Submitted to Journal of Cryptology, March 2000.

FLOATING-POINT ARITHMETIC AND MESSAGE AUTHENTICATION

DANIEL J. BERNSTEIN

ABSTRACT. There is a well-known class of message authentication systems guaranteeing that attackers will have a negligible chance of successfully forging a message. This paper shows how one of these systems can hash messages at extremely high speed—much more quickly than previous systems at the same security level—using IEEE floating-point arithmetic. This paper also presents a survey of the literature in a unified mathematical framework.

1. INTRODUCTION

Let $m = (m_0, m_1, \dots, m_{\ell-1})$ be a sequence of integers in $[-2^{31}, 2^{31} - 1]$. For any integer r define $h_r(m) = (r^{\ell+1} + m_0 r^{\ell} + m_1 r^{\ell-1} + \cdots + m_{\ell-1} r) \bmod (2^{127} - 1)$.

The point of this paper is an extremely fast method to compute $h_r(m)$, given m, after some precomputation depending only on r. The method is described in a bottom-up fashion in sections 2 through 6. A variant appears in section 7. My current implementation achieves the following speeds in cache:

ℓ	16	32	64	128	256	512
Bytes in m	64	128	256	512	1024	2048
Pentium cycles	550	825	1340	2480	4667	9275
Pentium-II cycles	566	792	1303	2406	4527	8850
Alpha-21264 cycles	600	864	1375	2561	4814	9432
$\approx$ UltraSPARC-I cycles	570	762	1148	2111	3818	7438
$\approx$ HP-8200 cycles						5000

See `http://cr.yp.to/hash127.html` for the latest results.

Message authentication. One can authenticate a single message m as follows. The sender and receiver share secret integers r and k. The sender computes an authenticator $s = (k + h_r(m)) \bmod (2^{127} - 1)$ and sends (m, s). The receiver verifies (m, s) by recomputing s. An attacker, given (m, s), has a negligible probability of successfully forging a different message. See section 8 for further discussion.

One can authenticate many messages by combining h_r with a block cipher such as Rijndael: the authenticator of a message m is $\text{Rijndael}_k(h_r(m))$. An attacker who can break this system can break Rijndael with similar success and similar speed. See section 9 for further discussion.

There are many functions that can be used in place of h_r. Section 10 presents a survey of the literature in a unified mathematical framework. What distinguishes

Date: 20000321.
1991 *Mathematics Subject Classification.* Primary 11Y16; Secondary 65G50, 94A60.
The author was supported by the National Science Foundation under grant DMS–9600083.

h_r is its speed: it is the first high-security system that offers better speed than the MD5-based systems in common use today.

Other applications. Checking a public-key signature in the system described in [20] means checking that $a^2 = bc + de$, given integers a, b, c, d, e. One way to do this, much faster than computing $a^2 - bc - de$, is to select a prime number p in $[2^{120}, 2^{128}]$ and see whether $((a \bmod p)^2 - (b \bmod p)(c \bmod p) - (d \bmod p)(e \bmod p)) \bmod p$ is zero; if p is kept secret then there is a negligible chance of anyone else being able to find a, b, c, d, e with $a^2 - bc - de$ nonzero but divisible by p. The algorithm described in this paper can easily be adapted to the problem of reducing large integers to (reasonably) small integers modulo p. Similar comments apply to other applications in which one must check the result of a series of ring operations.

The same techniques can be applied to other problems in multidigit arithmetic, including high-precision multiplication and modular exponentiation.

Acknowledgments. Thanks to Theodorus J. Dekker for sending me copies of [1] and [2]; Thomas Johansson for sending me a copy of [24]; David A. Molnar for sending me a copy of [50]; and Donald E. Knuth for his comments.

2. Fixed-precision floating-point arithmetic

A **b-bit floating-point number** is a real number of the form $2^e f$ where e and f are integers with $|f| < 2^b$.

Theorem 2.1. *Let b be a nonnegative integer, and let k be an integer. Let x be a b-bit floating-point number. If $|x| > 2^{k-1}(2^b - 1)$ then $x \in 2^k\mathbf{Z}$.*

Proof. By hypothesis $x = 2^e f$ for some integers e, f with $|f| \le 2^b - 1$. If $e \ge k$ then $x \in 2^k\mathbf{Z}$. If $e \le k - 1$ then $|x| \le 2^e(2^b - 1) \le 2^{k-1}(2^b - 1)$. □

Addition and multiplication. Modern computers can quickly perform several operations upon 53-bit floating-point numbers (under minor restrictions that are not relevant to this paper):

- Given 53-bit floating-point numbers x, y, compute $\mathrm{fp}_{53}(x + y)$.
- Given 53-bit floating-point numbers x, y, compute $\mathrm{fp}_{53}(x - y)$.
- Given 53-bit floating-point numbers x, y, compute $\mathrm{fp}_{53}(xy)$.

Here $\mathrm{fp}_b\, z$ is, for each real number z, a b-bit floating-point number close to z. Specifically, if $|z| \le 2^{e+b}$ then $|z - \mathrm{fp}_b\, z|$ is guaranteed to be at most 2^{e-1}.

Theorem 2.2. *Let b be a positive integer. Let z be a b-bit floating-point number. Then $z = \mathrm{fp}_b\, z$.*

Therefore, given 53-bit floating-point numbers x and y, modern computers can quickly compute $x + y$ if $x + y$ is a 53-bit floating-point number, and $x - y$ if $x - y$ is a 53-bit floating-point number, and xy if xy is a 53-bit floating-point number.

Proof. Suppose $\mathrm{fp}_b\, z \ne z$. Find an integer e such that $2^{e-1} < |z - \mathrm{fp}_b\, z| \le 2^e$. Then $|z| > 2^{e+b}$ by the fp_b guarantee, so $|\mathrm{fp}_b\, z| > 2^e(2^b - 1)$. Both z and $\mathrm{fp}_b\, z$ are in $2^{e+1}\mathbf{Z}$ by Theorem 2.1, so $|z - \mathrm{fp}_b\, z|$ is in $2^{e+1}\mathbf{Z}$; contradiction. □

Theorem 2.3. *Let b be a positive integer, and let k be an integer. Let z be a real number in $[2^{k+b-1}, 2^{k+b}]$. Then $\mathrm{fp}_b\, z \in 2^k\mathbf{Z}$ and $\mathrm{fp}_b\, z \in [2^{k+b-1}, 2^{k+b}]$.*

Proof. Write $x = \mathrm{fp}_b\, z$. Then $|z - x| \le 2^{k-1}$ by the fp_b guarantee; so $L \le x \le R$ where $L = 2^{k-1}(2^b - 1)$ and $R = 2^{k-1}(2^{b+1} + 1)$.

Observe that $x > L$. (If not then $z \le L + 2^{k-1} = 2^{k+b-1}$, so $|z - x| \le 2^{k-2}$ by the fp_b guarantee, so $z \le L + 2^{k-2} < 2^{k+b-1}$, contradiction.) Thus $x \in 2^k\mathbf{Z}$ by Theorem 2.1. The multiples of 2^k between L and R are all between 2^{k+b-1} and 2^{k+b} inclusive. □

Rounding. Define $\mathrm{top}_{k,b}\, x = \mathrm{fp}_b(\mathrm{fp}_b(x + 3\cdot 2^{k+b-2}) - 3\cdot 2^{k+b-2})$ and $\mathrm{bottom}_{k,b}\, x = \mathrm{fp}_b(x - \mathrm{top}_{k,b}\, x)$. Note that $3 \cdot 2^{k+b-2}$ is a b-bit floating-point number for $b \ge 2$.

Theorem 2.4. *Let b be an integer with $b \ge 2$, and let k be an integer. Let x be a b-bit floating-point number with $|x| \le 2^{k+b-2}$. Then $x = \mathrm{top}_{k,b}\, x + \mathrm{bottom}_{k,b}\, x$; $\mathrm{top}_{k,b}\, x \in 2^k\mathbf{Z}$; and $|\mathrm{bottom}_{k,b}\, x| \le 2^{k-1}$.*

Proof. Write $z = x + 3 \cdot 2^{k+b-2}$ and $y = \mathrm{fp}_b\, z - 3 \cdot 2^{k+b-2}$, so that $\mathrm{top}_{k,b}\, x = \mathrm{fp}_b\, y$. I will show that $\mathrm{top}_{k,b}\, x = y$ and $\mathrm{bottom}_{k,b}\, x = x - y$.

First $2^{k+b-1} \le z \le 2^{k+b}$. Thus $|x - y| = |z - \mathrm{fp}_b\, z| \le 2^{k-1}$. Furthermore, by Theorem 2.3, $\mathrm{fp}_b\, z \in 2^k\mathbf{Z}$, and $2^{k+b-1} \le \mathrm{fp}_b\, z \le 2^{k+b}$. Thus $y \in 2^k\mathbf{Z}$, and $|y| \le 2^{b-2}2^k$; so y is a b-bit floating-point number. By Theorem 2.2, $y = \mathrm{fp}_b\, y = \mathrm{top}_{k,b}\, x$.

If $y = 0$ then $x - y = x$ so $x - y$ is a b-bit floating-point number. Otherwise $|y| \ge 2^k$ so $|x| \ge 2^{k-1}$. By Theorem 2.1, $x \in 2^{k-b}\mathbf{Z}$. Thus $x - y \in 2^{k-b}\mathbf{Z}$, and $|x - y| \le 2^{b-1}2^{k-b}$; so $x - y$ is a b-bit floating-point number.

Finally $\mathrm{bottom}_{k,b}\, x = \mathrm{fp}_b(x - \mathrm{top}_{k,b}\, x) = \mathrm{fp}_b(x - y) = x - y$ by Theorem 2.2. □

Rounding down. Now define $\mathrm{topleft}_{k,b}\, z = \mathrm{top}_{k,b}\, \mathrm{fp}_b(z - (2^{k-1} - 2^{k-33}))$ and $\mathrm{bottomleft}_{k,b}\, z = \mathrm{fp}_b(z - \mathrm{topleft}_{k,b}\, z)$. Note that $2^{k-1} - 2^{k-33}$ is a b-bit floating-point number for $b \ge 32$.

Theorem 2.5. *Let b be an integer with $b \ge 34$, and let k be an integer. Let z be a real number with $z \in 2^{k-32}\mathbf{Z}$ and $|z| \le 2^{k+b-34}$. Then $\mathrm{topleft}_{k,b}\, z = 2^k \lfloor z/2^k \rfloor$ and $\mathrm{bottomleft}_{k,b}\, z = z \bmod 2^k$.*

Proof. Write $x = z - (2^{k-1} - 2^{k-33})$. Then $x \in 2^{k-33}\mathbf{Z}$, and $|x| < 2^{k+b-34} + 2^{k-1} < 2^b 2^{k-33}$, so x is a b-bit floating-point number. Thus $\mathrm{fp}_b\, x = x$ by Theorem 2.2.

Write $y = \mathrm{topleft}_{k,b}\, z$. Then $y = \mathrm{top}_{k,b}\, \mathrm{fp}_b\, x = \mathrm{top}_{k,b}\, x$. By Theorem 2.4, $y \in 2^k\mathbf{Z}$ and $|x - y| \le 2^{k-1}$. Now $y \le x + 2^{k-1} = z + 2^{k-33}$, and both y and z are in $2^{k-32}\mathbf{Z}$, so $y \le z$; and $z < x + 2^{k-1} \le y + 2^k$. Hence $y = 2^k \lfloor z/2^k \rfloor$.

