FINGERPRINTING BY RANDOM POLYNOMIALS

by

Michael O. Rabin

TR-15-81

FINGERPRINTING BY RANDOM POLYNOMIALS

by

Michael O. Rabin
Department of Mathematics
The Hebrew University of Jerusalem

and

Department of Computer Science
Harvard University

Abstract

Randomly chosen irreducible polynomials $p(t) \in Z_2[t]$ are used to "fingerprint" bit-strings. This method is applied to produce a very simple real-time string matching algorithm and a procedure for securing files against unauthorized changes. The method is provably efficient and highly reliable for every input.

This research was supported in part by NSF Grant MCS-80-12716 at the University of California at Berkeley.

Fingerprinting by Random Polynomials

by

Michael O. Rabin

1. Introduction

Prime numbers are used in several contexts to yield efficient randomized algorithms for various problems [1,4]. In these applications a randomly chosen prime p is used to "fingerprint" a long character-string by computing the residue of that string, viewed as a large integer, modulo p.

This method requires performing fixed-point arithmetic on k-bit integers, where $k = \lceil \log_2 p \rceil$, or at least addition/subtraction on such integers [4].

We propose using a randomly chosen irreducible (prime) polynomial $p(t) \in Z_2[t]$ of an appropriate small degree k instead of the prime integer p. It turns out that it is very easy to effect a random choice of an irreducible polynomial. The implementation of mod $p(t)$ arithmetic requires just length-k shift registers and the exclusive-or operation on k-bit vectors. These operations are fast, involve simple circuits, and in VLSI require little chip area.

The applications given here are: a real-time string matching algorithm; detection of unauthorized changes in a file. The method obviously extends into other areas.

Polynomial modular computations are used in algebraic error correction codes. In these applications one carefully constructs a specific polynomial which is designed to facilitate coding and decoding. Under assumption of a random distribution of all possible error patterns, the code

will detect/correct most occurrences of up to ℓ errors for some fixed ℓ. In the style of [5], we turn the tables around and instead of applying a fixed algorithm to (hopefully) randomly distributed inputs, we construct a randomizing algorithm which is efficient on every input. By randomizing the choice of the irreducible polynomial $p(t)$, we obtain a provably highly dependable and efficient algorithm for every instance of the string matching problem to be solved; we successfully protect every file against any deliberate modification, etc. The mathematically provable efficacy of our method is of particular significance for this last application.

2. Counting and Choosing Irreducible Polynomials

Even though any degree k can be used, implementation is most convenient when k is prime. Thus in practice we can use $k = 17, 19, \ldots, 31, 61$, etc.

Lemma 1. Let k be prime. The number of irreducible polynomials $p(x) = x^k + a_{k-1}x^{k-1} + \ldots + a_0 \in Z_2[x]$, is $(2^k - 2)/k$.

Proof. Let $GF(2^k) = E$ be the Galois field with 2^k elements. Every irreducible polynomial $p(x) \in Z_2[x]$ of degree k has exactly k roots in E, and since $1 < k$ these roots are in $E - Z_2$.

Let $\alpha \in E - Z_2$, and let $1 < m$ be the degree of the irreducible polynomial $q(x) \in Z_2[x]$ that α satisfies. Then $[Z_2(\alpha): Z_2] = m$ and hence $m|k$ (see [3]). Since k is prime, $m = k$. I.e., every $\alpha \in E - Z_2$ is a root of an irreducible polynomial of degree k.

Thus the set $E - Z_2$ is partitioned into sets of k elements, where each set consists of all roots of an irreducible polynomial

$f \in Z_2[x]$ of degree k, and all such polynomials are obtained in this way. It follows that the number of these polynomials is $(2^k-2)/k$. ■

Consider a fixed prime k, say $k = 31$, how do we randomly choose an irreducible polynomial $t^{31} + b_1 t^{30} + \ldots \in Z_2[t]$? We shall do this by calculating within the field $E = GF(2^k)$ of 2^k elements.

