Network Working Group
INTERNET-DRAFT
Intended Status: Informational

Glenn Fowler
Google
Landon Curt Noll
Cisco Systems
Kiem-Phong Vo
Google
Donald Eastlake
Huawei Technologies
Tony Hansen
AT&T Laboratories

Expires: November 28, 2019
May 29, 2019

The FNV Non-Cryptographic Hash Algorithm
<draft-eastlake-fnv-17.txt>

Abstract

FNV (Fowler/Noll/Vo) is a fast, non-cryptographic hash algorithm with good dispersion. The purpose of this document is to make information on FNV and open source code performing FNV conveniently available to the Internet community.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Distribution of this document is unlimited. Comments should be sent to the authors.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html. The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Table of Contents

1. Introduction

The FNV hash algorithm is based on 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 suggested an improvement 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. [FNV]

FNV hashes are designed to be fast while maintaining a low collision rate. The high dispersion of the FNV hashes makes them well suited for hashing nearly identical strings such as URLs, hostnames, filenames, text, IP addresses, etc. Their speed allows one to quickly hash lots of data while maintaining a reasonably low collision rate. However, they are generally not suitable for cryptographic use. (See Section 7.1.)

The FNV hash is widely used, for example in DNS servers, the Twitter service, database indexing hashes, major web search / indexing engines, netnews history file Message-ID lookup functions, anti-spam filters, a spellchecker programmed in Ada 95, flatassembler's open source x86 assembler - user-defined symbol hashtree, non-cryptographic file fingerprints, computing Unique IDs in DASM (DTN (Delay Tolerant Networking) Applications for Symbian Mobile-phones), Microsoft's hash_map implementation for VC++ 2005, the realpath cache in PHP 5.x (php-5.2.3/TSRM/tsrm_virtual_cwd.c), and many other uses.

A study has recommended FNV in connection with the IPv6 Flow Label field [IPv6flow].

FNV hash algorithms and source code have been released into the public domain. The authors of the FNV algorithm took deliberate steps to disclose the algorithm in a public forum soon after it was invented. More than a year passed after this public disclosure and the authors deliberately took no steps to patent the FNV algorithm. Therefore, it is safe to say that the FNV authors have no patent claims on the FNV algorithm as published.

If you use an FNV function in an application, you are kindly requested to send an EMail about it to: fnv-mail@asthe.com

2. FNV Basics

This document focuses on the FNV-1a function whose pseudo-code is as follows:

```
hash = offset_basis
for each octet_of_data to be hashed
        hash = hash xor octet_of_data
        hash = hash * FNV_Prime
return hash
```

In the pseudo-code above, hash is a power-of-two number of bits (32, 64, ... 1024) and offset_basis and FNV_Prime depend on the size of hash.

The FNV-1 algorithm is the same, including the values of offset_basis and FNV_Prime, except that the order of the two lines with the "xor" and multiply operations are reversed. Operational experience indicates better hash dispersion for small amounts of data with FNV-1a. FNV-0 is the same as FNV-1 but with offset_basis set to zero. FNV-1a is suggested for general use.

2.1 FNV Primes

The theory behind FNV_Prime's is beyond the scope of this document but the basic property to look for is how an FNV_Prime would impact dispersion. Now, consider any n-bit FNV hash where n is >= 32 and also a power of 2, in particular n = 2**s. For each such n-bit FNV hash, an FNV_Prime p is defined as:

When s is an integer and 4 < s < 11, then FNV_Prime is the smallest prime p of the form:

```
256**int((5 + 2**s)/12) + 2**8 + b
```

where b is an integer such that:

```
0 < b < 2**8
The number of one-bits in b is 4 or 5
```

and where (p mod (2**40 - 2**24 - 1)) > (2**24 + 2**8 + 2**7).

Experimentally, FNV_Primes matching the above constraints tend to have better dispersion properties. They improve the polynomial feedback characteristic when an FNV_Prime multiplies an intermediate hash value. As such, the hash values produced are more scattered throughout the n-bit hash space.

The case where s < 5 is not considered because the resulting hash quality is too low. Such small hashes can, if desired, be derived from a 32 bit FNV hash by XOR folding (see Section 3). The case where s > 10 is not considered because of the doubtful utility of such large FNV hashes and because the criteria for such large FNV_Primes is more complex, due to the sparsity of such large primes, and would needlessly clutter the criteria given above.

Per the above constraints, an FNV_Prime should have only 6 or 7 one-bits in it. Therefore, some compilers may seek to improve the performance of a multiplication with an FNV_Prime by replacing the multiplication with shifts and adds. However, note that the performance of this substitution is highly hardware-dependent and should be done with care. FNV_Primes were selected primarily for the quality of resulting hash function, not for compiler optimization.