Finally, $z - y$ is a multiple of 2^{k-32} between 0 and $2^{b-2}2^{k-32}$, so it is a b-bit floating-point number. By Theorem 2.2, $z - y = \mathrm{fp}_b(z - y) = \mathrm{bottomleft}_{k,b}\, z$. □

Notes. IEEE in [4] encouraged computer manufacturers to provide hardware for certain operations on 24-bit ("single-precision") and 53-bit ("double-precision") floating-point numbers. It is now generally safe to assume that, for example, a C compiler will interpret the addition of two `double` variables as the operation $x, y \mapsto \mathrm{fp}_{53}(x + y)$. Exception: On x86 chips, such as the Pentium, addition of `double` variables is *sometimes* interpreted as $x, y \mapsto \mathrm{fp}_{64}(x + y)$. Here's the story:

- The Pentium handles 64-bit ("extended-precision") addition at the same speed as 53-bit addition. Unfortunately, it uses the same instruction for 24-bit addition, 53-bit addition, and 64-bit addition; there is a global variable ("precision control") that selects the precision of subsequent additions.

- C compilers could set 24-bit precision before `float` additions, 53-bit precision before `double` additions, and 64-bit precision before `long double` additions; but that would take time and effort. Many compilers, notably `gcc`, leave the precision alone, under the incorrect theory that anyone asking for 53-bit precision will be just as happy with 64-bit precision.
- Some operating systems, notably Linux, set 64-bit precision by default.

Programs that need fp_{53}, and not fp_{64}, must set 53-bit precision manually.

For each real number z, the number of choices of $\mathrm{fp}_b\, z$ meeting the fp_b guarantee is either 1 or 2. IEEE specified a particular choice of $\mathrm{fp}_b\, z$, following the "round-to-even" rule discussed in [44, page 221].

The idea of Theorem 2.4 is known in numerical analysis as "loss of precision from cancellation"; see, e.g., [44, exercise 4.2.2–25]. The fact that precision loss can be useful—that, in particular, one can extract the high bits from a floating-point number by adding and then subtracting a larger number—was pointed out by Kahan in [41]. The use of a constant for the larger number, for example to round a floating-point number to an integer, appears in, e.g., [44, page 209].

Bit extraction is easy in hardware. Computers offer bit-extraction operations that are faster than the addition and subtraction in $\mathrm{top}_{k,b}$. Unfortunately, these bit-extraction operations work with numbers stored in one format, and floating-point operations work with numbers stored in another format; it takes time to convert numbers from one format to the other.

3. Carries modulo $2^{127}-1$

One can write an integer modulo $2^{127}-1$ in many ways as a sum $t_7+t_6+\cdots+t_0$ where t_j is a 53-bit floating-point number with $t_j \in 2^{16j}\mathbf{Z}$. Theorem 3.2 starts from a sum $t_7+t_6+\cdots+t_0$ with $|t_j| \le 0.98\cdot 2^{53}2^{16j}$ and constructs an equivalent sum $v_7+v_6+\cdots+v_0$ with $|v_j| \le 1.01\cdot 2^{15}2^{16j}$.

Define $\mathrm{carry}_i(t_0,\ldots,t_7)$ and $\mathrm{carrybound}_i(T_0,\ldots,T_7)$ for $i \in \{0,1,2,3,4,5,6\}$ as follows: $\mathrm{carry}_i(t_0,\ldots,t_7) = (u_0,\ldots,u_7)$ where $u_i = \mathrm{bottom}_{16+16i,53}\, t_i$, $u_{i+1} = \mathrm{fp}_{53}(t_{i+1} + \mathrm{top}_{16+16i,53}\, t_i)$, and $u_j = t_j$ otherwise; and $\mathrm{carrybound}_i(T_0,\ldots,T_7) = (U_0,\ldots,U_7)$ where $U_i = 2^{15+16i}$, $U_{i+1} = T_{i+1}+T_i+2^{15+16i}$, and $U_j = T_j$ otherwise.

Also define $\mathrm{carry}_i(t_0,\ldots,t_7)$ and $\mathrm{carrybound}_i(T_0,\ldots,T_7)$ for $i=7$ as follows: $\mathrm{carry}_7(t_0,\ldots,t_7) = (u_0,\ldots,u_7)$ where $u_0 = \mathrm{fp}_{53}(t_0 + 2^{-127}\,\mathrm{top}_{127,53}\, t_7)$, $u_7 = \mathrm{bottom}_{127,53}\, t_7$, and $u_j = t_j$ otherwise; $\mathrm{carrybound}_7(T_0,\ldots,T_7) = (U_0,\ldots,U_7)$ where $U_0 = T_0 + 2^{-127}T_7 + 2^{-1}$, $U_7 = 2^{126}$, and $U_j = T_j$ otherwise.

Finally define $\mathrm{squeeze} = \mathrm{carry}_1\,\mathrm{carry}_0\,\mathrm{carry}_7\,\mathrm{carry}_6\ldots\mathrm{carry}_1\,\mathrm{carry}_0$.

Theorem 3.1. *Let $T_0,\ldots,T_7$ be real numbers with $T_j \le 0.99\cdot 2^{53}2^{16j}$. Let $t_0,\ldots,t_7$ be real numbers with $t_j \in 2^{16j}\mathbf{Z}$ and $|t_j| \le T_j$. Fix $i \in \{0,\ldots,7\}$. Define $(u_0,\ldots,u_7) = \mathrm{carry}_i(t_0,\ldots,t_7)$ and $(U_0,\ldots,U_7) = \mathrm{carrybound}_i(T_0,\ldots,T_7)$. Then $u_7+\cdots+u_0 \equiv t_7+\cdots+t_0 \pmod{2^{127}-1}$; $u_j \in 2^{16j}\mathbf{Z}$; and $|u_j| \le U_j$.*

Proof. For $i=7$: Write $y = \mathrm{top}_{127,53}\, t_7$. By Theorem 2.4, $\mathrm{bottom}_{127,53}\, t_7 = t_7 - y$; $y \in 2^{127}\mathbf{Z}$; and $|t_7 - y| \le 2^{126}$. Thus $u_7 = t_7 - y \in 2^{112}\mathbf{Z}$ and $|u_7| \le 2^{126} = U_7$. Furthermore, $t_0 + 2^{-127}y \in \mathbf{Z}$, and

$$\left|t_0 + 2^{-127}y\right| \le T_0 + 2^{-127}T_7 + 2^{-1} \le 0.99(1+2^{-15})2^{53} + 2^{-1} < 2^{53},$$

so $u_0 = \mathrm{fp}_{53}(t_0 + 2^{-127}y) = t_0 + 2^{-127}y$ by Theorem 2.2. Hence $u_0 \in \mathbf{Z}$ and $|u_0| \le T_0 + 2^{-127}T_7 + 2^{-1} = U_0$. Also $u_7+\cdots+u_0 - t_7 - \cdots - t_0 = (1-2^{127})2^{-127}y \in (2^{127}-1)\mathbf{Z}$.

For $i < 7$: Write $y = \mathrm{top}_{16+16i,53}\, t_i$. By Theorem 2.4, $\mathrm{bottom}_{16+16i,53}\, t_i = t_i - y$; $y \in 2^{16+16i}\mathbf{Z}$; and $|t_i - y| \le 2^{15+16i}$. Thus $u_i = t_i - y \in 2^{16i}\mathbf{Z}$ and $|u_i| \le 2^{15+16i} = U_i$. Furthermore, $t_{i+1} + y \in 2^{16+16i}\mathbf{Z}$, and

$$|t_{i+1} + y| \le T_{i+1} + T_i + 2^{15+16i} \le (0.99(1+2^{-16})2^{53} + 2^{-1})2^{16+16i} < 2^{53}2^{16+16i},$$

so $u_{i+1} = \mathrm{fp}_{53}(t_{i+1} + y) = t_{i+1} + y$ by Theorem 2.2. Hence $u_{i+1} \in 2^{16+16i}\mathbf{Z}$ and $|u_{i+1}| \le T_{i+1} + T_i + 2^{15+16i} = U_{i+1}$. Also $u_7 + \cdots + u_0 = t_7 + \cdots + t_0$. □

Theorem 3.2. *Let $t_0, \ldots, t_7$ be real numbers with $t_j \in 2^{16j}\mathbf{Z}$ and $|t_j| \le 0.98 \cdot 2^{53}2^{16j}$. Define $(v_0, \ldots, v_7) = \mathrm{squeeze}(t_0, \ldots, t_7)$. Then $v_j \in 2^{16j}\mathbf{Z}$; $|v_j| \le 1.01 \cdot 2^{15}2^{16j}$; and $v_7 + \cdots + v_0 \equiv t_7 + \cdots + t_0 \pmod{2^{127} - 1}$.*

Proof. Write $T_j = 0.98 \cdot 2^{53}2^{16j}$. The reader may check that

$$\mathrm{carrybound}_1\, \mathrm{carrybound}_0\, \mathrm{carrybound}_7 \ldots \mathrm{carrybound}_0(T_0, T_1, \ldots, T_7)$$

is $(2^{15}, 2^{31}, (1.001\ldots)2^{47}, 2^{63}, 2^{79}, 2^{95}, 2^{111}, 2^{126})$, and that each of the intermediate carrybound results is at most $(0.99 \cdot 2^{53}, 0.99 \cdot 2^{53}2^{16}, \ldots, 0.99 \cdot 2^{53}2^{112})$. Apply Theorem 3.1 repeatedly. □

Notes. There are many variants of Theorem 3.2. One can replace squeeze with $\mathrm{carry}_6\, \mathrm{carry}_5\, \mathrm{carry}_4\, \mathrm{carry}_3\, \mathrm{carry}_2\, \mathrm{carry}_1\, \mathrm{carry}_0\, \mathrm{carry}_7\, \mathrm{carry}_6\, \mathrm{carry}_5$ to obtain a more pleasant bound $1 \cdot 2^{15}2^{16j}$. Or one can replace squeeze with

$$\mathrm{carry}_5\, \mathrm{carry}_1\, \mathrm{carry}_4\, \mathrm{carry}_0\, \mathrm{carry}_7\, \mathrm{carry}_3\, \mathrm{carry}_6\, \mathrm{carry}_2\, \mathrm{carry}_5\, \mathrm{carry}_1\, \mathrm{carry}_4\, \mathrm{carry}_0;$$

this involves more floating-point operations but allows much better parallelization on typical computers.

Section 4 explains how to find a *unique* representative for each integer modulo $2^{127} - 1$. In the computation of h_r, however, I do not find unique representatives until the end; instead I use Theorem 3.2 to find *small* representatives, sufficiently small to carry out further operations. This idea was published by Robertson in [52] as a way to speed up division circuitry, and expanded by Avizienis in [9] to other arithmetic operations.

4. Complete reduction modulo $2^{127} - 1$

Define $\mathrm{freeze}(t_0, t_1, t_2, t_3) = (x_0, x_1, x_2, x_3)$ where

$$\begin{aligned}
q_0 &= \mathrm{topleft}_{0,53}\, \mathrm{fp}_{53}(2^{-127}t_3 + 2^{-1}),\\
q_1 &= \mathrm{topleft}_{32,53}\, \mathrm{fp}_{53}(t_0 + q_0),\\
q_2 &= \mathrm{topleft}_{64,53}\, \mathrm{fp}_{53}(t_1 + q_1),\\
q_3 &= \mathrm{topleft}_{96,53}\, \mathrm{fp}_{53}(t_2 + q_2),\\
q_4 &= \mathrm{topleft}_{127,53}\, \mathrm{fp}_{53}(t_3 + q_3),\\
u_0 &= \mathrm{fp}_{53}(t_0 + 2^{-127}q_4),\\
u_1 &= \mathrm{fp}_{53}(t_1 + \mathrm{topleft}_{32,53}\, u_0),\\
u_2 &= \mathrm{fp}_{53}(t_2 + \mathrm{topleft}_{64,53}\, u_1),\\
x_0 &= \mathrm{bottomleft}_{32,53}\, u_0,\\
x_1 &= \mathrm{bottomleft}_{64,53}\, u_1,\\
x_2 &= \mathrm{bottomleft}_{96,53}\, u_2, \text{ and}\\
x_3 &= \mathrm{fp}_{53}(\mathrm{fp}_{53}(t_3 - q_4) + \mathrm{topleft}_{96,53}\, u_2).
\end{aligned}$$

Theorem 4.2 shows that if $t = t_3 + t_2 + t_1 + t_0$, with $t_j \in 2^{32j}\mathbf{Z}$ and t_j not too large, then the base-2^{32} representation of $t \bmod (2^{127} - 1)$ is $2^{-96}x_3, 2^{-64}x_2, 2^{-64}x_1, x_0$.