To this end we need <u>one</u> irreducible polynomial $f(x) = x^k + a_1 x^{k-1} + \ldots \in Z_2[x]$. The elements of E will be the k-tuples $\gamma = (c_1,\ldots,c_k)$, $c_i \in Z_2$. The addition of two elements is component-wise. To find the product $\gamma\cdot\delta$, where $\delta = (d_1,\ldots,d_k)$, calculate (in $Z_2[x]$) the residue

$$e_1 x^{k-1} + \ldots + e_k \equiv (c_1 x^{k-1} + \ldots + c_k)(d_1 x^{k-1} + \ldots + d_k) \bmod f(x);$$

then $\gamma\cdot\delta = (e_1,\ldots,e_k)$.

Details of how to computationally implement the arithmetic of a finite field can be found in [7], where an efficient method for finding irreducible polynomials of any degree n is also given. We assume that irreducible polynomials $f_2(x)$, $f_3(x)$, ... $f_{31}(x)$,..., of small prime degrees are tabulated once and for all.

We shall now effect a random choice of an irreducible polynomial $p(t) = t^{31} + \ldots \in Z_2[t]$. We use the indeterminate t to distinguish these polynomials from the fixed polynomials $f_2(x)$, $f_3(x)$,... . Choose randomly an element $\gamma = (c_1,\ldots,c_{31}) \in GF(2^{31}) - Z_2$. Namely, randomly generate a sequence of 31 bits and if it happens to be $(0,0,\ldots,0)$ or $(0,0,\ldots,0,1)$ discard it.

The element γ satisfies, by the proof of Lemma 1, a unique irreducible polynomial $p(t) = t^{31} + b_1 t^{30} + \ldots + b_{31} \in Z_2[t]$. To find it, compute in $GF(2^{31})$ (using $f_{31}(x)$) the powers

$\gamma^0 = (0,\ldots,1)$, γ, $\gamma^2,\ldots,$ γ^{31}. These are 32 vectors in Z_2^{31} and are therefore linearly dependent over Z_2. Hence there exist $b_0, b_1,\ldots, b_{31} \in Z_2$, not all 0, so that

$$(1) \qquad b_0\gamma^{31} + b_1\gamma^{30} + \ldots + b_{31}\gamma^0 = (0,\ldots,0).$$

The system (1) of linear equations can be solved by Gaussian elimination which is particularly simple over Z_2. Now b_0 must be 1 because the degree of γ over Z_2 is 31. Thus $t^{31} + b_1t^{30} + \ldots + b_{31}$ is the irreducible polynomial satisfied by γ.

Lemma 2. The above algorithm gives, for a prime degree k, a random choice with equal probabilities, of an irreducible polynomial $p(t) \in Z_2[t]$ of degree k.

Proof. Let $p_1(t),\ldots,p_d(t)$, $d = (2^k-2)/k$, be an enumeration of all the different irreducible polynomials of degree k. As in the proof of Lemma 1, $GF(2^k) - Z_2 = S_1 \cup S_2 \cup \ldots \cup S_d$, where S_i consists of the k roots of $p_i(t)$. The randomly chosen $\gamma \in GF(2^k) - Z_2$ has equal probability of falling within each of the S_j, since all these sets have the same number of elements. Thus with probability $1/d$ we chose $\gamma \in S_i$, in which case $p(t) = p_i(t)$.

3. Arithmetic Modulo p(t)

Henceforth, throughout the rest of this paper, all polynomials will be in $Z_2[t]$. Let $p(t) = t^k + b_1t^{k-1} + \ldots + b_k$ be a polynomial, not necessarily irreducible. For an arbitrary polynomial $g(t)$ denote by $\overline{g}(t)$ the residue $Res(g,p)$ of g when divided by p. A residue mod $p(t)$ is a polynomial $c_1t^{k-1} + \ldots + c_k$ and will be represented by

the bit-vector $(c_1,\ldots,c_k)$. For the small $k \leq 61$ which will actually be used, one or at most two computer words will suffice to store such vectors.