2.2 FNV offset_basis

The offset_basis values for the n-bit FNV-1a algorithms are computed by applying the n-bit FNV-0 algorithm to the 32 octets representing the following character string in [RFC20]:

chongo <Landon Curt Noll> /\../\

The \'s in the above string are not C-style escape characters. In C-string notation, these 32 octets are:

"chongo <Landon Curt Noll> /\\../\\"

That string was used because the person testing FNV with non-zero offset_basis values was looking at an email message from Landon and was copying his standard email signature line; however, they couldn't see very well and copied it incorrectly. In fact, he uses

chongo (Landon Curt Noll) /\oo/\

but, since it doesn't matter, no effort has been made to correct this.

In the general case, almost any offset_basis will serve so long as it is non-zero. The choice of a non-standard offset_basis may be beneficial in defending against some attacks that try to induce hash collisions.

2.3 FNV Endianism

For persistent storage or interoperability between different hardware platforms, an FNV hash shall be represented in the little endian format. That is, the FNV hash will be stored in an array hash[N] with N bytes such that its integer value can be retrieved as follows:

```
unsigned char   hash[N];
for ( i = N-1, value = 0; i >= 0; --i )
    value = ( value << 8 ) + hash[i];
```

Of course, when FNV hashes are used in a single process or a group of processes sharing memory on processors with compatible endian-ness, the natural endian-ness of those processors can be used regardless of its type, little, big, or some other exotic form.

The code provided in Section 6 has FNV hash functions that return a little endian byte vector. Because they are slightly more efficient, code returning FNV hashes of 32-bit or 64-bit size as integers, on computers supporting integers of those sides, are also provided. Such integers are compatible with the same size byte vectors on little endian computers but use of the functions returning integers on big endian or other non-little-endian machines will be byte-reversed or otherwise incompatible with the byte vectors.

3. Other Hash Sizes and XOR Folding

Many hash uses require a hash that is not one of the FNV sizes for which constants are provided in Section 5. If a larger hash size is needed, please contact the authors of this document.

Most hash applications make use of a hash that is a fixed size binary field. Assume that k bits of hash are desired and k is less than 1024 but not one of the sizes for which constants are provided in Section 5. The recommended technique is to take the smallest FNV hash of size S, where S is larger than k, and calculate the desired k-bit-hash using xor folding as shown below. The final bit masking operation is logically unnecessary if the size of the variable k-bit-hash is exactly k bits.

```
temp = FNV_S ( data-to-be-hashed )
k-bit-hash = ( temp xor temp>>k ) bitwise-and ( 2**k - 1 )
```

Hash functions are a trade-off between speed and strength. For example, a somewhat stronger hash may be obtained for exact FNV sizes by calculating an FNV twice as long as the desired output (S = 2*k) and performing such data folding using a k equal to the size of the desired output. However, if a much stronger hash, for example one suitable for cryptographic applications, is wanted, algorithms designed for that purpose, such as those in [RFC6234], should be used.

If it is desired to obtain a hash result that is a value between 0 and max, where max+1 is a not a power of two, simply choose an FNV hash size S such that 2**S > max. Then calculate the following:

```
FNV_S mod ( max+1 )
```

The resulting remainder will be in the range desired but will suffer from a bias against large values with the bias being larger if 2**S is only a little bigger than max. If this bias is acceptable, no further processing is needed. If this bias is unacceptable, it can be avoided by retrying for certain high values of hash, as follows, before applying the mod operation above:

```
X = ( int( ( 2**S - 1 ) / ( max+1 ) ) ) * ( max+1 )
while ( hash >= X )
      hash = ( hash * FNV_Prime ) + offset_basis
```

4. Hashing Multiple Values Together

It is common for there to be a few different component values, say three strings X, Y, and Z, where a hash over all of them is desired. The simplest thing to do is to concatenate them and compute the hash of that concatenation, as in

```
hash ( X | Y | Z )
```

where the vertical bar character ("|") represents string concatenation. Note that, for FNV, the same hash results if X, Y, and Z are actually concatenated and the FNV hash applied to the resulting string or if FNV is calculated on an initial substring and the result used as the offset_basis when calculating the FNV hash of the remainder of the string. This can be done several times. Assuming FNVoffset_basis (v, w) is FNV of w using v as the offset_basis, then in the example above, fnvx = FNV (X) could be calculated and then fnvxy = FNVoffset_basis (fnvx, Y), and finally fnvxyz = FNVoffset_basis (fnvxy, Z). The resulting fnvxyz would be the same as FNV (X | Y | Z);