Theorem 4.1. *Let u and v be integers with $|u| \le 2^{120}$ and $|v| \le 2^{120}$. Define $t = 2^{96}v + u$. Then $\lfloor t/(2^{127}-1) \rfloor = \lfloor 2^{-127}(t + 2^{-31}v + 2^{-1}) \rfloor$.*

Proof. Write $p = 2^{127} - 1$ and $q = \lfloor t/p \rfloor$. Then $|q| \le 2^{90}$ since $|t| \le (2^{96}+1)2^{120} \le 2^{90}p$; so $|2^{-127}q + 2^{-127}u| \le 2^{-37} + 2^{-7}$; so $0 < 2^{-1} - 2^{-127}q - 2^{-127}u < 1$.

Now write $r = t - pq$ and $x = r + 2^{-127}r + 2^{-1} - 2^{-127}q - 2^{-127}u$. Observe that $2^{-127}(t+2^{-31}v+2^{-1}) = q+2^{-127}x$. But $0 \le r \le p-1 < 2^{127}$, so $0 \le r+2^{-127}r < p$, so $0 < x < p+1 = 2^{127}$, so $\lfloor q + 2^{-127}x \rfloor = q$. □

Theorem 4.2. *Let t_0, t_1, t_2, t_3 be real numbers with $t_j \in 2^{32j}\mathbf{Z}$ and $|t_j| \le 2^{49}2^{32j}$. Define $(x_0, x_1, x_2, x_3) = \operatorname{freeze}(t_0, t_1, t_2, t_3)$. Then $x_j \in 2^{32j}\mathbf{Z}$; $0 \le x_j < 2^{32}2^{32j}$; and $(t_3 + t_2 + t_1 + t_0) \bmod (2^{127}-1) = x_3 + x_2 + x_1 + x_0$.*

Proof. Write $p = 2^{127} - 1$, $t = t_3 + t_2 + t_1 + t_0$, and $\epsilon = 2^{-127}t_3 + 2^{-1}$. Then $\lfloor t/p \rfloor = \lfloor 2^{-127}(t + \epsilon) \rfloor$ by Theorem 4.1.

Write $q_0 = \operatorname{topleft}_{0,53} \operatorname{fp}_{53} \epsilon$. Note that $\epsilon \in 2^{-32}\mathbf{Z}$ and $|\epsilon| \le 2^{19}$; so $q_0 = \operatorname{topleft}_{0,53} \epsilon$ by Theorem 2.2. Thus $q_0 = \lfloor \epsilon \rfloor$ by Theorem 2.5.

Similarly write $q_1 = \operatorname{topleft}_{32,53} \operatorname{fp}_{53}(t_0 + q_0)$, $q_2 = \operatorname{topleft}_{64,53} \operatorname{fp}_{53}(t_1 + q_1)$, $q_3 = \operatorname{topleft}_{96,53} \operatorname{fp}_{53}(t_2 + q_2)$, and $q_4 = \operatorname{topleft}_{127,53} \operatorname{fp}_{53}(t_3 + q_3)$. Then $q_1 = 2^{32} \lfloor 2^{-32}(t_0 + \epsilon) \rfloor$; $q_2 = 2^{64} \lfloor 2^{-64}(t_1 + t_0 + \epsilon) \rfloor$; $q_3 = 2^{96} \lfloor 2^{-96}(t_2 + t_1 + t_0 + \epsilon) \rfloor$; and $q_4 = 2^{127} \lfloor 2^{-127}(t_3 + t_2 + t_1 + t_0 + \epsilon) \rfloor = 2^{127} \lfloor t/p \rfloor$.

Hence $t \bmod p = t - q_4 + 2^{-127}q_4$. Also $|2^{-127}q_4| \le 2^{19}$ since $|t| \le 2^{19}p$.

Write $w = \operatorname{fp}_{53}(t_3 - q_4)$ and $u_0 = \operatorname{fp}_{53}(t_0 + 2^{-127}q_4)$. Note that $t_3 - q_4 \in 2^{96}\mathbf{Z}$ and $|t_3 - q_4| \le 2^{51}2^{96}$. Thus $w = t_3 - q_4$ by Theorem 2.2. Similarly, note that $t_0 + 2^{-127}q_4 \in \mathbf{Z}$ and $|t_0 + 2^{-127}q_4| \le 2^{50}$. Thus $u_0 = t_0 + 2^{-127}q_4$ by Theorem 2.2. Hence $t \bmod p = w + t_2 + t_1 + u_0$.

Now write $y_0 = \operatorname{topleft}_{32,53} u_0$, $u_1 = \operatorname{fp}_{53}(t_1 + y_0)$, $y_1 = \operatorname{topleft}_{64,53} u_1$, $u_2 = \operatorname{fp}_{53}(t_2 + y_1)$, and $y_2 = \operatorname{topleft}_{96,53} u_2$. Then $x_0 = \operatorname{bottomleft}_{32,53} u_0 = u_0 \bmod 2^{32}$ and $y_0 = 2^{32} \lfloor 2^{-32}u_0 \rfloor$ by Theorem 2.5. Apply Theorem 2.2 to see that $u_1 = t_1 + y_0 \in 2^{32}\mathbf{Z}$. Hence $t \bmod p = w + t_2 + u_1 + x_0$.

Similarly $x_1 = \operatorname{bottomleft}_{64,53} u_1 = u_1 \bmod 2^{64}$; $y_1 = 2^{64} \lfloor 2^{-64}u_1 \rfloor$; $u_2 = t_2 + y_1 \in 2^{64}\mathbf{Z}$; $t \bmod p = w + u_2 + x_1 + x_0$; $x_2 = \operatorname{bottomleft}_{96,53} u_2 = u_2 \bmod 2^{96}$; $y_2 = 2^{96} \lfloor 2^{-96}u_2 \rfloor$; $x_3 = \operatorname{fp}_{53}(w+y_2) = w+y_2 \in 2^{96}\mathbf{Z}$; and $t \bmod p = x_3+x_2+x_1+x_0$. □

Notes. Theorem 4.1 is a special case of the well-known fact that approximate $2b$-bit division suffices for correctly rounded b-bit division. This is often used in the design of hardware for the operation $x, y \mapsto \operatorname{fp}_b(x/y)$ required by [4].

5. Multiplication modulo $2^{127} - 1$

This section explains how to find a representative for the product of two integers modulo $2^{127} - 1$.

Define $\operatorname{prod}(r_0, r_1, r_2, r_3, u_0, u_1, \ldots, u_7) = (t_0, t_1, \ldots, t_7)$ where

$$\begin{pmatrix} t_0 & t_1 \\ t_2 & t_3 \\ t_4 & t_5 \\ t_6 & t_7 \end{pmatrix} = \begin{pmatrix} r_0 & 2^{-127}r_3 & 2^{-127}r_2 & 2^{-127}r_1 \\ r_1 & r_0 & 2^{-127}r_3 & 2^{-127}r_2 \\ r_2 & r_1 & r_0 & 2^{-127}r_3 \\ r_3 & r_2 & r_1 & r_0 \end{pmatrix} \begin{pmatrix} u_0 & u_1 \\ u_2 & u_3 \\ u_4 & u_5 \\ u_6 & u_7 \end{pmatrix}.$$

Define $\operatorname{power}_1(r_0, r_1, r_2, r_3) = (c_{1,0}, \ldots, c_{1,7})$ where $c_{1,2j} = \operatorname{bottom}_{16+32j,53} r_j$ and $c_{1,2j+1} = \operatorname{top}_{16+32j,53} r_j$. For $i \ge 1$ define recursively $\operatorname{power}_{i+1}(r_0, r_1, r_2, r_3) = \operatorname{squeeze}(\operatorname{prod}(r_0, r_1, r_2, r_3, \operatorname{power}_i(r_0, r_1, r_2, r_3)))$.

Theorem 5.1. *Let r_0, r_1, r_2, r_3 and $u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7$ be real numbers with $r_i \in 2^{32i}\mathbf{Z}$, $|r_i| \le 1.1 \cdot 2^{32}2^{32i}$, $u_j \in 2^{16j}\mathbf{Z}$, and $|u_j| \le 1.01 \cdot 2^{15}2^{16j}$. Define $(t_0, t_1, \ldots, t_7) = \operatorname{prod}(r_0, r_1, r_2, r_3, u_0, u_1, \ldots, u_7)$. Then $t_j \in 2^{16j}\mathbf{Z}$; $|t_j| \le 2^{50}2^{16j}$; and $t_7 + \cdots + t_0 \equiv (r_3 + r_2 + r_1 + r_0)(u_7 + \cdots + u_0) \pmod{2^{127} - 1}$.*

Proof. Observe that

$$\begin{pmatrix} t_0 & 2^{-16}t_1 \\ 2^{-32}t_2 & 2^{-48}t_3 \\ 2^{-64}t_4 & 2^{-80}t_5 \\ 2^{-96}t_6 & 2^{-112}t_7 \end{pmatrix} = \begin{pmatrix} r_0 & 2^{-95}r_3 & 2^{-63}r_2 & 2^{-31}r_1 \\ 2^{-32}r_1 & r_0 & 2^{-95}r_3 & 2^{-63}r_2 \\ 2^{-64}r_2 & 2^{-32}r_1 & r_0 & 2^{-95}r_3 \\ 2^{-96}r_3 & 2^{-64}r_2 & 2^{-32}r_1 & r_0 \end{pmatrix} \begin{pmatrix} u_0 & 2^{-16}u_1 \\ 2^{-32}u_2 & 2^{-48}u_3 \\ 2^{-64}u_4 & 2^{-80}u_5 \\ 2^{-96}u_6 & 2^{-112}u_7 \end{pmatrix}.$$

A column in the u matrix consists of four integers bounded in absolute value by $1.01 \cdot 2^{15}$. A row in the r matrix consists of four integers bounded in absolute value by $1.1 \cdot 2^{32}, 2.2 \cdot 2^{32}, 2.2 \cdot 2^{32}, 2.2 \cdot 2^{32}$. Thus each entry in the t matrix is an integer bounded in absolute value by $1.01(1.1 + 2.2 + 2.2 + 2.2)2^{32}2^{15} < 2^{50}$.

