Some applications of Rabin's fingerprinting method*

Andrei Z. Broder†

Abstract

Rabin's fingerprinting scheme is based on arithmetic modulo an irreducible polynomial with coefficients in $\mathbf{Z}_2$. This paper presents an implementation and several applications of this scheme that take considerable advantage of its algebraic properties.

1 Introduction

Fingerprints are short tags for larger objects. They have the property that if two fingerprints are different then the corresponding objects are certainly different and there is only a small probability that two different objects have the same fingerprint. (The latter event is called a collision.)

More precisely, a fingerprinting scheme is a certain collection of functions $\mathcal{F} = \left\{ f : \Omega \to \{0,1\}^k \right\}$, where Ω is the set of all possible objects of interest and k is the length of the fingerprint, such that, for any choice of a fixed set $S \subset \Omega$ of n distinct objects, if f is chosen uniformly at random in $\mathcal{F}$, then with high probability

$$|f(S)| = |S|.$$

In other words, if an adversary chooses a set $S \subset \Omega$ of n distinct objects, and we choose $f \in \mathcal{F}$ uniformly at random then

$$f(A) \neq f(B) \Longrightarrow A \neq B \tag{1}$$

$$\Pr(f(A) = f(B) \mid A \neq B) = \text{very small} \tag{2}$$

(The requirements and the model used here, are similar to those for "universal hashing" [1]; however the emphasis and the relation between n and k are different: for hashing we are interested in bounding the number of collisions, and typically n is a small fraction of 2^k; for fingerprinting we want to avoid collisions altogether, and we must take $n \ll 2^k$.)

*Published in R. Capocelli, A. De Santis, U. Vaccaro (eds), *Sequences II: Methods in Communications, Security, and Computer Science*, Springer-Verlag, 1993.

†DEC Systems Research Center, 130 Lytton Ave, Palo Alto, CA.

2 Rabin's fingerprinting scheme

The following fingerprinting scheme for strings is due to Michael Rabin [4].

Let $A = (a_1, a_2, \ldots, a_m)$ be a binary string. We assume that $a_1 = 1$. (In certain application this assumption might be false, hence the implementation must actually prefix every string to be fingerprinted by a 1. We ignore this technicality for the rest of the paper.) We associate to the string A a polynomial $A(t)$ of degree $m-1$ with coefficients in $\mathbf{Z}_2$,

$$A(t) = a_1 t^{m-1} + a_2 t^{m-2} + \cdots + a_m. \tag{3}$$

Let $P(t)$ be an irreducible polynomial of degree k, over $\mathbf{Z}_2$. (Such a polynomial can be easily found. See [3].) Having fixed P, we define the fingerprint of A to be the polynomial

$$f(A) = A(t) \bmod P(t). \tag{4}$$

Assume that an adversary chooses a set S of n distinct binary strings. After the adversary chooses S, we choose uniformly at random an irreducible polynomial P of degree k. We claim that for a proper choice of k the probability (over the random choices of P) that there exists a pair of distinct strings in S that have the same fingerprint, is extremely small.

Indeed, consider the product

$$Q = \prod_{\{A,B\}} (A(t) - B(t)), \tag{5}$$

taken over all the unordered pairs $A, B \in S$, with $A \neq B$. This product is a polynomial with coefficients in $\mathbf{Z}_2$. Its degree can be bound by

$$\begin{aligned} \deg Q &\leq \sum_{\{A,B\}} \max\{\deg A(t), \deg B(t)\} \\ &\leq \sum_{\{A,B\}} (\deg A(t) + \deg B(t)) \leq n \sum_{A \in S} |A|. \end{aligned} \tag{6}$$

In particular if all the strings in S have length less than m, then the degree of Q is less than $n^2 m$.

If there are two distinct strings, $A, B \in S$, such that $f(A) = f(B)$, it means that P divides $A(t) - B(t)$ and hence P divides Q. The adversary, by his choices of strings, has fixed a particular Q. This Q can not have more than $\deg(Q)/k$ irreducible factors of degree k. However the total number of irreducible polynomials of degree k with coefficients in $\mathbf{Z}_2$ is greater than $(2^k - 2^{k/2})/k$. Hence the probability that a randomly chosen irreducible polynomial of degree k divides Q, which is the same as the probability that two distinct strings have the same fingerprint, is less than

$$\frac{\deg Q}{k} \cdot \frac{k}{2^k - 2^{k/2}} \approx \frac{\deg Q}{2^k} \leq \frac{nm^2}{2^k}. \tag{7}$$

For instance, if the adversary chooses any collection of 2^{15} binary strings of total length less than 2^{25} bits, and we choose P uniformly at random among the irreducible polynomials of degree 64 over $\mathbf{Z}_2$, the probability that we made a bad choice of P is less than 2^{-23}.