Let $g(t) = x_1 t^{n-1} + x_2 t^{n-2} + \ldots + x_n$. The computation of $\overline{g(t)}$ can be effected in real-time as the coefficient bits $x_1, x_2, \ldots$, are read in. Let $g_i(t) = x_1 t^{i-1} + \ldots + x_i$, $1 \leq i \leq n$, so that $g(t) = g_n(t)$ and $g_{i+1} = g_i \cdot t + x_{i+1}$, $1 \leq i \leq n-1$. Hence

$$\overline{g}_{i+1} = \overline{\overline{g}_i \cdot t} + x_{i+1}, \quad 1 \leq i \leq n-1, \tag{2}$$

and we can work with residues mod $p(t)$ throughout.

Now, if $r(t) = c_1 t^{k-1} + \ldots + c_k$ and $x \in Z_2$, then

$$\overline{r(t)\cdot t}+x = c_2 t^{k-1} + \ldots + c_k t + x + c_1(b_1 t^{k-1} + \ldots + b_k), \tag{3}$$

since $t^k = b_1 t^{k-1} + \ldots + b_k \bmod p$ (recall that $b_i = -b_i$ in Z_2). So that the corresponding vector is

$$(c_2,\ldots,c_k,x) + c_1(b_1,\ldots,b_k). \tag{4}$$

The operation (4) is accomplished by a left-shift of the word $c_1 c_2 \ldots c_k$, introduction of x on the right, followed by the bit-wise exclusive-or with $b_1 b_2 \ldots b_k$ if the overflow c_1 is 1. All this is very fast when implemented by k-length shift registers.

The iteration (2) is now very rapidly calculated with one or two operations for every incoming bit x_{i+1}.

4. String Matching

Let $\pi = x_1 \ldots x_n$, $\tau = y_1 \ldots y_m$, $x_i, y_i \in \{0,1\}$ be bit strings. The string matching problem is to find one or all indices i such that $\pi = y_i y_{i+1} \cdots y_{i+n-1}$. For any such an i we say that the pattern π matches with the ith substring of length n of the text τ, or more shortly that a match occurs. In the style of [6] we construct a randomized algorithm which for every π and τ and given $0 < \varepsilon$ will solve the string matching problem with probability of error smaller than ε.

Let k be the smallest prime such that $k > \log_2(nm\varepsilon^{-1})$. In all practical applications $k = 61$ or even $k = 31$ will be ample, so that k-bit words will fit into one or at most two computer words. Define

$$a(t) = x_1 t^{n-1} + \ldots + x_n, \qquad a_i(t) = y_i t^{n-1} + \ldots + y_{i+n-1}, \quad 1 \leq i \leq m-n+1.$$

Obviously, a match for index i is equivalent to $a(t) = a_i(t)$.

Choose randomly an irreducible polynomial $p(t) = t^k + b_1 t^{k-1} + \ldots + b_k$. Denote, as in Section 2, by $\bar{g}(t)$ the residue modulo $p(t)$ of the polynomial $g(t)$.

Compute $\bar{a}(t)$, $\overline{t^{n-1}}$, $\bar{a}_1(t)$. This is done in real time. If $\bar{a}(t) = \bar{a}_1(t)$ output "match for index 1." At the ith stage we have stored $\bar{a}(t)$, $\overline{t^{n-1}}$, $\bar{a}_i(t)$, $p(t)$, i. Each of the first four is a k-bit word and i is a pointer into the text τ. Compute

$$\bar{a}_{i+1}(t) = (\bar{a}_i(t) + y_i \overline{t^{n-1}}) \cdot t + y_{i+n} \bmod p(t).$$

This is done as in Section 2, except that if $y_i = 1$ we start by doing an exclusive-or of the words corresponding to $\bar{a}_i(t)$ and $\overline{t^{n-1}}$. If $\bar{a}(t) = \bar{a}_{i+1}(t)$ then output "matching for i+1." The updating from $\bar{a}_i(t)$ to $\bar{a}_{i+1}(t)$ requires a fixed number of operations per text symbol, i.e., is done in real time.