Cases are also common where such a hash needs to be repeatedly calculated where the component values vary but some vary more frequently than others. For example, assume some sort of computer network traffic flow ID, such as the IPv6 flow ID [RFC6437], is to be calculated for network packets based on the source and destination IPv6 address and the Traffic Class [RFC2460]. If the Flow ID is calculated in the originating host, the source IPv6 address would likely always be the same or perhaps assume one of a very small number of values. By placing this quasi-constant IPv6 source address first in the string being FNV hashed, FNV (IPv6source) could be calculated and used as the offset_basis for calculating FNV of the IPv6 destination address and Traffic Class for each packet. As a result, the per packet hash would be over 17 bytes rather than over 33 bytes saving computational resources. The code in this document includes functions facilitating the use of a non-standard offset_basis.

5. FNV Constants

The FNV Primes are as follows:

```
  32 bit FNV_Prime = 2**24 + 2**8 + 0x93 = 16,777,619
                                         = 0x01000193

  64 bit FNV_Prime = 2**40 + 2**8 + 0xB3 = 1,099,511,628,211
                                       = 0x00000100 000001B3

 128 bit FNV_Prime = 2**88 + 2**8 + 0x3B =
                                309,485,009,821,345,068,724,781,371
                              = 0x00000000 01000000 00000000 0000013B

 256 bit FNV_Prime = 2**168 + 2**8 + 0x63 =
374,144,419,156,711,147,060,143,317,175,368,453,031,918,731,002,211 =
0x0000000000000000 0000010000000000 0000000000000000 0000000000000163

 512 bit FNV_Prime = 2**344 + 2**8 + 0x57 = 35,
835,915,874,844,867,368,919,076,489,095,108,449,946,327,955,754,392,
558,399,825,615,420,669,938,882,575,126,094,039,892,345,713,852,759 =
0x0000000000000000 0000000000000000 0000000001000000 0000000000000000
  0000000000000000 0000000000000000 0000000000000000 0000000000000157

1024 bit FNV_Prime = 2**680 + 2**8 + 0x8D = 5,
016,456,510,113,118,655,434,598,811,035,278,955,030,765,345,404,790,
744,303,017,523,831,112,055,108,147,451,509,157,692,220,295,382,716,
162,651,878,526,895,249,385,292,291,816,524,375,083,746,691,371,804,
094,271,873,160,484,737,966,720,260,389,217,684,476,157,468,082,573 =
0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
  0000000000000000 0000010000000000 0000000000000000 0000000000000000
  0000000000000000 0000000000000000 0000000000000000 0000000000000000
  0000000000000000 0000000000000000 0000000000000000 000000000000018D
```

The FNV offset_basis values are as follows:

```
  32 bit offset_basis = 2,166,136,261 = 0x811C9DC5

  64 bit offset_basis = 14695981039346656037 = 0xCBF29CE4 84222325

 128 bit offset_basis = 144066263297769815596495629667062367629 =
                          0x6C62272E 07BB0142 62B82175 6295C58D

 256 bit offset_basis = 100,029,257,958,052,580,907,070,968,
620,625,704,837,092,796,014,241,193,945,225,284,501,741,471,925,557 =
0xDD268DBCAAC55036 2D98C384C4E576CC C8B1536847B6BBB3 1023B4C8CAEE0535
```

```
 512 bit offset_basis = 9,
659,303,129,496,669,498,009,435,400,716,310,466,090,418,745,672,637,
896,108,374,329,434,462,657,994,582,932,197,716,438,449,813,051,892,
206,539,805,784,495,328,239,340,083,876,191,928,701,583,869,517,785 =
0xB86DB0B1171F4416 DCA1E50F309990AC AC87D059C9000000 0000000000000D21
  E948F68A34C192F6 2EA79BC942DBE7CE 182036415F56E34B AC982AAC4AFE9FD9

1024 bit offset_basis =  14,197,795,064,947,621,068,722,070,641,403,
218,320,880,622,795,441,933,960,878,474,914,617,582,723,252,296,732,
303,717,722,150,864,096,521,202,355,549,365,628,174,669,108,571,814,
760,471,015,076,148,029,755,969,804,077,320,157,692,458,563,003,215,
304,957,150,157,403,644,460,363,550,505,412,711,285,966,361,610,267,
868,082,893,823,963,790,439,336,411,086,884,584,107,735,010,676,915 =
0x0000000000000000 005F7A76758ECC4D 32E56D5A591028B7 4B29FC4223FDADA1
  6C3BF34EDA3674DA 9A21D90000000000 0000000000000000 0000000000000000
  0000000000000000 0000000000000000 0000000000000000 000000000004C6D7
  EB6E73802734510A 555F256CC005AE55 6BDE8CC9C6A93B21 AFF4B16C71EE90B3
```

6. The Source Code

[THIS CODE IS BEING WORKING AND IS INCONSISTENT AS BELOW.]