3 Properties of Rabin's scheme

Besides satisfying the basic conditions (1) and (2), Rabin's scheme has additional properties that are useful in some applications and facilitate its implementation. (In this section all arithmetic operations are in $\mathbf{Z}_2$.)

1. At the hardware level the representation of the string A and the polynomial $A(t)$ with coefficients over $\mathbf{Z}_2$ is identical. The basic operations with polynomials have simple implementations: addition is equivalent with bit-wise exclusive or, and multiplication by t is equivalent with shift left one bit.

2. Fingerprinting is distributive over addition (in $\mathbf{Z}_2$):

$$f(A+B) = f(A) + f(B). \tag{8}$$

3. Fingerprints can be computed in linear time. Consider the bit string A $= [b_1, \ldots, b_l]$; If

$$\begin{aligned} &f([b_1, \ldots, b_l]) \\ &\quad = (b_1 t^{l-1} + b_2 t^{l-2} + \ldots + b_l) \bmod P(t) \\ &\quad = r_1 t^{k-1} + r_2 t^{k-2} + \ldots + r_k \end{aligned}$$

   then

$$\begin{aligned} &f([b_1, \ldots, b_{l+1}]) \\ &\quad = (f(b_1, \ldots, b_l)t + b_{l+1}) \bmod P(t) \\ &\quad = r_2 t^{k-1} + r_3 t^{k-2} + \ldots + r_k t + b_{l+1} \\ &\qquad + (r_1 t^k) \bmod P(t) \end{aligned}$$

   Observe that

$$t^k \bmod P(t) = t^k - P(t) = P(t) - t^k,$$

   so $t^k \bmod P$ is equivalent to P with the leading coefficient removed.

   At the hardware level, the fingerprint of A can be kept in a shift register. Computing the fingerprint of A extended by b_l consists of one shift left operation with b_l as input bit and r_1 as output bit, and then, conditioned upon $r_1 = 1$, a bit-wise exclusive or operation, the second operand being P with the leading coefficient removed. (The next section discusses software implementation for a typical 32 bit word computer.)

4. More generally, the fingerprint of the concatenation of two strings can be computed via the equality

$$f(\text{concat}(A, B)) = f(\text{concat}(f(A), B)). \tag{9}$$

5. If we are given $f(A)$ and $f(B)$, and the length l of B then

$$\begin{aligned} f(\text{concat}(A, B)) &= A(t) * t^l + B(t) \bmod P(t) \\ &= f(f(A) * f(t^l)) + f(B) \end{aligned} \tag{10}$$

(In practical applications it is useful to precompute $f(t^l)$ for a suitable range of values.)

4 Implementation issues

A straightforward implementation of the linear algorithm described above in software for a 32 bit per word computer requires two word bit shifts and two word xors (exclusive ors) *per bit*, or 16 shifts and 16 xors per byte. This is too slow for many applications.

However it is easy to process more than one bit at a time, using precomputed tables. A good trade-off is to process 32 bits at a time divided into four bytes. It is convenient to take k (that is, the degree of P, and thus the length of the fingerprint) to be a multiple of 32, in order to have fingerprints that consist of an integral number of words. Below we take $k = 64$.

The notations $W_{b1}, \ldots, W_{b4}$ refer to the four bytes of the 32-bit word W, and the notations $W_1, \ldots, W_{32}$ refer to the 32 bits of of the same word.

The input is the string $A[1], \ldots, A[s]$ where each $A[i]$ is a 32-bit word.

The final algorithm is

$W[1] \leftarrow 0$; $W[2] \leftarrow 0$;
for $s = 1, \ldots, m$ **do**
  $h, i, j, l \leftarrow W[1]_{b1}, W[1]_{b2}, W[1]_{b3}, W[1]_{b4}$;
  $W[1] \leftarrow W[2]$ **xor** $TA[h, 1]$ **xor** $TB[i, 1]$ **xor** $TC[j, 1]$ **xor** $TD[l, 1]$;
  $W[2] \leftarrow A[s]$ **xor** $TA[h, 2]$ **xor** $TB[i, 2]$ **xor** $TC[j, 2]$ **xor** $TD[l, 2]$;
od

The invariant maintained by this algorithm is that

$$\begin{aligned} & W[1]_1 t^{63} + W[1]_2 t^{62} + \cdots + W[1]_{32} t^{32} \\ & + W[2]_1 t^{31} + \cdots + W[2]_{32} \\ & = f((A[1], A[2], \ldots, A[s])) \bmod P(t). \end{aligned}$$

