Create account Log in

Article Talk Read Edit View history

WIKIPEDIA
The Free Encyclopedia

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

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
Deutsch
Русский

Fowler–Noll–Vo hash function

From Wikipedia, the free encyclopedia
(Redirected from Fowler Noll Vo hash)

Fowler–Noll–Vo is a non-cryptographic hash function created by Glenn Fowler, Landon Curt Noll, and Phong Vo.

The basis of the FNV hash algorithm was taken from an idea sent as reviewer comments to the IEEE POSIX P1003.2 committee by Glenn Fowler and Phong Vo in 1991. In a subsequent ballot round, Landon Curt Noll improved on their algorithm. Some people tried this hash and found that it worked rather well. In an EMail message to Landon, they named it the *Fowler/Noll/Vo* or FNV hash. [1]

Contents [hide]

Overview [edit]

The current versions are FNV-1 and FNV-1a, which supply a means of creating non-zero FNV offset basis. FNV currently comes in 32-, 64-, 128-, 256-, 512-, and 1024-bit flavors. For pure FNV implementations, this is determined solely by the availability of FNV primes for the desired bit length; however, the FNV webpage discusses methods of adapting one of the above versions to a smaller length that may or may not be a power of two. [2][3]

The FNV hash algorithms and sample FNV source code have been released into the public domain. [4]

FNV is not a cryptographic hash.

The hash [edit]

One of FNV's key advantages is that it is very simple to implement. Start with an initial hash value of FNV offset basis. For each byte in the input, multiply *hash* by the FNV prime, then XOR it with the byte from the input. The alternate algorithm, FNV-1a, reverses the multiply and XOR steps.

FNV-1 hash [edit]

The FNV-1 hash algorithm is as follows: [5]

```
hash = FNV_offset_basis
for each octet_of_data to be hashed
     hash = hash × FNV_prime
     hash = hash XOR octet_of_data
```

```
    return hash
```

In the above pseudocode, all variables are unsigned integers. All variables, except for *octet_of_data*, have the same number of bits as the FNV hash. The variable, *octet_of_data*, is an 8 bit unsigned integer.

As an example, consider the 64-bit FNV-1 hash:

- All variables, except for *octet_of_data*, are 64-bit unsigned integers.
- The variable, *octet_of_data*, is an 8 bit unsigned integer.
- The ***FNV_offset_basis*** is the 64-bit FNV offset basis value: 14695981039346656037.
- The ***FNV_prime*** is the 64-bit FNV prime value: 1099511628211.
- The multiply (indicated by the × symbol) returns the lower 64-bits of the product.
- The XOR is an 8-bit operation that modifies only the lower 8-bits of the hash value.
- The *hash* value returned is a 64-bit unsigned integer.

The values for FNV prime and FNV offset basis may be found in this table.[6]

FNV-1a hash [edit]

The FNV-1a hash differs from the FNV-1 hash by only the order in which the multiply and XOR is performed: [7]

```
    hash = FNV_offset_basis
    for each octet_of_data to be hashed
         hash = hash XOR octet_of_data
         hash = hash × FNV_prime
    return hash
```

The above pseudocode has the same assumptions that were noted for the FNV-1 pseudocode. The small change in order leads to much better avalanche characteristics. [8]

Demonstrated weaknesses [edit]

The FNV hash has a few weaknesses identified by its authors that make it unsuitable as a cryptographic hash function:[9]

- Speed of Computation - As a hash designed primarily for hashtable and checksum use, FNV-1 and 1a were designed to be fast to compute. However, this same speed makes it more feasible for a computer to find hash values (and thus collisions) by brute-force.
- Sticky State - Being an iterative hash based primarily on multiplication and XOR, the algorithm is sensitive to the number zero. Specifically, if the hash value were to become zero at any point during calculation, and the next byte hashed was also all zeroes, the hash would not change. This makes colliding messages trivial to create given a message that results in a hash value of zero at some point in its calculation. Additional operations, such as the addition of a third constant prime on each step, can mitigate this but may have detrimental effects on avalanche effect or random distribution of hash values.
- Diffusion - The ideal secure hash function is one in which each byte of input has an equally-complex effect on every bit of the hash. In the FNV hash, the ones place (the rightmost bit) is always the XOR of the rightmost bit of every input byte. This can be mitigated by XOR-folding (computing a hash twice the desired length, and then XORing the bits in the "upper half" with the bits in the "lower half").

See also [edit]

- Pearson hashing (uses a constant linear permutation instead of a constant prime seed)
- Jenkins hash function

- MurmurHash

Notes [edit]

1. ^ FNV hash history
2. ^ Changing the FNV hash size - xor-folding
3. ^ Changing the FNV hash size - non-powers of 2
4. ^ FNV put into the public domain
5. ^ The core of the FNV hash
6. ^ Parameters of the FNV-1 hash
7. ^ FNV-1a alternate algorithm
8. ^ Avalanche
9. ^ The FNV Non-Cryptographic Hash Algorithm - Why is FNV Non-Cryptographic?

External links [edit]

- Landon Curt Noll's webpage on FNV (with table of base & prime parameters)
- Internet draft by Fowler, Noll, Vo, and Eastlake (2011, work in progress)

Categories: Hash functions

This page was last modified on 7 October 2012 at 21:36.

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details.
Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.

Contact us

Privacy policy About Wikipedia Disclaimers Mobile view