The following sub-sections provide reference C source code and a test driver for FNV-1a.

Alternative source code, including 32 and 64 bit FNV-1 and FNV-1a in x86 assembler, is currently available at [FNV].

Section 6.2 provides a test driver.

6.1 FNV-1a C Code

This section provides the direct FNV-1a function for each of the lengths for which it is specified in this document. The functions provided are listed below. Those whose name is of the form FNVxxxB* output a byte vector that will be compatible between systems of different endian-ness, where xxx is "32", "64", "128", "256", "512", or "1024". Those whose name is of the form FNVxxx* (with no "B" after the xxx) return an integer and are NOT compatible between systems of different endian-ness. These integer based functions exist only for xxx of "32" and, on systems supporting 64-bit integers, "64".

FNVxxxstring, FNVxxxblock:
FNVxxxBstring, FNVxxxBblock: These are simple functions for directly returning the FNV hash of a zero terminated byte string not including the zero and the FNV hash of a counted block of bytes. Note that for applications of FNV-32 where 32-bit integers are supported and FNV-64 where 64-bit integers are supported and an integer data type output is acceptable, the code is sufficiently simple that, to maximize performance, use of open coding or macros may be more appropriate than calling a subroutine.

FNVxxxinit, FNVxxxinitBasis:
FNVxxxBinit, FNVxxxBinitBasis: These functions and the next two sets of functions below provide facilities for incrementally calculating FNV hashes. They all assume a data structure of type FNVxxx(B)context that holds the current state of the hash. FNVxxx(B)init initializes that context to the standard offset_basis. FNVxxx(B)initBasis takes an offset_basis value as a parameter and may be useful for hashing concatenations, as described in Section 4, as well as for simply using a non-standard offset_basis.

FNVxxxblockin, FNVxxxstringin:
FNVxxxBblockin, FNVxxxBstringin: These functions hash a sequence of bytes into an FNVxxx(B)context that was originally initialized by FNVxxx(B)init or FNVxxx(B)initBasis. FNVxxx(B)blockin hashes in a counted block of bytes. FNVxxx(B)stringin hashes in a zero terminated byte string not incuding the final zero.

FNVxxxresult:
FNVxxxBresult: This function extracts the final FNV hash result from an FNVxxx(B)context.

The following code is a private header file used by all the FNV functions further below and which states the terms for use and redistribution of all of this code.

<CODE BEGINS>

```
/************************ fnv-private.h ************************/
/****************** See RFC NNNN for details ******************/
/* Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * *  Neither the name of Internet Society, IETF or IETF Trust, nor the
 *    names of specific contributors, may be used to endorse or promote
 *    products derived from this software without specific prior
 *    written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
 * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
 * COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
```

```
 */

#ifndef _FNV_PRIVATE_H_
#define _FNV_PRIVATE_H_

/*
 *      Six FNV-1a hashes are defined with these sizes:
 *              FNV32          32 bits, 4 bytes
 *              FNV64          64 bits, 8 bytes
 *              FNV128         128 bits, 16 bytes
 *              FNV256         256 bits, 32 bytes
 *              FNV512         512 bits, 64 bytes
 *              FNV1024        1024 bits, 128 bytes
 */

/* Private stuff used by this implementation of the FNV
 * (Fowler, Noll, Vo) non-cryptographic hash function FNV-1a.
 * External callers don't need to know any of this.  */

enum {  /* State value bases for context->Computed */
    FNVinited = 22,
    FNVcomputed = 76,
    FNVemptied = 220,
    FNVclobber = 122 /* known bad value for testing */
};

/* Deltas to assure distinct state values for different lengths */
enum {
   FNV32state = 1,
   FNV32Bstate = 17,
   FNV64state = 3,
   FNV64Bstate = 19,
   FNV128state = 5,
   FNV256state = 7,
   FNV512state = 11,
   FNV1024state = 13
};

#endif
```

<CODE ENDS>

The following code is a simple header file to include all the specific length FNV header files.

<CODE BEGINS>

```
/****************************** FNV.h ******************************/
/****************** See RFC NNNN for details. ********************/
/*
 * Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
```

```
 * See fnv-private.h for terms of use and redistribution.
 */

#ifndef _FNV_H_
#define _FNV_H_

#include "FNV32.h"
#include "FNV32B.h"
#ifdef FNV_64bitIntegers
# include "FNV64.h"
#endif /* FNV_64bitIntegers */
#include "FNV64B.h"
#include "FNV128.h"
#include "FNV256.h"
#include "FNV512.h"
#include "FNV1024.h"

#endif /* _FNV_H_ */
```

<CODE ENDS>

The following code is a simple header file to control configuration related to big integer and big endian support.