Theorem 3. For any pattern $\pi = x_1 \ldots x_n$ and text $\tau = y_1 \ldots y_m$, if $k > \log_2(nm\varepsilon^{-1})$ then the probability that the above randomized algorithm will produce one or more errors is smaller than ε. Furthermore the algorithm will discover all actual matches.

Proof. The second assertion is obvious, since $a(t) = a_i(t)$ implies $\bar{a}(t) = \bar{a}_i(t)$ for every modulus $p(t)$.

Consider the product

$$H(t) = \prod_{a(t) \neq a_i(t)} (a(t) - a_i(t)). \tag{5}$$

If for a polynomial $p(t)$ the algorithm outputs for some i "match for index i" even though there is no match, then we must have $\bar{a}(t) = \bar{a}_i(t)$ and $a(t) \neq a_i(t)$. Hence $p(t)$ divides $H(t)$, which we write as $p(t) | H(t)$.

Conversely, if for an irreducible polynomial $p(t) | H(t)$ then $p(t)$ must divide some factor $a(t) - a_i(t)$ of $H(t)$. But then $\bar{a}(t) = \bar{a}_i(t)$ while $a(t) \neq a_i(t)$, so that if this $p(t)$ is used in the algorithm it produces a wrong output.

Let $p_1(t), \ldots, p_\ell(t)$ be a list of all the irreducible polynomials of degree k which divide $H(t)$. Then their product $P(t)$ also divides $H(t)$. Consequently

$$\ell k = \deg P \leq \deg H \leq nm. \tag{6}$$

The right hand inequality holds because $H(t)$ is a product of at most $m-n$ factors each of degree at most n. From (6) we have $\ell \leq nm/k$. Thus for every pair π, τ of pattern and text, at most nm/k irreducible polynomial $p(t)$ of degree k will produce an error when used in the algorithm.

By Lemma 1, the total number of all irreducible polynomials of degree k is $N = (2^k-2)/k \sim 2^k/k$. Since $k > \log_2(nm\varepsilon^{-1})$ we have $N > nm\varepsilon^{-1}/k$. The probability that for a randomly chosen $p(t)$ the algorithm will produce an error, equals the number ℓ of error producing polynomials divided by the number N of all irreducible polynomials of degree k. Thus we get

$$\Pr(p(t)|\text{Algorithm produces error}) \leq (nm/k)/(nm\varepsilon^{-1}/k) = \varepsilon. \quad \blacksquare$$

The only part of the string matching algorithm which is not strictly real-time is the random choice of $p(t)$ of degree k. Since $k = 3 \log_2 m$ will give high reliability, the choice of $p(t)$ will involve $(3 \log_2 m)^2$ exclusive-or operations on k-bit vectors (for the Gaussian elimination), and is negligible as compared to the $|\tau| = m$ steps of the algorithm. Alternatively, one can prepare in advance a list of randomly chosen polynomials from which $p(t)$ will be selected in one step.

Illustrating Theorem 3, assume that $n = |\pi| = 1000$, $m = |\tau| = 10^6$, $\varepsilon = 2^{-30}$. Then $k = 61$ satisfies $k > \log_2(nm\varepsilon^{-1})$ so that by using polynomials of degree 61, i.e., 61-bit words, the probability of error for every particular pair of pattern/text is at most $2^{-30} \sim 10^{-9}$.