The tables TA, TB, TC, and TD are defined by

$$\begin{aligned} & TA[i,1]_1 t^{63} + TA[i,1]_2 t^{62} + \cdots + TA[i,1]_{32} t^{32} \\ & + TA[i,2]_1 t^{31} + \cdots + TA[i,2]_{32} \\ & = \quad i_1 t^{95} + i_2 t^{94} + \cdots + i_8 t^{88} \bmod P(t); \end{aligned}$$

$$\begin{aligned} & TB[i,1]_1 t^{63} + TB[i,1]_2 t^{62} + \cdots + TB[i,1]_{32} t^{32} \\ & + TB[i,2]_1 t^{31} + \cdots + TB[i,2]_{32} \\ & = \quad i_1 t^{87} + i_2 t^{86} + \cdots + i_8 t^{80} \bmod P(t); \end{aligned}$$

$$\begin{aligned} & TC[i,1]_1 t^{63} + TC[i,1]_2 t^{62} + \cdots + TC[i,1]_{32} t^{32} \\ & + TC[i,2]_1 t^{31} + \cdots + TC[i,2]_{32} \\ & = \quad i_1 t^{79} + i_2 t^{78} + \cdots + i_8 t^{72} \bmod P(t); \end{aligned}$$

$$\begin{aligned} & TD[i,1]_1 t^{63} + TD[i,1]_2 t^{62} + \cdots + TD[i,1]_{32} t^{32} \\ & + TD[i,2]_1 t^{31} + \cdots + TD[i,2]_{32} \\ & = \quad i_1 t^{71} + i_2 t^{70} + \cdots + i_8 t^{64} \bmod P(t), \end{aligned}$$

where $0 \le i < 256$ and i_j denotes the j-th bit in the binary representation of i.

This algorithm requires 2 xors and 1 table indexing per byte. For most applications this is acceptable, particularly if the inner loop is coded in assembly language. For instance in our implementation the time required to fingerprint the contents of a file is under 3% of the time required to read the file in memory.

Finding a random irreducible polynomial $P(t)$ is simple (see [4]) and computing the associated tables is straightforward. However if all we want is to fingerprint relatively short strings, the overhead is prohibitive. Nevertheless in typical applications it is not necessary to pick a new random $P(t)$ every time the fingerprinting package of procedures is used. Instead, the polynomial $P(t)$ can be "wired-in," thus saving the computation of the tables $TA, \ldots, TD$. The idea is to assume that the virtual adversary has committed to all the strings that will fingerprinted by the package in the future in a certain context, and use equation (7) to bound the probability of collision. For instance, we can imagine that the adversary has committed to all variable names used in all Modula-2 programs in the next twenty years at DEC-SRC. We want to avoid fingerprint collisions within the same module. Assuming that the typical module has 2^9 variable names of 2^6 bits average length, the probability of collision within one module is less than $2^{25}/2^k$. If we assume that the number of modified modules is 2^{10} per day and that we use 64 bits fingerprints, the probability of even one collision in 20 years is less than 2^{-16}, which is negligible compared to the likelihood that the compiler produces erroneous results from other causes.

However, in order to maintain the fiction that the adversary has committed to all the future strings, it is crucial to ensure that no string depends on our choice for

$P(t)$. In particular, we must maintain the rule that no fingerprinting of strings that contain fingerprints is allowed.

If fingerprints are computed piecemeal via equations (9) and (10) then it makes sense to make the length of the underlying string a part of the fingerprint, and have the fingerprint package keep track of it. Since

$$x^{2^k-1} \equiv 1 \bmod P(t),$$

it suffices to record the length modulo $2^k - 1$ even if larger lengths might appear. (However notice that in this case the bound (7) becomes useless.)

5 Applications

Below there are several fingerprinting usages that take advantage of the algebraic properties of Rabin's scheme. For more applications see [2] and [5].

5.1 Fingerprinting dags

This application of fingerprints was motivated by *Vesta*, a software development environment in construction at DEC Systems Research Center.

Vesta needs to make identifying tags for derived objects, that is, files mechanically derived from other files (e.g. via compilation or linking). Essentially, the relations between all the derived objects (ever) and all the sources can be put in the form of a gigantic dag (directed acyclic graph) with labeled edges. The vertices that have no predecessors are sources. To any vertex we can associate an edge-labeled tree: The tree of a vertex is formed in the obvious manner from the trees of its direct predecessors.