<CODE BEGINS>

```
/************************** FNVconfig.h ***************************/
/******************* See RFC NNNN for details. ********************/
/*
 * Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

#ifndef _FNVconfig_H_
#define _FNVconfig_H_

/*
 *  Description:
 *      This file provides configuration ifdefs for the
 *      FNV-1a non-cryptographic hash algorithms.
 *
 * >>>>>>>> IMPORTANT CONFIGURATION ifdefs: <<<<<<<<<< */

/*      FNV_64bitIntegers - Define this if your system supports 64-bit
 *          arithmetic including 32-bit x 32-bit multiplication
 *          producing a 64-bit product. If undefined, it will be
 *          assumed that 32-bit arithmetic is supported including
 *          16-bit x 16-bit multiplication producing a 32-bit result.
 */
// #define FNV_64bitIntegers
```

```
/*
 *      The following allow the FNV test program to override the
 *      above configuration settings.
 */

#ifdef FNV_TEST_PROGRAM
# ifdef TEST_FNV_64bitIntegers
#  ifndef FNV_64bitIntegers
#   define FNV_64bitIntegers
#  endif
# else
#  undef FNV_64bitIntegers
# endif
# ifndef FNV_64bitIntegers /* causes an error if uint64_t is used */
#  define uint64_t foobar /* RFC 3092 */
# endif
#endif

#endif /* _FNVconfig_H_ */
```

<CODE ENDS>

6.1.1 FNV32 Code

The header and C source for 32-bit FNV-1a returning a 32-bit integer.

<CODE BEGINS>

```
/***************************** FNV32.h *****************************/
/******************* See RFC NNNN for details ********************/
/*
 * Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
*/

#ifndef _FNV32_H_
#define _FNV32_H_

#include "FNVconfig.h"

#include <stdint.h>
/* #define FNV32size (32/8)   not used */

/* If you do not have the ISO standard stdint.h header file, then you
 * must typedef the following types:
 *
 *    type              meaning
 *  uint32_t         unsigned 32 bit integer
 *  uint8_t          unsigned 8 bit integer (i.e., unsigned char)
```

```
 */

#ifndef _FNV_ErrCodes_
#define _FNV_ErrCodes_
/*********************************************************************
 *  All FNV functions provided return as integer as follows:
 *       0 -> success
 *      >0 -> error as listed below
 */
enum {    /* success and errors */
    fnvSuccess = 0,
    fnvNull,            /* Null pointer parameter */
    fnvStateError,      /* called Input after Result, etc. */
    fnvBadParam         /* passed a bad parameter */
};
#endif /* _FNV_ErrCodes_ */

/*
 *  This structure holds context information for an FNV32 hash
 */
typedef struct FNV32context_s {
    int Computed;  /* state */
    uint32_t Hash;
} FNV32context;

/*
 *  Function Prototypes
 *    FNV32string: hash a zero terminated string not including
 *                 the terminating zero
 *    FNV32block: hash a specified length byte vector
 *    FNV32init:  initializes an FNV32 context
 *    FNV32initBasis: initializes an FNV32 context with a
 *                    provided basis
 *    FNV32blockin:  hash in a specified length byte vector
 *    FNV32stringin: hash in a zero terminated string not
 *                   including the zero
 *    FNV32result: returns the hash value
 *
 *    Hash is returned as a 32-bit integer
 */

#ifdef __cplusplus
extern "C" {
#endif

/* FNV32 */
extern int FNV32string ( const char *in,
                         uint32_t * const out );
extern int FNV32block ( const void *in,
                        long int inlength,
```

```
                        uint32_t * const out );
extern int FNV32init ( FNV32context * const );
extern int FNV32initBasis ( FNV32context * const,
                            uint32_t basis );
extern int FNV32blockin ( FNV32context * const,
                          const void *in,
                          long int inlength );
extern int FNV32stringin ( FNV32context * const,
                           const char *in );
extern int FNV32result ( FNV32context * const,
                         uint32_t * const out );

#ifdef __cplusplus
}
#endif

#endif /* _FNV32_H_ */
```

