\mathcal{S}_1$, he also knows $H(\mathcal{S}_1)$, and therefore, by using property (i) (invertibility of hashes) and the above equality, he can recover $H(\mathcal{S}_2)$, thereby making it somewhat easier to recover $\mathcal{S}_2$ and then $\mathcal{S}$.

Another, more serious, weakness implied by the associativity is the following. Suppose a collision is found, say, $H(\mathcal{T}_1) = H(\mathcal{T}_2)$ for some bitstrings $\mathcal{T}_1, \mathcal{T}_2$. Then the equality (1) above yields, for any bitstring $\mathcal{S}$: $H(\mathcal{S}T_1) = H(\mathcal{S}T_2)$. Thus, one collision easily yields many other.

To be fair, we have to point out (see e.g. [8]) that the property (ii) above is useful to make hashing more efficient because it allows one to parallelize computation, e.g. to compute the hash $H(\mathcal{S}_1 S_2)$ as $H(\mathcal{S}_1) \cdot H(\mathcal{S}_2)$.

Finally, we note that a fairly well understood group structure of $SL_2(F_{2^n})$ may eventually help to find semigroup relations between the matrices A and B, thus revealing a collision. In this sense, the absence of structure (i.e., a "mess") is something that can be borrowed from the SHA family in this context.

4 Hashing with polynomials

In this section, we present a simple and efficiently computable hash function which has the advantages of the Tillich-Zémor hash function, but does not seem to have its (potential) weaknesses.

Let $R = \mathbf{F}_{2^n} = \mathbf{F}_2[x]/(p(x))$ and α be as in the Tillich-Zémor construction (see also our Section 2). Let $P(\alpha)$ and $Q(\alpha)$ be two elements of R; they are going to be hashes of the "0" and the "1" bit, respectively:

$$H(0) = P(\alpha), \qquad H(1) = Q(\alpha).$$

The hash of the concatenation $\mathcal{S}_1\mathcal{S}_2$ of two bitstrings is computed by the following recursive formula, *which is only applied in the situations described below (after the formula)*:

$$H(\mathcal{S}_1\mathcal{S}_2) = H(\mathcal{S}_1) \circ H(\mathcal{S}_2) = H(\mathcal{S}_1) \cdot H(\mathcal{S}_2) + (H(\mathcal{S}_1))^2 \cdot u_1(\alpha) + (H(\mathcal{S}_2))^2 \cdot u_2(\alpha) + v(\alpha),$$

where $u_i(\alpha)$ and $v(\alpha)$ are some fixed elements of R. We note that the operation $\circ$ is non-associative and non-commutative if $u_1(\alpha) \neq u_2(\alpha)$.

Now suppose $\mathcal{S}$ is a bitstring of length $n \geq 2$. To hash $\mathcal{S}$:

1. Split $\mathcal{S}$ into blocks $\mathcal{B}_1, \mathcal{B}_2, \ldots$ of length 32 going left to right (the rightmost block may therefore have smaller length).
2. Compute the hash of each block $\mathcal{B}_i$ independently, going left to right bit by bit and using the recursive formula above with $\mathcal{S}_2$ being a single bit every time.
3. Compute the hash of $\mathcal{S}$ inductively, going left to right block by block and using the recursive formula above with $\mathcal{S}_2$ being a single block $\mathcal{B}_i$ every time.

We emphasize once again that if $u_1(\alpha) \neq u_2(\alpha)$, then the operation $\circ$ defined above is non-associative and non-commutative. However, since the ring R itself is commutative (and periodic), we take full advantage of commutativity and periodicity as hiding tools here. In fact, our hash function is based on essentially the same dynamical system as the Tillich-Zémor hash function; we just get rid of the matrices to avoid associativity and invertibility. At the same time, since our operation $\circ$ is non-commutative, we have the "commutativity in the corset of non-commutativity" property that was discussed in our Section 2.

Thus, our hash function has the same advantages as the Tillich-Zémor hash function does. On the other hand, it does not have the weaknesses discussed in our Section 3. Indeed, we have already mentioned that the operation $\circ$ is non-associative. It is also "non-invertible" in the following sense. If, for some bitstring $\mathcal{S} = \mathcal{S}_1\mathcal{S}_2$, you know $H(\mathcal{S})$ and $H(\mathcal{S}_1)$, this does not allow you to find $H(\mathcal{S}_2)$ the way it can be done for the Tillich-Zémor hash function.

Moreover, if you know $H(\mathcal{S}_1)$ and $H(\mathcal{S}_2)$, this does not help you, in general, to find $H(\mathcal{S}_1 S_2)$, because the formula for $H(\mathcal{S}_1\mathcal{S}_2) = H(\mathcal{S}_1) \circ H(\mathcal{S}_2)$ is applicable only in very special cases of $\mathcal{S}_1$ and $\mathcal{S}_2$, see the definition above. This implies, in particular, that knowing one collision does not immediately yield any other, contrasting the situation with the Tillich-Zémor hash function (see the previous section).

5 Parameters

In this section, we suggest particular polynomials that can be used in the definition of a hash function given in the previous section. There is no specific motivation behind this particular choice of parameters; as with most dynamical systems, "generic" parameters yield sufficiently complex behavior of the system.