The difference between $(\sum r_i)(\sum u_j)$ and $\sum t_j$ is $2^{128} - 2$ times the integer $2^{-96}r_3 2^{-32}(u_2+u_3) + 2^{-64}(r_2+r_3)2^{-64}(u_4+u_5) + 2^{-32}(r_1+r_2+r_3)2^{-96}(u_6+u_7)$. □

Theorem 5.2. *Let r_0, r_1, r_2, r_3 and $u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7$ be real numbers with $r_i \in 2^{32i}\mathbf{Z}$, $|r_i| \le 1.1 \cdot 2^{32}2^{32i}$, $u_j \in 2^{16j}\mathbf{Z}$, and $|u_j| \le 1.01 \cdot 2^{15}2^{16j}$. Define $(t_0, t_1, \ldots, t_7) = \operatorname{prod}(r_0, r_1, r_2, r_3, u_0, u_1, \ldots, u_7)$. Then*

$$t_0 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_0u_0) + \operatorname{fp}(2^{-127}r_3u_2)) + \operatorname{fp}(2^{-127}r_2u_4)) + \operatorname{fp}(2^{-127}r_1u_6)),$$
$$t_1 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_0u_1) + \operatorname{fp}(2^{-127}r_3u_3)) + \operatorname{fp}(2^{-127}r_2u_5)) + \operatorname{fp}(2^{-127}r_1u_7)),$$
$$t_2 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_1u_0) + \operatorname{fp}(r_0u_2)) + \operatorname{fp}(2^{-127}r_3u_4)) + \operatorname{fp}(2^{-127}r_2u_6)),$$
$$t_3 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_1u_1) + \operatorname{fp}(r_0u_3)) + \operatorname{fp}(2^{-127}r_3u_5)) + \operatorname{fp}(2^{-127}r_2u_7)),$$
$$t_4 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_2u_0) + \operatorname{fp}(r_1u_2)) + \operatorname{fp}(r_0u_4)) + \operatorname{fp}(2^{-127}r_3u_6)),$$
$$t_5 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_2u_1) + \operatorname{fp}(r_1u_3)) + \operatorname{fp}(r_0u_5)) + \operatorname{fp}(2^{-127}r_3u_7)),$$
$$t_6 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_3u_0) + \operatorname{fp}(r_2u_2)) + \operatorname{fp}(r_1u_4)) + \operatorname{fp}(r_0u_6)), \text{ and}$$
$$t_7 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_3u_1) + \operatorname{fp}(r_2u_3)) + \operatorname{fp}(r_1u_5)) + \operatorname{fp}(r_0u_7))$$

where $\operatorname{fp} = \operatorname{fp}_{53}$.

Thus, in the situation of Theorem 5.1, prod can be computed with a series of floating-point operations. Note that the order of summation here is not relevant: for example, $t_7 = \operatorname{fp}(\operatorname{fp}(\operatorname{fp}(\operatorname{fp}(r_0u_7) + \operatorname{fp}(r_1u_5)) + \operatorname{fp}(r_2u_3)) + \operatorname{fp}(r_3u_1))$.

Proof. Apply Theorem 2.2 repeatedly. All the products such as r_0u_0 and partial sums such as $r_0u_0 + 2^{-127}r_3u_2$ are 53-bit floating-point numbers; the reader may check this directly, or apply Theorem 5.1 after strategically replacing various u's with 0. □

Theorem 5.3. *Let r_0, r_1, r_2, r_3 be real numbers with $r_j \in 2^{32j}\mathbf{Z}$ and $|r_j| \le 2^{31}2^{32j}$. Define $(c_{i,0}, c_{i,1}, \ldots, c_{i,7}) = \operatorname{power}_i(r_0, r_1, r_2, r_3)$. Then $c_{i,j} \in 2^{16j}\mathbf{Z}$; $|c_{i,j}| \le 1.01 \cdot 2^{15}2^{16j}$; and $(r_3 + r_2 + r_1 + r_0)^i \equiv c_{i,7} + \cdots + c_{i,0} \pmod{2^{127} - 1}$.*

Proof. By Theorem 2.4, $r_j = c_{1,2j} + c_{1,2j+1}$; $c_{1,2j+1} \in 2^{16+32j}\mathbf{Z}$; and $|c_{1,2j}| \le 2^{15+32j}\mathbf{Z}$. Thus $c_{1,2j} = r_j - c_{1,2j+1} \in 2^{32j}\mathbf{Z}$; and $|c_{1,2j+1}| \le 2^{15}2^{16+32j}$ since $|r_j| \le 2^{31}2^{32j}$. Also $r_3 + r_2 + r_1 + r_0 = c_{1,7} + \cdots + c_{1,0}$.

For $i \ge 1$, assume inductively that $c_{i,j} \in 2^{16j}\mathbf{Z}$; $|c_{i,j}| \le 1.01 \cdot 2^{15}2^{16j}$; and $(r_3 + r_2 + r_1 + r_0)^i \equiv c_{i,7} + \cdots + c_{i,0} \pmod{2^{127} - 1}$.

Write $(d_0, \ldots, d_7) = \operatorname{prod}(r_0, r_1, r_2, r_3, c_{i,0}, \ldots, c_{i,7})$. By Theorem 5.1, $d_j \in 2^{16j}\mathbf{Z}$; $|d_j| \le 2^{50}2^{16j}$; and $d_7 + \cdots + d_0 \equiv (r_3 + r_2 + r_1 + r_0)(c_{i,7} + \cdots + c_{i,0})$.

Now $(c_{i+1,0},\ldots,c_{i+1,7}) = \mathrm{squeeze}(d_0,\ldots,d_7)$. By Theorem 3.2, $c_{i+1,j} \in 2^{16j}\mathbf{Z}$; $c_{i+1,7}+\cdots+c_{i+1,0} \equiv d_7+\cdots+d_0 \equiv (r_3+r_2+r_1+r_0)^{i+1}$; and $|c_{i+1,j}| \le 1.01 \cdot 2^{15}2^{16j}$. □

Notes. In the language of [19], Theorem 5.1 maps the ring $\mathbf{Z}/(2^{127}-1)$ to the isomorphic ring $\mathbf{Z}[x]/(x-2^{16},2^{127}-1)$, then lifts to $\mathbf{Z}[x]/(x^8-2)$, then carries out arithmetic in $\mathbf{Z}[x]/(x^8-2)$. Observe that $\sum_j 2^{-16j}t_jx^j$ in Theorem 5.1 is the product of $\sum_j 2^{-16j}u_jx^j$ and $\sum_i 2^{-32i}r_ix^{2i}$ in $\mathbf{Z}[x]/(x^8-2)$. Meanwhile, Theorem 3.2 reduces elements of $\mathbf{Z}[x]/(x^8-2)$ modulo $x-2^{16}$ and $2^{15}x^7-1$.

The fact that one can build fast high-precision arithmetic from floating-point arithmetic, using floating-point additions and subtractions to split each floating-point number into two small pieces so that subsequent floating-point multiplications are exact, was pointed out by Gerhard W. Veltkamp in 1968—see [1], [2], and [44, exercise 4.2.2–21]—and independently by Dekker in [32]. The main improvement in this paper is that each floating-point number is in a range specified in advance; this eliminates all floating-point comparisons, many additions and subtractions, and some multiplications.

6. Computation of h_r

Let $m_0,m_1,\ldots,m_{\ell-1}$ be integers with $|m_j| \le 2^{31}$. Let $r_0,r_1,r_2,r_3,k_0,k_1,k_2,k_3$ be real numbers with $r_j \in 2^{32j}\mathbf{Z}$, $|r_j| \le 2^{31}2^{32j}$, $k_j \in 2^{32j}\mathbf{Z}$, and $|k_j| \le 2^{48}2^{32j}$.

Write $r = r_3+r_2+r_1+r_0$, $k = k_3+k_2+k_1+k_0$, and $p = 2^{127}-1$. This section explains how to compute $(r^{\ell+1}+m_0r^\ell+m_1r^{\ell-1}+\cdots+m_{\ell-1}r+k) \bmod p$ using floating-point operations.

Precompute $(c_{i,0},\ldots,c_{i,7}) = \mathrm{power}_i(r_0,r_1,r_2,r_3)$ for $1 \le i \le 97$. Then $c_{i,j} \in 2^{16j}\mathbf{Z}$, $|c_{i,j}| \le 1.01 \cdot 2^{15}2^{16j}$, and $r^i \equiv c_{i,7}+\cdots+c_{i,0} \pmod p$ by Theorem 5.3.

Short inputs. For $\ell \le 96$: Define $u_j = c_{\ell+1,j}+m_0c_{\ell,j}+\cdots+m_{\ell-1}c_{1,j}$. Then $u_j \in 2^{16j}\mathbf{Z}$; $|u_j| \le 0.98 \cdot 2^{53}2^{16j}$; and $r^{\ell+1}+m_0r^\ell+\cdots+m_{\ell-1}r \equiv u_7+\cdots+u_0$. Note that $u_j = \mathrm{fp}_{53}(\ldots\mathrm{fp}_{53}(c_{\ell+1,j}+\mathrm{fp}_{53}(m_0c_{\ell,j}))+\cdots+\mathrm{fp}_{53}(m_{\ell-1}c_{1,j}))$.

Define $(v_0,\ldots,v_7) = \mathrm{squeeze}(u_0,\ldots,u_7)$. By Theorem 3.2, $|v_j| \le 1.01 \cdot 2^{15}2^{16j}$; $v_j \in 2^{16j}\mathbf{Z}$; and $r^{\ell+1}+m_0r^\ell+\cdots+m_{\ell-1}r \equiv v_7+\cdots+v_0$.

Next define $w_j = v_{2j}+v_{2j+1}$. Then $w_j \in 2^{32j}\mathbf{Z}$; $|w_j| \le 1.02 \cdot 2^{31}2^{32j}$; and $r^{\ell+1}+m_0r^\ell+\cdots+m_{\ell-1}r \equiv w_3+w_2+w_1+w_0$. Note that $w_j = \mathrm{fp}_{53}(v_{2j}+v_{2j+1})$.

Next define $t_j = k_j+w_j$. Then $r^{\ell+1}+m_0r^\ell+\cdots+m_{\ell-1}r+k \equiv t_3+t_2+t_1+t_0$; $t_j \in 2^{32j}\mathbf{Z}$; and $|t_j| \le 2^{49}2^{32j}$. Note that $t_j = \mathrm{fp}_{53}(k_j+w_j)$.

Finally define $(x_0,x_1,x_2,x_3) = \mathrm{freeze}(t_0,t_1,t_2,t_3)$. The result is the base-2^{32} representation of $(r^{\ell+1}+m_0r^\ell+\cdots+m_{\ell-1}r+k) \bmod p$ by Theorem 4.2.

Long inputs. For $\ell \ge 97$: Apply the same method recursively to $m_0,\ldots,m_{\ell-98}$ to construct real numbers w'_0,w'_1,w'_2,w'_3 such that $w'_j \in 2^{32j}\mathbf{Z}$; $|w'_j| \le 1.02 \cdot 2^{31}2^{32j}$; and $r^{\ell-96}+m_0r^{\ell-97}+\cdots+m_{\ell-98}r \equiv w'_3+w'_2+w'_1+w'_0$.

Define $a_0 = w'_0+m_{\ell-97}$, $a_1 = w'_1$, $a_2 = w'_2$, and $a_3 = w'_3$. Then $a_j \in 2^{32j}\mathbf{Z}$; $|a_j| \le 1.01 \cdot 2^{32}2^{32j}$; and $r^{\ell-96}+m_0r^{\ell-97}+\cdots+m_{\ell-98}r+m_{\ell-97} \equiv a_3+a_2+a_1+a_0$. Note that $a_0 = \mathrm{fp}_{53}(w'_0+m_{\ell-97})$.

Define $(b_0,\ldots,b_7) = \mathrm{prod}(a_0,a_1,a_2,a_3,c_{97,0},\ldots,c_{97,7})$. By Theorem 5.1, $b_j \in 2^{16j}\mathbf{Z}$; $|b_j| \le 2^{50}2^{16j}$; and $r^{\ell+1}+m_0r^\ell+\cdots+m_{\ell-98}r^{98}+m_{\ell-97}r^{97} \equiv b_7+\cdots+b_0$.

Define $u_j = b_j + m_{\ell-96}c_{96,j} + \cdots + m_{\ell-1}c_{1,j}$. Then $u_j \in 2^{16j}\mathbf{Z}$; $|u_j| \le 2^{50}2^{16j} + 96 \cdot 1.01 \cdot 2^{31}2^{15}2^{16j} \le 0.98 \cdot 2^{53}2^{16j}$; and $r^{\ell+1} + m_0 r^\ell + \cdots + m_{\ell-1}r \equiv u_7 + \cdots + u_0$. Note that $u_j = \mathrm{fp}_{53}(\ldots \mathrm{fp}_{53}(b_j + \mathrm{fp}_{53}(m_{\ell-96}c_{96,j})) + \cdots + \mathrm{fp}_{53}(m_{\ell-1}c_{1,j}))$.

