Create account Log in

Article Talk Read Edit More Search

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Српски / srpski
Edit links

Pearson hashing

From Wikipedia, the free encyclopedia

Pearson hashing[1][2] is a hash function designed for fast execution on processors with 8-bit registers. Given an input consisting of any number of bytes, it produces as output a single byte that is strongly dependent[1] on every byte of the input. Its implementation requires only a few instructions, plus a 256-byte lookup table containing a permutation of the values 0 through 255.

This hash function is a CBC-MAC that uses an 8-bit substitution cipher implemented via the substitution table. An 8-bit cipher has negligible cryptographic security, so the Pearson hash function is not cryptographically strong; but it offers these benefits:

- It is extremely simple.
- It executes quickly on resource-limited processors.
- There is no simple class of inputs for which collisions (identical outputs) are especially likely.
- Given a small, privileged set of inputs (e.g., reserved words for a compiler), the permutation table can be adjusted so that those inputs yield distinct hash values, producing what is called a perfect hash function.

One of its drawbacks when compared with other hashing algorithms designed for 8-bit processors is the suggested 256 byte lookup table, which can be prohibitively large for a small microcontroller with a program memory size on the order of hundreds of bytes. A workaround to this is to use a simple permutation function instead of a table stored in program memory. However, using a too simple function, such as *T[i] = 255-i* partly defeats the usability as a hash function as anagrams will result in the same hash value; using a too complex function, on the other hand, will affect speed negatively.

The algorithm can be described by the following pseudocode, which computes the hash of message *C* using the permutation table *T*:

```
h := 0
for each c in C loop
  index := h xor c
  h := T[index]
end loop
return h
```

C implementation to generate 64-bit (16 hex chars) hash [edit]

```
void Pearson16(const unsigned char *x, size_t len, char *hex, size_t hexlen) {
   size_t i, j;
   unsigned char hh[8];
   static const unsigned char T[256] = {
   // 256 values 0-255 in any (random) order suffices
     98,  6, 85,150, 36, 23,112,164,135,207,169,  5, 26, 64,165,219, //  1
     61, 20, 68, 89,130, 63, 52,102, 24,229,132,245, 80,216,195,115, //  2
     90,168,156,203,177,120,  2,190,188,  7,100,185,174,243,162, 10, //  3
    237, 18,253,225,  8,208,172,244,255,126,101, 79,145,235,228,121, //  4
    123,251, 67,250,161,  0,107, 97,241,111,181, 82,249, 33, 69, 55, //  5
     59,153, 29,  9,213,167, 84, 93, 30, 46, 94, 75,151,114, 73,222, //  6
    197, 96,210, 45, 16,227,248,202, 51,152,252,125, 81,206,215,186, //  7
     39,158,178,187,131,136,  1, 49, 50, 17,141, 91, 47,129, 60, 99, //  8
    154, 35, 86,171,105, 34, 38,200,147, 58, 77,118,173,246, 76,254, //  9
    133,232,196,144,198,124, 53,  4,108, 74,223,234,134,230,157,139, // 10
    189,205,199,128,176, 19,211,236,127,192,231, 70,233, 88,146, 44, // 11
    183,201, 22, 83, 13,214,116,109,159, 32, 95,226,140,220, 57, 12, // 12
    221, 31,209,182,143, 92,149,184,148, 62,113, 65, 37, 27,106,166, // 13
      3, 14,204, 72, 21, 41, 56, 66, 28,193, 40,217, 25, 54,179,117, // 14
    238, 87,240,155,180,170,242,212,191,163, 78,218,137,194,175,110, // 15
     43,119,224, 71,122,142, 42,160,104, 48,247,103, 15, 11,138,239  // 16
   };

```

```
        for (j = 0; j < 8; j++) {
            unsigned char h = T[(x[0] + j) % 256];
            for (i = 1; i < len; i++) {
                h = T[h ^ x[i]];
            }
            hh[j] = h;
        }

        snprintf(hex, hexlen, "%02X%02X%02X%02X%02X%02X%02X%02X",
            hh[0], hh[1], hh[2], hh[3],
            hh[4], hh[5], hh[6], hh[7]);
    }
```

For a given string or chunk of data, Pearson's original algorithm produces only an 8 bit byte or integer, 0-255. But the algorithm makes it extremely easy to generate whatever length of hash is desired. The scheme used above is a very straightforward implementation of the algorithm. As Pearson noted: a change to any bit in the string causes his algorithm to create a completely different hash (0-255). In the code above, following every completion of the inner loop, the first byte of the string is incremented by one.

Every time that simple change to the first byte of the data is made, a different Pearson hash, h, is generated. xPear16 builds a 16 hex character hash by concatenating a series of 8-bit Pearson (h) hashes. Instead of producing a value from 0 to 255, it generates a value from 0 to 18,446,744,073,709,551,615.

Pearson's algorithm can be made to generate hashes of any desired length, simply by adding 1 to the first byte of the string, re-computing h for the string, and concatenating the results. Thus the same core logic can be made to generate 32-bit or 128-bit hashes.

References [edit]

1. ^ a b Pearson, Peter K. (June 1990), "Fast Hashing of Variable-Length Text Strings", *Communications of the ACM* **33** (6): 677, doi:10.1145/78973.78978
2. ^ Online PDF file of the CACM paper.

Categories: Error detection and correction | Hash functions

This page was last modified on 25 January 2015, at 03:17.

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Privacy policy About Wikipedia Disclaimers Contact Wikipedia Developers Mobile view