<CODE ENDS>

<CODE BEGINS>

```
/***************************** FNV32.c ****************************/
/****************** See RFC NNNN for details. *********************/
/* Copyright (c) 2016, 2017 IETF Trust and the persons identified
 * as authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

/* This code implements the FNV (Fowler, Noll, Vo) non-cryptographic
 * hash function FNV-1a for 32-bit hashes returning a 32-bit
 * integer.
 */

#ifndef _FNV32_C_
#define _FNV32_C_

#include "fnv-private.h"
#include "FNV32.h"

/* 32 bit FNV_prime = 2^24 + 2^8 + 0x93 */
#define FNV32prime 0x01000193
#define FNV32basis 0x811C9DC5

/* FNV32 hash a zero terminated string not including the zero
*****************************************************************/
int FNV32string ( const char *in, uint32_t * const out )
{
uint32_t    temp;
uint8_t     ch;

if ( in && out )
```

```
    {
    temp = FNV32basis;
    while ( (ch = *in++) )
        temp = FNV32prime * ( temp ^ ch );
    *out = temp;
    return fnvSuccess;
    }
return fnvNull; /* Null input pointer */
}   /* end FNV32string */

/* FNV32 hash a counted block
 *************************************************************/
int FNV32block ( const void *vin,
                 long int length,
                 uint32_t * const out )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t    temp;

if ( in && out )
    {
    if ( length < 0 )
        return fnvBadParam;
    for ( temp = FNV32basis; length > 0; length-- )
        temp = FNV32prime * ( temp ^ *in++ );
    *out = temp;
    return fnvSuccess;
    }
return fnvNull; /* Null input pointer */
}   /* end FNV32block */


/*************************************************************
 *       Set of init, input, and output functions below      *
 *       to incrementally compute FNV32                      *
 *************************************************************/

/* initialize context
 *************************************************************/
int FNV32init ( FNV32context * const ctx )
{
return FNV32initBasis ( ctx, FNV32basis );
}   /* end FNV32init */

/* initialize context with a provided basis
 *************************************************************/
int FNV32initBasis ( FNV32context * const ctx, uint32_t basis )
{
if ( ctx )
    {
```

```
    ctx->Hash = basis;
    ctx->Computed = FNVinited+FNV32state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV32initBasis */

/* hash in a counted block
 *************************************************************/
int FNV32blockin ( FNV32context * const ctx,
                   const void *vin,
                   long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t    temp;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV32state:
            ctx->Computed = FNVcomputed+FNV32state;
        case FNVcomputed+FNV32state:
            break;
        default:
            return fnvStateError;
        }
    for ( temp = ctx->Hash; length > 0; length-- )
        temp = FNV32prime * ( temp ^ *in++ );
    ctx->Hash = temp;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV32blockin */

/* hash in a zero terminated string not including the zero
 *************************************************************/
int FNV32stringin ( FNV32context * const ctx,
                    const char *in )
{
uint32_t    temp;
uint8_t     ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV32state:
```

```
            ctx->Computed = FNVcomputed+FNV32state;
        case FNVcomputed+FNV32state:
            break;
        default:
            return fnvStateError;
        }
    temp = ctx->Hash;
    while ( (ch = (uint8_t)*in++) )
        temp = FNV32prime * ( temp ^ ch );
    ctx->Hash = temp;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV32stringin */

/* return hash
 *************************************************************/
int FNV32result ( FNV32context * const ctx,
                  uint32_t * const out )
{
if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV32state )
        return fnvStateError;
    ctx->Computed = FNVemptied+FNV32state;
    *out = ctx->Hash;
    ctx->Hash = 0;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV32result */

#endif    /* _FNV32_C_ */
```

<CODE ENDS>

The header and C source for 32-bit FNV-1a returning a byte vector.