Vesta requires the vertices of the dag to have identical tags if and only if their associated trees are isomorphic. (In other words, two derived objects have the same tag if and only if they were derived in the same way from the same sources.)

5.1.1 Simplistic solution

It is straightforward to encode these trees as strings; the string for a parent is the concatenation of the strings corresponding to its children separated by labels and delimiters. We can use as a tag the fingerprint of the string. Fortunately it is not necessary to keep the strings themselves around. The tag of the parent can be computed from the tags of the children, provided that the length of the strings involved is known.

More precisely let A represent a string and α its associated polynomial. Let "$\|$" denote concatenation. Then ignoring delimiters

$$A(\text{parent}) = A(\text{child1}) \parallel A(\text{child2})$$

or

$$\alpha(\text{parent}) = \alpha(\text{child1}) * t^{\text{length}(A(\text{child2}))} + \alpha(\text{child2}),$$

and hence

$$f(\text{parent}) = (f(\text{child1}) * t^{\text{length}(A(\text{child2}))} + f(\text{child2})) \bmod P(t).$$

The problem with this approach is that the length of the strings involved might be extremely large and hence the formula (7) does not give any meaningful bound on the probability of collision. (The length of the describing string may grow exponentially in the size of the dag.)

5.1.2 General solution

Assume that an adversary chooses a dag G on n nodes. He also associates source strings to the nodes with no predecessors. After the adversary chooses G, we choose uniformly at random an irreducible polynomial $P(t)$ of degree k and a random permutation T of the set with 2^k elements. (For the *Vesta* application described above, a reasonable choice is $k = 96$.)

Let f be the Rabin's fingerprinting function. The tag of a node is computed as follows:

$$Tag(\text{parent}) = T(f(\text{Label1} \parallel Tag(\text{child1}) \parallel \text{Label2} \parallel Tag(\text{child2}) \parallel \ldots)).$$

For source nodes:

$$Tag(\text{source}) = T(f(\text{text}))$$

Observe that each node has a fingerprint and a tag.

Because T is chosen uniformly at random we can imagine that we construct T iteratively as follows: Start by assigning each sink (i.e. each *Vesta* source) in the dag a distinct random tag. Process the nodes of the dag in reverse topological order. If the result of f at a certain node v (i.e. either $f(\text{text})$ or $f(\text{label1} \parallel \ldots)$) is a new value x, we let $T(x)$ be a new random tag; otherwise T(x) is already determined.

Alternatively we can think that all these random tag are chosen simultaneously. But with high probability for this assignment of tags all the node fingerprints are different: we are fingerprinting n strings of (reasonable) average length m so the formula (7) applies and with high probability there are no collisions.

This would be a perfect solution except that generating a truly random permutation on 2^k elements requires time exponential in k. There are many ways of doing pseudo-random permutations: In our implementation we use first a non-linear mapping (each byte of the fingerprint is mapped via a random permutation of the set $\{0, \ldots, 255\}$) followed by an arithmetic mapping (the fingerprint is viewed as a vector of numbers in $\mathbf{Z}_{2^{32}}$ and multiplied by a non-singular matrix).

5.2 Fingerprinting subsets

Consider a programming application where there is a need to fingerprint n subsets $A_1, \ldots, A_n$, of a given ground set $\Omega = \{\omega_1, \omega_2, \ldots, \omega_r\}$. (An example of such an application is given below.) Associate to each ω_i a distinct irreducible polynomial $R_i(t)$ of degree m and define

$$R(A) = \prod_{\omega_i \in A} R_i.$$

Let the fingerprint of A be defined by

$$f(A) = R(A) \bmod P(t) = \prod_{\omega_i \in A} R_i \bmod P(t), \tag{11}$$

where as before $P(t)$ is a random irreducible polynomial of degree k.

With this setup we obtain that

$$\Pr(\exists i, j \text{ such that } A_i \neq A_j \text{ and } f(A_i) = f(A_j)) \approx \frac{n^2 rm}{2^k}.$$

It is also possible to take the R_i to be just random polynomials of degree k. The increase in the probability of collision is only $1/2^k$ per pair, so the above formula becomes

$$\Pr(\exists i, j \text{ such that } A_i \neq A_j \text{ and } f(A_i) = f(A_j)) \approx \frac{n^2 rk}{2^k}.$$

Note that if the fingerprints have to be computed from scratch, then the bit complexity of the approach above, $O(rk^2)$, is worse than simply representing each set as a sorted list and computing the fingerprint of the list in the obvious manner, in $O(rk \log r)$ bit operations.