If we are worried about errors having probability 10^{-9} we could use a larger k. If, however, we do not want to move to larger word-length then we can randomly choose two irreducible polynomials $p_1(t)$, $p_2(t)$ of degree 61 and run the algorithm twice, either in parallel or by interleaving steps, one time with p_1 and another time with p_2. Since the error probabilities are independent, the probability of a wrong output is 10^{-18}.

The method of computing residues modulu a polynomial can be combined with anyone of the algorithm in [4] to produce efficient solutions to the other pattern matching problems such as two and higher-dimensional

matching problems, solved there. See [2,5] for non-randomized string matching algorithms.

5. Protection of Files

We shall use irreducible polynomials to "fingerprint" files so that any unauthorized change will be detected with a very high probability. Furthermore, updating the fingerprint when the file is locally modified in an authorized change, is very simple.

The method will use a randomly chosen irreducible polynomial $p(t)$ of an appropriate degree k ($k = 61$ will be ample). This polynomial, actually k-bit word, is chosen by the guardian of the filing system's security and will be kept secret. A file F will be fingerprinted and the fingerprint $\overline{F}(t)$, a polynomial of degree $k-1$, will be securely kept by the guardian. To check whether F was tempered with, the file is again fingerprinted and the current fingerprint $\overline{F}_1$ is compared with the stored value $\overline{F}$. If $\overline{F}_1 \neq \overline{F}$ then the guardian knows that the file was changed without authorization.

The method for updating the fingerprint $\overline{F}$ when authorized changes are made is detailed later on.

The same polynomial p will be employed to fingerprint all the files $F_1, F_2, \ldots$ in the system. An arrangement must be made for the guardian to securely store p, and the fingerprints $\overline{F}_1, \overline{F}_2, \ldots$. The fingerprinting and updating computations must be performed securely so that none other than the guardian can access p or the $\overline{F}$. In view of the simple computations involved in the fingerprinting and the small amount of information requiring secure storage, namely just one word per file, it is feasible to construct a special purpose "box" for handling the security check algorithm. The box is attached to the system and the pages of the file pass through it, but none of the secret numbers ever leaves the box.

Assume that the file F has m pages $P_0, \ldots, P_{m-1}$ and that each page stores up to n bits. If $P_i = x_{i1}x_{i2}\ldots x_{in}$, define

$$P_i(t) = x_{i1} + \ldots + x_{in}t^{n-1},$$

$$F(t) = P_0(t) + P_1(t)t^n + \ldots + P_{m-1}(t)t^{n(m-1)}$$

Choose randomly an irreducible polynomial $p(t)$ of prime degree k. Denote, as before, $\overline{P}_i(t) = \text{res}\,(P_i(t),p(t))$, then $\deg \overline{P}_i = k-1$. As in Section 2, this $\overline{P}_i(t)$ is very rapidly calculable. If $r(t) = \overline{t^n}$ then the residue $\overline{F}(t) = \text{res}\,(F(t),p(t))$ is given by

$$\overline{F}(t) = \overline{P}_0(t) + \overline{\overline{P}_1(t)\cdot r(t)} + \ldots + \overline{\overline{P}_{m-1}(t)\cdot r(t)^{m-1}} \tag{7}$$

The fingerprint for the whole file F will be $\overline{F}(t)$. This residue is also rapidly computable. Namely, calculate $r(t) = t^n \bmod p(t)$ by any of the rapid exponentiation algorithms. Read P_0 in and compute $\overline{P}_0(t)$ in real-time. Compute $\overline{P}_1(t)$, $\overline{\overline{P}_1(t)\cdot r(t)}$, $\overline{r(t)^2}$, and $\overline{P}_0(t) + \overline{\overline{P}_1(t)\cdot r(t)}$, etc. The general step involves computation of $\overline{P}_i(t)$ and two multiplications modulu $p(t)$ of k-1-degree polynomials. The latter multiplications require little time compared to the computation of $\overline{P}_i(t)$, even if the straightforward method is used.