Now construct $(v_0, \ldots, v_7) = \mathrm{squeeze}(u_0, \ldots, u_7)$, $w_j = v_{2j} + v_{2j+1}$, $t_j = k_j + w_j$, and $(x_0, x_1, x_2, x_3) = \mathrm{freeze}(t_0, t_1, t_2, t_3)$ as above.

Efficiency. The time spent on this computation has four major pieces:

- Time to multiply input vectors by the precomputed $c_{i,j}$ matrix. This is the heart of the computation.
- Time to multiply by r^{97} after every 97 input components. One can save time here, at the expense of a larger matrix, by instead multiplying by (e.g.) r^{970} after every 970 input components.
- Time to squeeze the intermediate results after every 97 input components. One can save some time here by more carefully analyzing the actual size of $\sum_i m_{\ell-i}c_{i,j}$. For example, $\sum_{1\le i\le 150} |c_{i,j}|$ is below $100 \cdot 2^{15}2^{16j}$ for *most* choices of r_0, r_1, r_2, r_3.
- Time to squeeze and freeze the final result. This is negligible for long inputs.

In practice, for long inputs, the matrix-vector products account for most of the total time.

Beware that there are several easy ways to drastically slow down floating-point operations. The most obvious is failure to keep necessary data in registers. Less obvious is failure to parallelize independent operations. Implementors must not assume that the "optimizers" in typical compilers will avoid these hazards. For example, straightforward C code to compute h_r (with the improvement described in the next section), compiled for a Pentium by a widespread C compiler, is *five times slower* than the speed reported in section 1.

7. The 64-bit variant

This section outlines a 64-bit variant of the algorithm described in section 6. In this variant, one represents an integer modulo $2^{127} - 1$ as a sum $t_4 + t_3 + t_2 + t_1 + t_0$ where t_j is a 64-bit floating-point number with $t_j \in 2^{26j}\mathbf{Z}$. This variant is useful on computers such as the Pentium and Pentium II, where five 64-bit floating-point operations are faster than eight 53-bit floating-point operations.

The analogue of Theorem 3.2 starts from a sum $t_4 + t_3 + t_2 + t_1 + t_0$ with $|t_j| \le 0.98 \cdot 2^{64}2^{26j}$. It uses six carries to produce a sum $v_4 + v_3 + v_2 + v_1 + v_0$ with $|v_j| \le 1.01 \cdot 2^{25}2^{26j}$.

The analogue of Theorem 5.1 starts from one sum $r_4 + r_3 + r_2 + r_1 + r_0$ with $|r_i| \le 1.1 \cdot 2^{32}2^{26i}$ and another sum $u_4 + u_3 + u_2 + u_1 + u_0$ with $|u_j| \le 1.01 \cdot 2^{25}2^{26j}$. It produces a sum $t_4 + t_3 + t_2 + t_1 + t_0$ with $|t_j| \le 29 \cdot 2^{56}2^{26j}$.

The result of the precomputation is a matrix of real numbers $c_{i,j}$ with $|c_{i,j}| \le 1.01 \cdot 2^{25}2^{26j}$ and $r^i \equiv c_{i,4} + \cdots + c_{i,0} \pmod{2^{127} - 1}$. In the main computation, $29 \cdot 2^{56}2^{26j} + 96 \cdot 1.01 \cdot 2^{31}2^{25}2^{26j}$ fits comfortably below $2^{64}2^{26j}$. (In fact, it is below $2^{63}2^{26j}$, so 63-bit floating-point arithmetic suffices.) The final reduction converts from radix 2^{26} to radix 2^{32} and then applies Theorem 4.2 with 53 changed to 64.

Notes. On many computers one can conveniently simulate 63-bit floating-point arithmetic using "64-bit integer arithmetic" or "32-bit integer arithmetic." Some tinkertoy computers do not have *any* built-in hardware for fast multiplication; on these computers it helps to create small multiplication tables. Implementors should be careful to avoid variable-time multiplication hardware and software (including, as an extreme, out-of-cache table lookups indexed by secret data) in cryptographic applications where timing information may be available to an attacker.

8. Guaranteeing the integrity of one message

In this section, p is the prime number $2^{127}-1$; I is the set of integers in the interval $[-2^{127}-2^{95}-2^{63}-2^{31}, 2^{127}-2^{95}-2^{63}-2^{31}-1]$; L is a fixed nonnegative integer; and a **message** is a sequence of length between 0 and L inclusive, consisting of integers in $[-2^{31}, 2^{31}-1]$.

Here is a protocol for authenticated transmission of a single message m through an untrusted channel. The sender and the receiver share a secret uniform random pair (r,k) in $I \times I$. The sender computes an authenticator $s = (k + h_r(m)) \bmod p$ and sends (m,s). At this point an attacker can feed arbitrary pairs (m',s') to the receiver. The receiver accepts (m',s') if $s' = (k + h_r(m')) \bmod p$.

Generating r and k. Let $\rho_0, \rho_1, \rho_2, \rho_3$ be independent uniform random integers in $[-2^{31}, 2^{31}-1]$. Write $r = r_3 + r_2 + r_1 + r_0$ where $r_j = 2^{32j}\rho_j$. Then r is a uniform random element of I. Note that r_0, r_1, r_2, r_3 meet the hypotheses of section 6. Similar comments apply to k.

Encoding byte strings. There are several standard ways to encode a sequence of 4ℓ integers $b_0, b_1, \ldots, b_{4\ell-1}$ in $[0,255]$ as a sequence of ℓ integers $m_0, m_1, \ldots, m_{\ell-1}$ in $[-2^{31}, 2^{31}-1]$. For example, **little-endian twos-complement** defines

$$m_j = ((b_{4j} + 2^8 b_{4j+1} + 2^{16} b_{4j+2} + 2^{24} b_{4j+3} + 2^{31}) \bmod 2^{32}) - 2^{31}.$$

One can encode an arbitrary-length sequence of integers in $[0,255]$ as a sequence of length 4ℓ for some ℓ by appending either 1 or $1,0$ or $1,0,0$ or $1,0,0,0$.

Security. The rest of this section analyzes the attacker's chance of deceiving the receiver. Theorem 8.2 shows that any possible forgery (m',s') with $m' \neq m$ has negligible chance of being accepted by the receiver.

Theorem 8.1. *Let m and m' be distinct messages. Let δ be an integer. Then at most $2L+4$ integers r in I satisfy $h_r(m) \equiv h_r(m') + \delta \pmod p$.*

Proof. Say $m = (m_0, m_1, \ldots, m_{\ell-1})$ and $m' = (m'_0, m'_1, \ldots, m'_{\ell'-1})$.

Define polynomials $g, g' \in \mathbf{Z}[t]$ by $g = t^{\ell+1} + m_0 t^\ell + m_1 t^{\ell-1} + \cdots + m_{\ell-1} t$ and $g' = t^{\ell'+1} + m'_0 t^{\ell'} + m'_1 t^{\ell'-1} + \cdots + m'_{\ell'-1} t + \delta$. Note that the polynomial $g - g'$ is not divisible by p. Otherwise $\ell = \ell'$, and each $m_i - m'_i$ is divisible by p, so $m_i = m'_i$ since $|m_i - m'_i| < 2^{32} < p$, so $m = m'$, contradiction.

If $h_r(m) - h_r(m') - \delta$ is divisible by p then $g(r) - g'(r)$ is divisible by p. Now p is prime, and $g - g'$ has degree at most $L+1$, so $g - g'$ has at most $L+1$ roots modulo p. Hence there are at most $L+1$ choices for r in $\{V, \ldots, V+p-1\}$, and $L+1$ choices for r in $\{V+p, \ldots, V+2p-1\}$, and 2 choices for r in $\{V+2p, V+2p+1\}$, where $V = -2^{127} - 2^{95} - 2^{63} - 2^{31}$; in short, at most $2L+4$ choices for r in $\{V, \ldots, V+2p+1\} = I$. $\square$

Theorem 8.2. *Let m and m' be distinct messages. Let s and s' be integers in $[0, p-1]$. Let r and k be independent uniform random elements of I. Then*

$$\Pr[s = (k + h_r(m)) \bmod p \text{ and } s' = (k + h_r(m')) \bmod p] \leq 3(L+2)/2^{255}$$

and $\Pr[s = (k + h_r(m)) \bmod p] \geq 1/2^{127}$.

Thus the conditional probability that $s' = (k + h_r(m')) \bmod p$, given that $s = (k + h_r(m)) \bmod p$, is at most $3(L+2)/2^{128}$. It doesn't matter how much time the attacker spends staring at (m, s); any choice of (m', s') with $m' \neq m$ has chance at most $3(L+2)/2^{128}$ of being accepted by the receiver. (See Theorem 9.1 for a slightly better bound that takes account of the distribution of s.)

Proof. For each of the 2^{128} choices of r, there are at least two choices of k satisfying $s = (k + h_r(m)) \bmod p$. Thus $\Pr[s = (k + h_r(m)) \bmod p] \geq 2^{129}/2^{256}$.

If $s \equiv k + h_r(m) \pmod p$ and $s' \equiv k + h_r(m')$ then $h_r(m) \equiv h_r(m') + s - s'$. By Theorem 8.1, this happens for at most $2L + 4$ choices of r; and, for each of those choices of r, there are at most three choices of k satisfying $s \equiv k + h_r(m)$. Thus $\Pr[s \equiv k + h_r(m) \text{ and } s' \equiv k + h_r(m')] \leq 3(2L + 4)/2^{256}$. $\square$

Notes. When L is around (say) 2^{30}, the forgery probability in Theorem 8.2 is around $1/2^{96}$. Carter and Wegman in [27, page 149] observed that one can compress the authenticator from 127 bits down to 96 bits with only a small increase in the forgery probability. For example, one can use $\pi((k + h_r(m)u) \bmod (2^{127} - 1))$ as an authenticator, where u is another uniform random element of I, and π is some public function from 127-bit strings to 96-bit strings. This may be useful in applications where every bit of storage is sacred. Beware that the simpler formula $\pi((k + h_r(m)) \bmod (2^{127} - 1))$ is *not* safe; it increases the forgery probability by another factor of 2^{31}, at least for some choices of π.

Some writers claim that forgery probabilities around $1/2^{32}$ are adequate for most applications. The attacker's cost of 2^{32} forgery attempts, they say, is much larger than the attacker's benefit from forging a single message. Unfortunately, even if all attackers acted on the basis of rational economic analyses, this argument would be wrong, because it wildly underestimates the attacker's benefit. In a typical authentication system, as soon as the attacker is lucky enough to succeed at a few forgeries, he can immediately figure out enough secret information to let him forge messages of his choice. (This does not contradict the information-theoretic security expressed by Theorem 8.2; the attacker is gaining information from the receiver, not from the sender.) It is crucial for the forgery probability to be so small that attackers have no hope.

9. Guaranteeing the integrity of many messages

This section explains several protocols for authenticated transmission of at most N messages through an untrusted channel. Here N can be arbitrarily large. Each protocol uses the same hash function h_r; the protocols differ in how they hide $h_r(m)$ from the attacker.

Notation and terminology. As in section 8, p is the prime number $2^{127} - 1$; I is the set of integers in the interval $[-2^{127} - 2^{95} - 2^{63} - 2^{31}, 2^{127} - 2^{95} - 2^{63} - 2^{31} - 1]$; L is a fixed nonnegative integer; and a **message** is a sequence of length between 0 and L inclusive, consisting of integers in $[-2^{31}, 2^{31} - 1]$. Also C, D, N are fixed positive integers with $C \leq N$.

A **chosen-message attack** is a (probabilistic) algorithm that makes C message queries to an oracle, and that then prints D triples (n', m', s'), where each m' is a message different from any of the oracle queries. For any function g, the attack **succeeds for** g if, given an oracle that responds to the nth query m with $g(n, m)$, the attack prints at least one (n', m', s') such that $s' = g(n', m')$.