<CODE BEGINS>

```
/***************************** FNV32B.h *****************************/
/******************* See RFC NNNN for details ***********************/
/*
 * Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
*/

#ifndef _FNV32_H_
#define _FNV32_H_

#include "FNVconfig.h"
```

```
#include <stdint.h>
#define FNV32size (32/8)

/* If you do not have the ISO standard stdint.h header file, then you
 * must typedef the following types:
 *
 *    type              meaning
 *  uint32_t         unsigned 32 bit integer
 *  uint8_t          unsigned 8 bit integer (i.e., unsigned char)
 */

#ifndef _FNV_ErrCodes_
#define _FNV_ErrCodes_
/*********************************************************************
 *  All FNV functions provided return as integer as follows:
 *       0 -> success
 *      >0 -> error as listed below
 */
enum {    /* success and errors */
    fnvSuccess = 0,
    fnvNull,            /* Null pointer parameter */
    fnvStateError,      /* called Input after Result, etc. */
    fnvBadParam         /* passed a bad parameter */
};
#endif /* _FNV_ErrCodes_ */

/*
 *  This structure holds context information for an FNV32 hash
 */
typedef struct FNV32Bcontext_s {
    int Computed;  /* state */
    uint32_t Hash;
} FNV32Bcontext;

/*
 *  Function Prototypes
 *    FNV32Bstring: hash a zero terminated string not including
 *                  the terminating zero
 *    FNV32Bblock: hash a specified length byte vector
 *    FNV32Binit:  initializes an FNV32 context
 *    FNV32BinitBasis: initializes an FNV32 context with a
 *                     provided basis
 *    FNV32Bblockin:  hash in a specified length byte vector
 *    FNV32Bstringin: hash in a zero terminated string not
 *                    including the zero
 *    FNV32Bresult: returns the hash value
 *
 *    Hash is returned as a 32-bit integer
 */
```

```
#ifdef __cplusplus
extern "C" {
#endif

/* FNV32 */
extern int FNV32Bstring ( const char *in,
                          uint8_t * const out[FNV32size] );
extern int FNV32Bblock ( const void *in,
                         long int inlength,
                         uint8_t * const out[FNV32size] );
extern int FNV32Binit ( FNV32Bcontext * const );
extern int FNV32BinitBasis ( FNV32Bcontext * const,
                             uint32_t basis );
extern int FNV32Bblockin ( FNV32Bcontext * const,
                           const void *in,
                           long int inlength );
extern int FNV32Bstringin ( FNV32Bcontext * const,
                            const char *in );
extern int FNV32Bresult ( FNV32Bcontext * const,
                          int8_t * const out[FNV32size] );

#ifdef __cplusplus
}
#endif

#endif /* _FNV32_H_ */
   <CODE ENDS>

   <CODE BEGINS>
/***************************** FNV32B.c ***************************/
/****************** See RFC NNNN for details. ********************/
/* Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

/* This code implements the FNV (Fowler, Noll, Vo) non-cryptographic
 * hash function FNV-1a for 32-bit hashes returning a byte vector.
 */

#ifndef _FNV32_C_
#define _FNV32_C_

#include "fnv-private.h"
#include "FNV32B.h"

/* 32 bit FNV_prime = 2^24 + 2^8 + 0x93 */
#define FNV32prime 0x01000193
#define FNV32basis 0x811C9DC5
```

```
/* FNV32 hash a zero terminated string not including the zero
*****************************************************************/
int FNV32Bstring ( const char *in, uint8_t * const out[FNV32size] )
{
uint32_t    temp;
uint8_t     ch;

if ( in && out )
    {
    temp = FNV32basis;
    while ( (ch = *in++) )
        temp = FNV32prime * ( temp ^ ch );
    *out[0] = temp & 0xFF;
    temp =>> 8;
    *out[1] = temp & 0xFF;
    temp =>> 8;
    *out[2] = temp & 0xFF;
    temp =>> 8;
    *out[3] = temp & 0xFF;
    return fnvSuccess;
    }
return fnvNull; /* Null input pointer */
}   /* end FNV32Bstring */

/* FNV32 hash a counted block
 ****************************************************************/
int FNV32Bblock ( const void *vin,
                  long int length,
                  uint8_t * const out[FNV32size] )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t    temp;

if ( in && out )
    {
    if ( length < 0 )
        return fnvBadParam;
    for ( temp = FNV32basis; length > 0; length-- )
        temp = FNV32prime * ( temp ^ *in++ );
    *out[0] = temp & 0xFF;
    temp =>> 8;
    *out[1] = temp & 0xFF;
    temp =>> 8;
    *out[2] = temp & 0xFF;
    temp =>> 8;
    *out[3] = temp & 0xFF;
    *out = temp;
    return fnvSuccess;
    }
return fnvNull; /* Null input pointer */
```

```
}   /* end FNV32Bblock */


/*************************************************************
 *       Set of init, input, and output functions below      *
 *       to incrementally compute FNV32                      *
 *************************************************************/

/* initialize context
 *************************************************************/
int FNV32Binit ( FNV32Bcontext * const ctx )
{
return FNV32BinitBasis ( ctx, FNV32basis );
}   /* end FNV32Binit */

/* initialize context with a provided basis
 *************************************************************/
int FNV32BinitBasis ( FNV32Bcontext * const ctx, uint32_t basis )
{
if ( ctx )
    {
    ctx->Hash = basis;
    ctx->Computed = FNVinited+FNV32Bstate;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV32BinitBasis */

/* hash in a counted block
 *************************************************************/
int FNV32Bblockin ( FNV32Bcontext * const ctx,
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t    temp;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV32Bstate:
            ctx->Computed = FNVcomputed+FNV32Bstate;
        case FNVcomputed+FNV32Bstate:
            break;
        default:
            return fnvStateError;
        }
```

```
    for ( temp = ctx->Hash; length > 0; length-- )
        temp = FNV32prime * ( temp ^ *in++ );
    ctx->Hash = temp;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV32Bblockin */

/* hash in a zero terminated string not including the zero
 ************************************************************/
int FNV32Bstringin ( FNV32Bcontext * const ctx,
                     const char *in )
{
uint32_t    temp;
uint8_t     ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV32Bstate:
            ctx->Computed = FNVcomputed+FNV32Bstate;
        case FNVcomputed+FNV32Bstate:
            break;
        default:
            return fnvStateError;
        }
    temp = ctx->Hash;
    while ( (ch = (uint8_t)*in++) )
        temp = FNV32prime * ( temp ^ ch );
    ctx->Hash = temp;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV32Bstringin */

/* return hash
 ************************************************************/
int FNV32Bresult ( FNV32Bcontext * const ctx,
                   uint8_t * const out[FNV32size] )
{
if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV32Bstate )
        return fnvStateError;
    ctx->Computed = FNVemptied+FNV32Bstate;
    *out[0] = ctx->Hash & 0xFF;
    ctx->Hash =>> 8;
    *out[1] = ctx->Hash & 0xFF;
    ctx->Hash =>> 8;
```

```
    *out[2] = ctx->Hash & 0xFF;
    ctx->Hash =>> 8;
    *out[3] = ctx->Hash & 0xFF;
    ctx->Hash = 0;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV32Bresult */

#endif    /* _FNV32_C_ */
```