Yet another possibility is to define the fingerprint of a set via

$$f(A) = \sum_{\omega_i \in A} t^i \bmod P(t). \tag{12}$$

This is the most efficient approach when the ground set is explicitly available and its elements can be easily numbered.

However there are situations when the sets encountered are derived from previously fingerprinted sets and when the ground set is not fully known at once. We can then take advantage of the following relations

- If $C = A \cup B$ for $A \cap B = \emptyset$, then

$$f(C) = (f(A) * f(B)) \bmod P.$$

- If $C = A \setminus B$ for $B \subset A$, then

$$f(C) = (f(A) * f(B)^{-1}) \bmod P,$$

  where the inversion is in the field of residues modulo P.

As an example consider fingerprinting the spanning trees of a graph $G(V, E)$. Each spanning tree can be viewed as a set of $|V|-1$ edges. If we add an edge e to a tree T it forms a cycle with the tree edges; the cycle can be broken by removing a suitable edge f thus obtaining a new spanning tree, T'. (If the edges e and f are chosen at random among eligible edges, the process becomes a Markov Chain on the set of spanning trees. We were interested in fingerprinting these trees in order to check certain statistics.)

Using the definition (11) we have

$$f(T') = (f(T) * R_e * R_f^{-1}) \bmod P(t),$$

where R_e and R_f are irreducible (or random) polynomials associated to e and f respectively. Note that R_f^{-1} (or even all pairs $R_e * R_f^{-1}$) can be precomputed.

For definition (12) we have

$$f(T') = (f(T) + t^{N(e)} - t^{N(f)}) \bmod P(t),$$

where $N(\cdot)$ is just a numbering of the edges. In this case this approach is more efficient, in particular if we precompute $t^i \bmod P(t)$ for $i = 1, \ldots, r$.

5.3 Fingerprinting binary trees

We can fingerprint binary trees using one of the two methods described in section 5.1. For small trees, Jim O'Toole has found another approach (unpublished) that avoids using delimiters and keeping track of the lengths of the strings involved.

Consider two strings A and B. Pad the shorter string with 0's to the left, to ensure equal length. Define the *intercalation* of A and B to be the string obtained by alternating the bits of A and B, that is, if $A = (a_1, \ldots, a_m)$ and $B = (b_1, \ldots, b_m)$ then

$$S(A, B) = (a_1, b_1, a_2, b_2, \ldots, a_m, b_m). \tag{13}$$

Let C be the string associated to the parent of two nodes with associated strings A and B. The encoding rule is

$$C = S(A, B), \tag{14}$$

and the fingerprint of the tree rooted at the parent

$$f(C) = C(t) \bmod P(t). \tag{15}$$

In order to make this encoding unambiguous we enlarge the input tree to a full binary tree (that is, each node has degree either 2 or 0) by adding dummy leaves. We associate the string 11 to the true leaves and 10 to the dummy leaves. It is easy to check that with these initial conditions, equation (14) defines an unambiguous encoding of binary trees.

Note that (14) implies that

$$C(t) = A(t^2) * t + B(t).$$

But in $\mathbf{Z}_2$ we have $A(t^2) = (A(t))^2$, and therefore

$$\begin{aligned} C(t) \bmod P(t) &= \big((A(t) \bmod P(t))^2 * t + B(t) \bmod P(t)\big) \bmod P(t) \\ &= S(f(A), f(B)) \bmod P(t) \end{aligned}$$

In other words the fingerprint of C is obtained by fingerprinting the intercalation of the fingerprints of A and B.

The disadvantage of this approach is that the length of the encoding might be exponential in the size of the tree, while the length of the encoding presented in section 5.1.1 is only linear in the size of the tree.

Acknowledgement

I am indebted to Jim O'Toole for allowing me to present here his results on fingerprinting binary trees and for several stimulating discussions.

References

[1] L. Carter and M. Wegman, Universal Classes of Hash Functions. *JCSS*, 18:143–154, 1979.

[2] R. M. Karp and M. O. Rabin, Efficient randomized pattern matching algorithms. Center for Research in Computing Technology, Harvard University, Report TR-31-81, 1981.

[3] M. O. Rabin, Probabilistic algorithms in finite fields. *SIAM J. of Computing*, 9:273–280, 1980.

[4] M. O. Rabin, Fingerprinting by random polynomials. Center for Research in Computing Technology, Harvard University, Report TR-15-81, 1981.

[5] M. O. Rabin, Discovering repetitions in strings. In A. Apostolico and Z. Galil, (eds.), "Combinatorial Algorithms on Words," Springer-Verlag, 279–288, 1985.