The reader is assumed to be comfortable with the concept of a random function, i.e., a random variable whose values are functions. For the definition of "random" see, e.g., [18, Appendix]. Note that random variables are not necessarily uniformly distributed.

Protocol 1: one-time pad. In this protocol, the sender and receiver share two independent secrets: a uniform random integer r in I, and a uniform random function f from $\{1,2,\ldots,N\}$ to I. Given the nth message m, the sender computes an authenticator $s = (f(n) + h_r(m)) \bmod p$, and sends (n, m, s). The receiver accepts (n', m', s') if $s' = (f(n') + h_r(m')) \bmod p$.

Theorem 9.1. *Let A be a chosen-message attack. Let f be a uniform random function from $\{1,2,\ldots,N\}$ to I. Let r be a uniform random element of I. If r and f are independent then A succeeds for $(n,m) \mapsto (f(n) + h_r(m)) \bmod p$ with probability at most $(C + D(L+2))/2^{127}$.*

Proof. Let $m_1, \ldots, m_C$ be messages, and let $s_1, \ldots, s_C$ be integers in $[0, p-1]$. Let X be the event that A's oracle queries are $m_1, \ldots, m_C$ in that order; the oracle's responses are $s_1, \ldots, s_C$ respectively; and $f(1), \ldots, f(C) \in [V, V + 2p - 1]$ where $V = -2^{127} - 2^{95} - 2^{63} - 2^{31}$.

Observe that the conditional distribution of r, given X, is uniform. Indeed, for each possible r, there are exactly two choices of $f(n)$ in $[V, V+2p-1]$ satisfying $s_n = (f(n) + h_r(m_n)) \bmod p$.

For any (n', m', s') with $m' \notin \{m_1, m_2, \ldots, m_C\}$, the conditional probability that $s' = (f(n') + h_r(m')) \bmod p$, given X, is at most $(L+2)/2^{127}$. Indeed, if $n' \in \{1, 2, \ldots, C\}$ then $h_r(m_{n'}) - s_{n'} \equiv h_r(m') - s' \pmod{p}$ with conditional probability at most $(L+2)/2^{127}$ by Theorem 8.1, since the conditional distribution of r is uniform. If $n' \notin \{1, 2, \ldots, C\}$ then $s' = (f(n') + h_r(m')) \bmod p$ with conditional probability at most $3/2^{128} < (L+2)/2^{127}$, since the conditional distribution of $f(n')$ is uniform.

Thus A succeeds with conditional probability at most $D(L+2)/2^{127}$ given X. Now sum over all $m_1, \ldots, m_C, s_1, \ldots, s_C$: A succeeds with conditional probability at most $D(L+2)/2^{127}$ given that all of $f(1), f(2), \ldots, f(C)$ are in $[V, V+2p-1]$. Finally, there is probability at most $2C/2^{128}$ that at least one of $f(1), f(2), \ldots, f(C)$ is in $\{V + 2p, V + 2p + 1\}$. □

Protocol 2: scrambling. In this protocol, the sender and receiver share two independent secrets: a uniform random integer r in I, and a uniform random function f from $\{0, 1, \ldots, p-1\}$ to $\{0, 1, \ldots, 2^{128} - 1\}$. Given a message m, the sender computes an authenticator $s = f(h_r(m))$, and sends (m, s). The receiver accepts (m', s') if $s' = f(h_r(m'))$.

Theorem 9.2. *Let A be a chosen-message attack. Let f be a uniform random function from $\{0, 1, \ldots, p-1\}$ to $\{0, 1, \ldots, 2^{128} - 1\}$. Let r be a uniform random element of I. If r and f are independent then A succeeds for $(n, m) \mapsto f(h_r(m))$ with probability at most $(\binom{C}{2} + CD)(L+2)/2^{127} + D/2^{128}$.*

Proof. Let $Q = (m_1, m_2, \ldots, m_n, m'_1, m'_2, \ldots, m'_D)$ be a sequence of messages with $n \le C$, with $m_1, m_2, \ldots, m_n$ all different from each other, and with $m_1, m_2, \ldots, m_n$ different from $m'_1, m'_2, \ldots, m'_D$. Let $R = (s_1, s_2, \ldots, s_n, s'_1, s'_2, \ldots, s'_D)$ be a sequence of integers in $[0, 2^{128} - 1]$.

Let $X_{Q,R}$ be the event that $s_i = f(h_r(m_i))$ for all i and $s'_j \neq f(h_r(m'_j))$ for all j. By Theorem 8.1, the chance that $h_r(m_1), \dots, h_r(m_n)$ are different from each other and different from $h_r(m'_1), \dots, h_r(m'_D)$ is at least $1 - (\binom{n}{2} + nD)(L+2)/2^{127} \geq 1 - \epsilon$ where $\epsilon = (\binom{C}{2} + CD)(L+2)/2^{127}$. In that case $f(h_r(m_1)), \dots, f(h_r(m_n))$ are independent of each other and independent of $f(h_r(m'_1)), \dots, f(h_r(m'_D))$. Hence $\Pr[X_{Q,R}] \geq (1-\epsilon)(1 - D/2^{128})/2^{128n}$.

Let $Y_{Q,R}$ be the event that

- A's distinct oracle queries are $m_1, m_2, \dots, m_n$ in that order: the first oracle query is m_1, the first oracle query different from m_1 is m_2, etc., and there are no oracle queries other than $m_1, m_2, \dots, m_n$;
- the oracle responses to $m_1, m_2, \dots, m_n$ are $s_1, s_2, \dots, s_n$ respectively;
- the messages printed by A are $m'_1, m'_2, \dots, m'_D$ in that order, with forged authenticators $s'_1, s'_2, \dots, s'_D$ respectively; and
- A does not succeed: $s'_1 \neq f(h_r(m'_1))$, $s'_2 \neq f(h_r(m'_2))$, etc.

Observe that the conditional probability of $Y_{Q,R}$ given $X_{Q,R}$ is some quantity $\alpha_{Q,R}$ predetermined by A: it is the chance that A decides to use query m_1, to use query m_2 after query m_1 and response s_1, to use query m_3 after queries m_1, m_2 and responses s_1, s_2, etc., and finally to stop after $m_1, s_1, \dots, m_n, s_n$ and print $m'_1, s'_1, \dots$. Hence $\Pr[Y_{Q,R}] \geq (1-\epsilon)(1 - D/2^{128})\alpha_{Q,R}/2^{128n}$.

If A were used with a uniform random oracle then the query-response-print sequence would be $m_1, s_1, m_2, s_2, \dots, m_n, s_n, m'_1, s'_1, \dots$ with probability exactly $\alpha_{Q,R}/2^{128n}$. Thus $\sum_{n,Q,R} \alpha_{Q,R}/2^{128n} = 1$. Hence the chance that A does not succeed is $\sum \Pr[Y_{Q,R}] \geq (1-\epsilon)(1-D/2^{128}) \sum \alpha_{Q,R}/2^{128n} = (1-\epsilon)(1-D/2^{128})$. □

Protocol 3: one-time scrambling. In this protocol, the sender and receiver share two independent secrets: a uniform random integer r in I, and a uniform random function f from $\{1, 2, \dots, N\} \times \{0, \dots, p-1\}$ to $\{0, \dots, 2^{128} - 1\}$. Given the nth message m, the sender computes an authenticator $s = f(n, h_r(m))$, and sends (n, m, s). The receiver accepts (n', m', s') if $s' = f(n', h_r(m'))$.

Theorem 9.3. *Let A be a chosen-message attack. Let f be a uniform random function from $\{1, 2, \dots, N\} \times \{0, 1, \dots, p-1\}$ to $\{0, 1, \dots, 2^{128} - 1\}$. Let r be a uniform random element of I. If r and f are independent then A succeeds for $(n, m) \mapsto f(n, h_r(m))$ with probability at most $D(2L+5)/2^{128}$.*

Proof. Say the oracle queries are $m_1, m_2, \dots, m_C$. For each possible r, the pairs $(1, h_r(m_1)), (2, h_r(m_2)), \dots, (C, h_r(m_C))$ are all different, so the oracle responses $f(1, h_r(m_1)), f(2, h_r(m_2)), \dots, f(C, h_r(m_C))$ are independent and uniform. Thus the conditional distribution of r, given the oracle responses, is uniform.

Now consider any (n', m', s') with $m' \notin \{m_1, \dots, m_C\}$. If $n' \in \{1, 2, \dots, C\}$ then, by Theorem 8.1, $h_r(m') = h_r(m_{n'})$ with conditional probability at most $(2L+4)/2^{128}$. If that does not happen then $f(n', h_r(m'))$ is independent of the oracle responses, and thus has chance $1/2^{128}$ of equalling s'. The overall chance of success for (n', m', s') is at most $(2L+4)/2^{128} + 1/2^{128}$. □

Random numbers instead of counters. In Theorem 9.1 it is crucial that the sender never reuse a message number. In some applications it is more convenient to use a uniform random element of $\{1, 2, \dots, N\}$ than to use a counter; then N must be chosen large enough that the chance of message-number repetition is negligible.

Similar comments apply to Theorem 9.3. On the other hand, $f(n, h_r(m))$ is much more tolerant of message-number repetition than $(f(n) + h_r(m)) \bmod p$. Even if the same number n is used again and again, the security of $f(n, h_r(m))$ is no worse than the security of $f(h_r(m))$ shown in Theorem 9.2.

Unpredictable random functions. The construction $(f(n)+h_r(m)) \bmod p$ uses a shared secret containing $128 + 128N$ bits of entropy; the construction $f(h_r(m))$ uses $128+128p = 2^{134}$ bits; the construction $f(n, h_r(m))$ uses $128+128pN \approx 2^{134}N$ bits. None of these protocols can be used in practice, except $(f(n) + h_r(m)) \bmod p$ for small N. Fortunately, it seems that uniform random functions can be safely simulated by random functions of much lower entropy.

Consider, for example, the block cipher Rijndael. If k is a 128-bit string then Rijndael_k is a function from S to S, where $S = \{0, 1, \ldots, 2^{128} - 1\}$. If k is a uniform random 128-bit string then Rijndael_k is a good simulation of a uniform random function from S to S; it seems very difficult for an attacker to tell the difference. More precisely: It seems that, for every practical algorithm A that uses an oracle and prints either 0 or 1, there is negligible difference between the average output of A using Rijndael_k as an oracle and the average output of A using a uniform random function as an oracle. In short, Rijndael_k seems to be **unpredictable**.

Consequently it seems safe to use, e.g., $\text{Rijndael}_k(h_r(m))$ as an authenticator of m, where k is a uniform random 128-bit string, r is a uniform random element of I, and r and k are independent. If Rijndael_k is, in fact, unpredictable, then there is no practical attack on this system that achieves significantly better success than the bound in Theorem 9.2.

Similar comments apply to $(f(n) + h_r(m)) \bmod p$ and $f(n, h_r(m))$, where n is a unique message number as above. One variant of $f(n, h_r(m))$ is the authenticator $\text{Rijndael}_k(\text{Rijndael}_k(n) \text{ xor } h_r(m))$; note that if f is unpredictable then $(x, y) \mapsto f(f(x) \text{ xor } y)$ is unpredictable.

There are many other published examples of random functions that can be used instead of Rijndael_k. For example, it is convenient to use $\text{MD5}(k, n, h_r(m))$ as an authenticator, where k is a uniform random 256-bit string, and both n and $h_r(m)$ are encoded as 128-bit strings. If the random function $x \mapsto \text{MD5}(k, x)$ on 256-bit inputs is unpredictable then there is no practical attack on this system achieving significantly better success than the bound in Theorem 9.3. Similar comments apply to the first 128 bits of the output of SHA-1.