1. In the definition of $R = \mathbf{F}_{2^n} = \mathbf{F}_2[x]/(p(x))$, we suggest to take
$$p(x) = x^{163} + x^7 + x^6 + x^5 + x^4 + x + 1.$$
Thus, any bitstring will be hashed to a polynomial of degree at most 162 over $\mathbf{F}_2$, which is equivalent to hashing to a 163-bit string.
2. In the definition of $H(0) = P(\alpha),\ H(1) = Q(\alpha)$, we suggest to take
$$H(0) = P(\alpha) = \alpha^7 + 1, \qquad H(1) = Q(\alpha) = \alpha^8 + 1.$$
3. In the definition of
$H(\mathcal{S}_1 S_2) = H(\mathcal{S}_1) \circ H(\mathcal{S}_2) = H(\mathcal{S}_1) \cdot H(\mathcal{S}_2) + (H(\mathcal{S}_1))^2 \cdot u_1(\alpha) + (H(\mathcal{S}_2))^2 \cdot u_2(\alpha) + v(\alpha)$, we suggest to take
$$u_1(\alpha) = \alpha^2,\ u_2(\alpha) = \alpha, \qquad v(\alpha) = 1.$$

Thus, whenever applicable,
$$H(\mathcal{S}_1 S_2) = H(\mathcal{S}_1) \cdot H(\mathcal{S}_2) + (H(\mathcal{S}_1))^2 \cdot \alpha^2 + (H(\mathcal{S}_2))^2 \cdot \alpha + 1.$$

Below we give examples of computing $H(\mathcal{S})$ for some simple bitstrings $\mathcal{S}$.

1. $H(00) = H(0) \cdot H(0) + (H(0))^2 \cdot \alpha^2 + (H(0))^2 \cdot \alpha + 1 = \alpha^{16} + \alpha^{15} + \alpha^{14} + \alpha^2 + \alpha.$
2. $H(01) = H(0) \cdot H(1) + (H(0))^2 \cdot \alpha^2 + (H(1))^2 \cdot \alpha + 1 = \alpha^{17} + \alpha^{16} + \alpha^{15} + \alpha^8 + \alpha^7 + \alpha^2 + \alpha.$

3. $H(10) = H(1)\cdot H(0) + (H(1))^2\cdot\alpha^2 + (H(0))^2\cdot\alpha + 1 = \alpha^{18} + \alpha^8 + \alpha^7 + \alpha^2 + \alpha.$

4. $H(11) = H(1)\cdot H(1) + (H(1))^2\cdot\alpha^2 + (H(1))^2\cdot\alpha + 1 = \alpha^{18} + \alpha^{17} + \alpha^{16} + \alpha^2 + \alpha.$

5. $H(001) = H(00)\cdot H(1) + (H(00))^2\cdot\alpha^2 + (H(1))^2\cdot\alpha + 1 =$
$= \alpha^{34}+\alpha^{32}+\alpha^{30}+\alpha^{24}+\alpha^{23}+\alpha^{22}+\alpha^{17}+\alpha^{16}+\alpha^{14}+\alpha^{10}+\alpha^9+\alpha^6+\alpha^4+\alpha^2+1.$

6. $H(010) = H(01)\cdot H(0) + (H(01))^2\cdot\alpha^2 + (H(0))^2\cdot\alpha + 1 =$
$\alpha^{36}+\alpha^{34}+\alpha^{32}+\alpha^{24}+\alpha^{23}+\alpha^{22}+\alpha^{18}+\alpha^{15}+\alpha^{14}+\alpha^9+\alpha^8+\alpha^7+\alpha^6+\alpha^4+\alpha^2+1.$

7. $H(110) = H(11)\cdot H(0) + (H(11))^2\cdot\alpha^2 + (H(0))^2\cdot\alpha + 1 =$
$\alpha^{38}+\alpha^{36}+\alpha^{34}+\alpha^{25}+\alpha^{24}+\alpha^{23}+\alpha^{18}+\alpha^{17}+\alpha^{16}+\alpha^{15}+\alpha^9+\alpha^8+\alpha^6+\alpha^4+\alpha^2+1.$

Acknowledgement. I am indebted to Rainer Steinwandt for numerous helpful discussions.

References

1. K. S. Abdukhalikov and C. Kim, *On the Security of the Hashing Scheme Based on SL_2*, in FSE 1998, Lecture Notes Comp. Sc. **1372** (1998), 93-102.
2. D. Charles, E. Goren, and K. Lauter, *Cryptographic hash functions from expander graphs*, preprint. http://www.math.mcgill.ca/goren/PAPERSpublic/Hashfunction.pdf
3. C. Charnes and J. Pieprzyk, *Attacking the SL_2 hashing scheme*, in ASIACRYPT 1994, Lecture Notes Comp. Sc. **917** (1995), 322-330.
4. S. Contini, A. K. Lenstra and R. Steinfeld, *VSH, an Efficient and Provable Collision Resistant Hash Function*, in: Eurocrypt 2006, Lecture Notes Comp. Sc. **4004** (2006), 165–182.
5. W. Geiselmann, *A Note on the Hash Function of Tillich and Zémor*, in Cryptography and Coding, Lecture Notes Comp. Sc. **1025** (1995), 257-263.
6. S. Landau, *Find Me a Hash*, Notices Amer. Math. Soc. **53** (2006), 330–332.
7. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.
8. R. Steinwandt, M. Grassl, W. Geiselmann, T. Beth, *Weaknesses in the* $\mathrm{SL}_2(\mathbf{F}_{2^n})$ *Hashing Scheme*, in CRYPTO 2000, Lecture Notes Comp. Sc. **1880** (2000), 287–299.
9. J.-P. Tillich and G. Zémor, *Hashing with SL_2*, in CRYPTO 1994, Lecture Notes Comp. Sc. **839** (1994), 40–49.
10. X. Wang, Y. L. Yin, and H. Yu, *Finding Collisions in the Full SHA-1*, in CRYPTO 2005, Lecture Notes Comp. Sc. **3621** (2005), 17-36.