Formula (7) suggests how updating of $\overline{F}(t)$ is done locally. Assume that $\overline{F}(t)$, $p(t)$, $r(t)$ are securely stored and that page P_i is to be modified into P'_i. As P_i is read in, the residue $\overline{P}_i(t)$ is recomputed. Also compute $r_i(t) = \text{res}\,(r(t)^i,p)$, this again by any of the rapid exponentiation algorithms. After P'_i is produced, compute $\overline{P'_i}(t)$. The updated fingerprint is

$$\overline{F'}(t) = \overline{F}(t) + \text{res}\,(r_i(t)(\overline{P}_i + \overline{P'}_i),p).$$

To illustrate the reliability of the security-check algorithm let us

work out an example rather than write general formulas.

Proposition. If a file F consists of a thousand pages each containing up to 4000 bytes (32,000 bits) and we use polynomials of degree 61. then the probability of any unauthorized change in F to go undetected is at most 2^{-46}, i.e., completely negligible.

Proof. Assume that page P_i was changed into page P_i', let $\overline{F}(t)$ denote, as before, the fingerprint (7) of the original file F and let $\overline{F'}(t)$ denote the fingerprint of the modified file F'. The change will be detected if $\overline{F} \neq \overline{F}$.

We have deg $F(t)$ = deg $F'(t) = 32 \cdot 10^6$ so that deg $(F - F') \leqslant 32 \cdot 10^6 \leqslant 2^{25}$. If $F(t) - F'(t) \neq 0$ then it has at most $2^{25}/61$ irreducible divisors of degree 61. Since there are $2^{61}/61$ irreducible polynomials of degree 61, the probability that for a randomly chosen such $p(t)$ we have $\overline{F'} = \overline{F}$ is at most $(2^{25}/61)/(2^{61}/61) = 2^{-46}$.

Remark. We can dispense with the assumption that the file F is a sequence of numbered pages and consider F as a set $Q = \{P_0,\ldots,P_{m-1}\}$ of pages. We shall use two irreducible polynomials, $p(t)$ of degree k and $q(t)$ of degree $k_1 \geqslant k$. Denote by $n(P)$ the sequence of coefficient bits of $res(P(t),p(t))$. View $n(P)$ as an integer, then $n(P) < 2^k$.

Define

$$Q(t) = \prod_{P \in Q} t^{n(P)}. \qquad Q_1(t) = res\,(Q(t),q(t));$$

$Q_1(t)$ will be the fingerprint of Q. The computation of Q_1 as well as the updating follow the previous pattern. The set formulation has the

further advantage that the fingerprint of the union $Q \cup Q'$ of two disjoint files is simply $Q_1(t) + Q'_1(t)$. We omit the detailed determination of degrees k and k_1 which will ensure a desired probability ε for a change to go undetected.

BIBLIOGRAPHY

1. Freivolds, R. Probabilistic machines can use less running time. Information Processing 77, Gilchrist, B. (ed.), pp. 839-842.

2. Galil, Z. and J. Seiferas. Time-space optimal string matching. 13th Annual ACM STOC (1981), pp. 106-113.

3. Herstein, I.N. Topics in Algebra, 2nd Edition. Xerox College Publishing, Lexington, MA, 1975.

4. Karp, R.M. and M.O. Rabin. Efficient randomized pattern-matching algorithms. Submitted for publication.

5. Knuth, D.E., Morris, J.H. and V.R. Pratt. Fast pattern matching in strings. SIAM J. on Computing, Vol. 6 (1977), pp. 323-350.

6. Rabin, M.O. Probabilistic algorithms. In Algorithms and Complexity, Recent Results and New Directions, J.F. Traub (Ed.). Academic Press, New York, 1976, pp. 21-40.

7. Rabin, M.O. Probabilistic algorithms in finite fields. SIAM J. on Computing, Vol. 9 (1980), pp. 273-280.