Notes. Wegman and Carter in [62, section 4] proposed the form $f(n) + h(m)$ for an authenticator and observed that the forgery probability is proportional to the number of forgery attempts. The bound in Theorem 9.1 has an extra $C/2^{127}$ term to account for the slight nonuniformity of $f(n) \bmod p$.

The idea of simulating a uniform random function with an unpredictable random function is now considered obvious. In the context of $f(n) + h(m)$ it is usually credited to Brassard in [26]. See [18, section 2] for further historical notes on the concept of unpredictability.

Bellare, Canetti, and Krawczyk in [13, section 1.5] proposed the form $f(h(m))$ for an authenticator. They pointed out that, in fact, $m \mapsto f(h(m))$ is unpredictable. For precise bounds on the level of predictability, see, e.g., [18]; my proof of Theorem 9.2 is based on my proof of [18, Theorem 3.1]. For precise bounds on the forgery probability given unpredictability, see, e.g., [17, section 6]. This detour through unpredictability leads to a bound similar to Theorem 9.2 but involving $\binom{C+D}{2}$ instead of $\binom{C}{2} + CD$.

For systems of the form $f(n, h(m))$, see, e.g., [13, section 5] and [39, section 2.3]. Shoup in [56, section 2] analyzed the exact security of several other variants of the Wegman-Carter construction.

The unpredictability of $(x, y) \mapsto f(f(x) \text{ xor } y)$ if f is uniform (and hence if f is unpredictable) was proven by Bellare, Kilian, and Rogaway in [17, section 3]. For a simpler proof see [18, Theorem 3.1].

Rijndael was published by Rijmen and Daemen in [31]. MD5 was published by Rivest in [51]. SHA-1 was published in [3]. The apparent unpredictability of random functions such as $x \mapsto \text{MD5}(k, x)$, for fixed-length x, was pointed out in [15] but had already been used in other cryptographic constructions based on MD5.

There are many other examples in the literature of easily computable, low-entropy random functions that seem to be difficult to predict. Unfortunately, most of the examples were designed under constraints such as invertibility or collision resistance; these constraints distract attention from, and often interfere with, the crucial property of unpredictability. One exception is the Naor-Reingold system in [47], based directly on a difficult number-theoretic problem; unfortunately, this system is too slow for many applications. Another exception is SEAL 3.0, published by Coppersmith and Rogaway in [30]; unfortunately, SEAL's 32-bit input size is too small for most applications.

10. Random hash functions with low collision probability

The crucial property of h_r, for random r, is that it has low collision probability: if $m \neq m'$ then $\Pr[h_r(m) = h_r(m')]$ is small. See Theorem 8.1.

The construction of h_r can be generalized as follows. Select a ring. Select a "large" subset of the ring as the set of messages, and a "small" subset of the ring as the set of hash outputs. Each hash function is determined by an ideal (typically a maximal ideal) of the ring; the hash function takes a message m to an output congruent to m modulo the ideal. Examples:

- Messages are elements of $\mathbf{Z}$. The hash of a message m is $m \bmod p$ where p is a uniform (or nearly uniform) random prime number in $[2^{120}, 2^{128}]$.
- Messages are elements of the polynomial ring $(\mathbf{Z}/2)[t]$. The hash of a message m is m mod p where p is a uniform random irreducible degree-128 polynomial.
- Messages are elements of $F[t]$ where F is a field of size around 2^{32}. The hash of a message m is $m \bmod p$ where p is a uniform random monic irreducible degree-4 polynomial.
- Messages are elements of $F[t]$ where F is a field of size around 2^{128}. The hash of a message m is m mod $(t - r)$ where r is a uniform random element of F; in short, $m(r)$. For example, the hash of $t^\ell + m_0 t^{\ell-1} + m_1 t^{\ell-2} + \cdots + m_{\ell-1}$ is $r^\ell + m_0 r^{\ell-1} + m_1 r^{\ell-2} + \cdots + m_{\ell-1}$.
- Messages are elements of the multivariate polynomial ring $F[t_0, t_1, \ldots]$ over a field F of size around 2^{128}. The hash of m is m mod $(t_0 - r_0, t_1 - r_1, \ldots)$ where $r_0, r_1, \ldots$ are independent uniform random elements of F. For example, the hash of the homogeneous linear polynomial $m_0 t_0 + m_1 t_1 + \cdots + m_{\ell-1} t_{\ell-1} + t_\ell$ is $m_0 r_0 + m_1 r_1 + \cdots + m_{\ell-1} r_{\ell-1} + r_\ell$. As another example, the hash of the multilinear polynomial $m_0 + m_1 t_0 + m_2 t_1 + m_3 t_0 t_1 + \cdots + m_{2^\ell - 1} t_0 t_1 \ldots t_{\ell-1}$ is $m_0 + m_1 r_0 + m_2 r_1 + m_3 r_0 r_1 + \cdots + m_{2^\ell - 1} r_0 r_1 \ldots r_{\ell-1}$.

Each of these random functions can be used in place of h_r in the constructions $f(h_r(m))$ and $f(n, h_r(m))$ discussed in the previous section. Each function, under a small restriction on the set of messages, can also be used in a construction analogous to $(f(n) + h_r(m)) \bmod (2^{127} - 1)$.

How to choose a ring. The hash h_r featured in this paper computes a polynomial modulo $t - r$ in the ring $(\mathbf{Z}/(2^{127} - 1))[t]$. I selected this ring for two reasons:

- Today's computers support much faster multiplication in rings of large (or zero) characteristic, such as $\mathbf{Z}$ or $(\mathbf{Z}/(2^{31} - 1))[t]$ or $(\mathbf{Z}/(2^{127} - 1))[t]$, than in rings of characteristic 2, such as $(\mathbf{Z}/2)[t]$ or $\mathbf{F}_{2^{32}}[t]$ or $\mathbf{F}_{2^{128}}[t]$. It is just as easy to build hardware for $(\mathbf{Z}/2)[t]$ as for $\mathbf{Z}$, but there is much less market demand.
- It is easy to generate a degree-1 monic polynomial over a large field. It takes much more code (and a noticeable amount of time) to generate a higher-degree monic irreducible polynomial over a smaller field, or to generate a prime in $\mathbf{Z}$. Karp and Rabin in [43] suggested allowing non-primes to save time, but that hurts the collision probability.

Higher-dimensional rings such as $(\mathbf{Z}/(2^{127} - 1))[t_0, t_1]$ are as fast as $(\mathbf{Z}/(2^{127} - 1))[t]$ but need more bits to specify a hash function. (They also offer a sublinear collision probability for long messages. However, this is generally not helpful for message authentication; sensible attackers will generate many short messages instead.)

Historical notes. Gilbert, MacWilliams, and Sloane in [37, section 9] observed that a uniform random F-linear function $(m_0, \ldots, m_{\ell-1}) \mapsto m_0 r_0 + \cdots + m_{\ell-1} r_{\ell-1}$ from F^ℓ to F has collision probability $1/\#F$. Some implementation results for $F = \mathbf{Z}/(2^{32} + 15)$, with a few changes in the function for the sake of simpler code, were published by Halevi and Krawczyk in [39]. Black, Halevi, Krawczyk, Krovetz, and Rogaway in [23] reported further implementation results, using Winograd's dot-product algorithm in [64] to trade half the multiplications for additions.

Wegman and Carter in [62, section 3] pointed out that a random hash function of entropy much smaller than L suffices for authentication of messages of length L. The hash $m_0 + m_1 r_0 + m_2 r_1 + m_3 r_0 r_1 + \cdots + m_{2^\ell - 1} r_0 r_1 \ldots r_{\ell-1}$ is Stinson's improvement in [60, section 6] of the Wegman-Carter construction.

Karp and Rabin in [43, section 3], independently of Wegman and Carter, pointed out the low collision probability of large primes in $\mathbf{Z}$. Rabin in [50] (which was written after [43]) made the analogous observation for $(\mathbf{Z}/2)[t]$. Krawczyk in [46] later popularized $(\mathbf{Z}/2)[t]$ as "cryptographic CRCs" and "LFSR-based hashing"; implementation results for this "division hash" were published by Shoup in [56] and by Nevelsteen and Preneel in [48]. Krawczyk also proposed an "LFSR-based Toeplitz matrix" construction, which amounts to $m \mapsto \lfloor (mv \bmod p) t^{128} / p \rfloor$; here v is a nonzero polynomial of degree below 128.

The low collision probability of degree-1 irreducible polynomials in $F[t]$, when F is a large field, was pointed out by den Boer in [24], independently by Taylor in [61, section 3], and independently by Bierbrauer, Johansson, Kabatianskii, and Smeets in [21, section 4]. (Taylor considered only prime fields F, and suggested several good possibilities, including $\mathbf{Z}/(2^{127} - 1)$. Beware that the distribution of polynomials in [61] was extremely far from uniform.) Some implementation results for this "evaluation hash," with F of characteristic 2, were published by Shoup in [56]; by Afanassiev, Gehrmann, and Smeets in [8], reinventing a division algorithm published by Kaminski in [42]; and by Nevelsteen and Preneel in [48]. I published fast code for h_r in April 1999.

Shoup in [56] suggested the "generalized division hash"—irreducible polynomials of any degree over fields of any size—as a common generalization of the "division hash" and the "evaluation hash." Shoup published some implementation results for a field of size 2^8.

Many of these articles follow the terminology of [27] and [62]: a random function f from A to B is "universal" if $\Pr[f(m) = f(m')] \le 1/\#B$ for all pairs of distinct inputs (m, m'). This is not a useful concept; one wants $\Pr[f(m) = f(m')]$ to be small, and one wants $\#B$ to be small, but there is no reason to demand that the product be bounded by 1. Some authors say "f is ϵ-almost universal" to mean that the collision probability of f is at most ϵ.

Generalizations and variants. Let g be a random function on length-ℓ inputs with collision probability at most ϵ. Then $(m_1, \dots, m_k) \mapsto (g(m_1), \dots, g(m_k))$ is a random function on length-$k\ell$ inputs with collision probability at most ϵ. If the output $g(m_1), \dots, g(m_k)$ is too long then one can apply another hash function to it. This is the general idea of the Wegman-Carter construction in [62, section 3]. If g is $(\mathbf{Z}/2)$-linear then one can apply it to 32 inputs in parallel using the 32-bit exclusive-or operation in today's computers; see, e.g., [54], [40], and [48].

Let $g_1, g_2, \dots, g_k$ be independent random functions from length-ℓ inputs to some commutative group. If $\Pr[g_j(m) - g_j(m') = \delta] \le \epsilon$ for all δ, j, m, m' with $m \ne m'$, then $(m_1, m_2, \dots, m_k) \mapsto g_1(m_1) + g_2(m_2) + \dots + g_k(m_k)$ has collision probability at most ϵ. This was pointed out by Zobrist in [65] for the special case of $(j, m) \mapsto g_j(m)$ being uniform; by Gilbert, MacWilliams, and Sloane in [37], as noted above, for the special case of each g_i being a uniform random linear function over a field; and finally by Carter and Wegman in [27, Proposition 8] for the general case. The "XOR hash" published many years later in [15] is the same as Zobrist's hash, but with "unpredictable" in place of "uniform."

Let $g_1, g_2, \dots, g_k$ be independent random functions, each with collision probability at most ϵ. Then $m \mapsto (g_1(m), g_2(m), \dots, g_k(m))$ has collision probability at most ϵ^k. For example, consider messages in F^ℓ where $F = \mathbf{Z}/2$. A uniform random F-linear function from F^ℓ to F has collision probability $1/2$ as noted above, so a uniform random F-linear function from F^ℓ to F^k—i.e., multiplication by a uniform random matrix—has collision probability $(1/2)^k$. This was pointed out by Carter and Wegman in [27, page 151]. Rogaway in [54] pointed out that multiplication by a random *sparse* matrix still has low collision probability if the output is large enough.

Alternate input encodings. Carter and Wegman in [27, page 151] suggested encoding a string of $k\ell$ bits as a string of $2^k\ell$ bits, exactly ℓ of which are 1's.

Karp and Rabin in [43, section 6] suggested encoding a string of bits $b_1, \dots, b_\ell$ as the matrix product $M(b_1) \dots M(b_\ell)$ in the ring of 2×2 matrices over $\mathbf{Z}$, where $M(0) = \left(\begin{smallmatrix} 1 & 0 \\ 1 & 1 \end{smallmatrix}\right)$ and $M(1) = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$; and then reducing the product modulo a big prime p. An improvement is the product $(1\ 1)M(b_1) \dots M(b_\ell)$ in $\mathbf{Z}^2$; the simplest method of computing the image of this product in $(\mathbf{Z}/p)^2$ takes one addition modulo p for each bit of input.

Conjectural constructions and current practice. If p is a uniform random 512-bit string then $m \mapsto \mathrm{MD5}(p, m)$ *appears* to have very low collision probability, as pointed out by Bellare et al. in [15] and in [13]. In fact, nobody has been able to exhibit *any* collisions in MD5. This was the primary goal of MD5. (On the other hand, Dobbertin in [34] found matching-IV collisions in the compression function of MD5.)

Authenticators based on MD5 are popular because they are very fast. The implementations of MD5 in [7] (including the careful Pentium implementation by Bosselaers in [25]) take about 5.3 Pentium cycles per byte, or about 5.9 Pentium-II cycles per byte, or about 9.4 UltraSPARC-I cycles per byte, for long messages. But h_r is even faster!

References

[1] —, *Algolprocedures voor het berekenen van een inwendig product in dubbele precisie*, RC-Informatie nr. 22, Technische Hogeschool Eindhoven (1968).

[2] —, *ALGOL procedures voor het rekenen in dubbele lengte*, RC-Informatie nr. 21, Technische Hogeschool Eindhoven (1969).

[3] —, *Secure hash standard*, Federal Information Processing Standard 180-1, National Institute of Standards and Technology, Washington, 1995.

[4] —, *IEEE standard for binary floating-point arithmetic*, Standard 754–1985, Institute of Electrical and Electronics Engineers, New York, 1985.

[5] —, *37th annual symposium on foundations of computer science*, IEEE Computer Society Press, Los Alamitos, 1996.

[6] —, *38th annual symposium on foundations of computer science*, IEEE Computer Society Press, Los Alamitos, 1997.

[7] —, `OpenSSL 0.9.3`, available from `http://www.openssl.org` (1999).

[8] Valentine Afanassiev, Christian Gehrmann, Ben Smeets, *Fast message authentication using efficient polynomial evaluation*, in [22] (1997), 190–204.

[9] Algirdas A. Avizienis, *Signed-digit number representations for fast parallel arithmetic*, IRE Transactions on Electronic Computers **10** (1961), 389–400.

[10] Mihir Bellare, Ran Canetti, Hugo Krawczyk, *Keying hash functions for message authentication*, in [45] (1996), 16–30.

[11] Mihir Bellare, Ran Canetti, Hugo Krawczyk, *Keying hash functions for message authentication*, draft available from `http://www-cse.ucsd.edu/~mihir/papers/hmac.html`; previous version in [10].

[12] Mihir Bellare, Ran Canetti, Hugo Krawczyk, *Pseudorandom functions revisited: the cascade construction and its concrete security*, in [5] (1996), 514–523.

[13] Mihir Bellare, Ran Canetti, Hugo Krawczyk, *Pseudorandom functions revisited: the cascade construction and its concrete security*, draft available from `http://www-cse.ucsd.edu/~mihir/papers/cascade.html`; previous version in [12].

[14] Mihir Bellare, Roch Guérin, Phillip Rogaway, *XOR MACs: new methods for message authentication using block ciphers*, in [29] (1995), 15–28.

[15] Mihir Bellare, Roch Guérin, Phillip Rogaway, *XOR MACs: new methods for message authentication using block ciphers*, draft available from `http://www-cse.ucsd.edu/~mihir/papers/xormacs.html`; previous version in [14].

[16] Mihir Bellare, Joe Kilian, Phillip Rogaway, *The security of cipher block chaining*, in [33] (1994), 341–358.

[17] Mihir Bellare, Joe Kilian, Phillip Rogaway, *The security of the cipher block chaining message authentication code*, draft available as `http://www-cse.ucsd.edu/~mihir/papers/cbc.ps.gz`; previous version in [16].

[18] Daniel J. Bernstein, *How to stretch random functions: the security of protected counter sums*, Journal of Cryptology **12** (1999), 185–192.

[19] Daniel J. Bernstein, *Multidigit multiplication for mathematicians*, to appear, Advances in Applied Mathematics.

[20] Daniel J. Bernstein, *A secure digital signature system with extremely fast verification*, draft.

[21] Jürgen Bierbrauer, Thomas Johansson, Gregory Kabatianskii, Ben Smeets, *On families of hash functions via geometric codes and concatenation*, in [58] (1994), 331–342.

[22] Eli Biham (editor), *Fast Software Encryption '97*, Lecture Notes in Computer Science 1267, Springer-Verlag, Berlin, 1997.

[23] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, Phillip Rogaway, *UMAC: fast and secure message authentication*, in [63] (1999), 216–233.

[24] Bert den Boer, *A simple and key-economical unconditional authentication scheme*, Journal of Computer Security **2** (1993), 65–71.

[25] Antoon Bosselaers, *Even faster hashing on the Pentium*, available from `ftp://ftp.esat.kuleuven.ac.be/pub/COSIC/bosselae/pentiumplus.ps.gz` (1997).

[26] Gilles Brassard, *On computationally secure authentication tags requiring short secret shared keys*, in [28] (1983), 79–86.

[27] J. Lawrence Carter, Mark N. Wegman, *Universal classes of hash functions*, Journal of Computer and System Sciences **18** (1979), 143–154.

[28] David Chaum, Ronald L. Rivest, Alan T. Sherman (editors), *Advances in cryptology*, Plenum Press, New York, 1983.

[29] Don Coppersmith (editor), *Advances in cryptology—CRYPTO '95*, Lecture Notes in Computer Science 963, Springer-Verlag, Berlin, 1995.

[30] Don Coppersmith, Phillip Rogaway, *A software-optimized encryption algorithm*, Journal of Cryptology **11** (1998), 273–287.

[31] Joan Daemen, Vincent Rijmen, *AES proposal: Rijndael*, draft available from `http://www.esat.kuleuven.ac.be/~rijmen/rijndael/Rijndaeldoc.PDF`.

[32] Theodorus J. Dekker, *A floating-point technique for extending the available precision*, Numerische Mathematik **18** (1971), 224–242.

[33] Yvo Desmedt (editor), *Advances in cryptology—CRYPTO '94*, Lecture Notes in Computer Science 839, Springer-Verlag, Berlin, 1994.

[34] Hans Dobbertin, *Cryptanalysis of MD5 compress*, draft in `http://www.cs.ucsd.edu/users/bsy/dobbertin.ps` (1996).

[35] Joan Feigenbaum (editor), *Advances in cryptology—CRYPTO '91*, Lecture Notes in Computer Science 576, Springer-Verlag, Berlin, 1992.

[36] Walter Fumy (editor), *Advances in cryptology—EUROCRYPT '97*, Lecture Notes in Computer Science 1233, Springer-Verlag, Berlin, 1997.

[37] Edgar N. Gilbert, F. Jessie MacWilliams, Neil J. A. Sloane, *Codes which detect deception*, Bell System Technical Journal **53** (1974), 405–424.

[38] Shai Halevi, Hugo Krawczyk, *MMH: software message authentication in the Gbit/second rates*, in [22] (1997), 172–189.

[39] Shai Halevi, Hugo Krawczyk, *MMH: software message authentication in the Gbit/second rates*, draft; previous version in [38].

[40] Thomas Johansson, *Bucket hashing with a small key size*, in [36] (1997), 149–162.

[41] William M. Kahan, *Further remarks on reducing truncation errors*, Communications of the ACM **8** (1965), 40.

[42] Michael Kaminski, *A linear time algorithm for residue computation and a fast algorithm for division with a sparse divisor*, Journal of the ACM **34** (1987), 968–984.

[43] Richard M. Karp, Michael O. Rabin, *Efficient randomized pattern-matching algorithms*, IBM Journal of Research and Development **31** (1987), 249–260.

[44] Donald E. Knuth, *The art of computer programming, volume 2: seminumerical algorithms*, 2nd edition, Addison-Wesley, Reading, Massachusetts, 1981.

[45] Neal Koblitz (editor), *Advances in cryptology—CRYPTO '96*, Lecture Notes in Computer Science 1109, Springer-Verlag, Berlin, 1996.

[46] Hugo Krawczyk, *LFSR-based hashing and authentication*, in [33] (1994), 129–139.

[47] Moni Naor, Omer Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, in [6] (1997), 458–467.

[48] Wim Nevelsteen, Bart Preneel, *Software performance of universal hash functions*, in [57] (1999), 24–41.

[49] Carl Pomerance, John L. Selfridge, Samuel S. Wagstaff, Jr., *The pseudoprimes to* $25 \cdot 10^9$, Mathematics of Computation **35** (1980), 1003–1026.

[50] Michael O. Rabin, *Fingerprinting by random polynomials*, Harvard Aiken Computational Laboratory TR-15-81 (1981).

[51] Ronald L. Rivest, *The MD5 message-digest algorithm*, Request For Comments 1321; available from `http://theory.lcs.mit.edu/~rivest/rfc1321.txt` (1992).

[52] James E. Robertson, *A new class of digital division methods*, IRE Transactions on Electronic Computers **7** (1958), 218–222.

[53] Phillip Rogaway, *Bucket hashing and its application to fast message authentication*, in [29] (1995), 29–42.

[54] Phillip Rogaway, *Bucket hashing and its application to fast message authentication*, Journal of Cryptology **12** (1999), 91–115; previous version in [53].

[55] Victor Shoup, *On fast and provably secure message authentication based on universal hashing*, in [45] (1996), 313–328.

[56] Victor Shoup, *On fast and provably secure message authentication based on universal hashing*, draft available as `http://www.shoup.net/papers/macs.ps.Z`; previous version in [55].

[57] Jacques Stern (editor), *Advances in cryptology—EUROCRYPT '99*, Lecture Notes in Computer Science 1592, Springer-Verlag, Berlin, 1999.

[58] Douglas R. Stinson (editor), *Advances in cryptology—CRYPTO '93*, Lecture Notes in Computer Science 773, Springer-Verlag, Berlin, 1994.

[59] Douglas R. Stinson, *Universal hashing and authentication codes*, in [35] (1991), 74–85.

[60] Douglas R. Stinson, *Universal hashing and authentication codes*, Designs, Codes and Cryptography **4** (1994), 369–380; previous version in [59].

[61] Richard Taylor, *An integrity check value algorithm for stream ciphers*, in [58] (1994), 40–48.

[62] Mark N. Wegman, J. Lawrence Carter, *New hash functions and their use in authentication and set equality*, Journal of Computer and System Sciences **22** (1981), 265–279.

[63] Michael Wiener (editor), *Advances in cryptology—CRYPTO '99*, Lecture Notes in Computer Science 1666, Springer-Verlag, Berlin, 1999.

[64] Shmuel Winograd, *A new algorithm for inner product*, IEEE Transactions on Computers **17** (1968), 693–694.

[65] Albert L. Zobrist, *A hashing method with applications for game playing*, Technical Report 88, Computer Sciences Department, University of Wisconsin (1970).

Department of Mathematics, Statistics, and Computer Science (M/C 249), The University of Illinois at Chicago, Chicago, IL 60607–7045

E-mail address: `djb@cr.yp.to`