<CODE ENDS>

6.1.2 FNV64 C Code

The header and C source for 64-bit FNV-1a returning a 64-bit integer.

<CODE BEGINS>

```
/***************************** FNV64.h *****************************/
/****************** See RFC NNNN for details. ********************/
/*
 * Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

#ifndef _FNV64_H_
#define _FNV64_H_

/*
 *  Description:
 *      This file provides headers for the 64-bit version of the FNV-1a
 *      non-cryptographic hash algorithm.
 */

#include "FNVconfig.h"

#include <stdint.h>
#define FNV64size (64/8)

/* If you do not have the ISO standard stdint.h header file, then you
 * must typedef the following types:
 *
 *    type             meaning
 *  uint64_t        unsigned 64 bit integer (ifdef FNV_64bitIntegers)
 *  uint32_t        unsigned 32 bit integer
 *  uint16_t        unsigned 16 bit integer
 *  uint8_t         unsigned 8 bit integer (i.e., unsigned char)
 */
```

```
#ifndef _FNV_ErrCodes_
#define _FNV_ErrCodes_
/*********************************************************************
 *  All FNV functions provided return as integer as follows:
 *       0 -> success
 *      >0 -> error as listed below
 */
enum {    /* success and errors */
    fnvSuccess = 0,
    fnvNull,            /* Null pointer parameter */
    fnvStateError,      /* called Input after Result, etc. */
    fnvBadParam         /* passed a bad parameter */
};
#endif /* _FNV_ErrCodes_ */

/*
 *  This structure holds context information for an FNV64 hash
 */
#ifdef FNV_64bitIntegers
    /* version if 64 bit integers supported */

typedef struct FNV64context_s {
        int Computed;  /* state */
        uint64_t Hash;
} FNV64context;

#else
    /* version if 64 bit integers NOT supported */

typedef struct FNV64context_s {
        int Computed;  /* state */
        uint16_t Hash[FNV64size/2];
} FNV64context;

#endif /* FNV_64bitIntegers */

/*
 *  Function Prototypes
 *    FNV64string: hash a zero terminated string not including
 *                 the terminating zero
 *    FNV64block: FNV64 hash a specified length byte vector
 *    FNV64init: initializes an FNV64 context
 *    FNV64initBasis: initializes an FNV64 context with a
 *                    provided basis
 *    FNV64blockin: hash in a specified length byte vector
 *    FNV64stringin: hash in a zero terminated string not
 *                   incluing the zero
 *    FNV64result: returns the hash value
 *
 *    Hash is returned as a 64-bit integer if supported, otherwise
```

```
 *         as a vector of 8-bit integers
 */

#ifdef __cplusplus
extern "C" {
#endif

/* FNV64 */
extern int FNV64init ( FNV64context * const );
extern int FNV64blockin ( FNV64context * const,
                          const void * in,
                          long int length );
extern int FNV64stringin ( FNV64context * const,
                           const char * in );

#ifdef FNV_64bitIntegers
  extern int FNV64string ( const char *in,
                           uint64_t * const out );
  extern int FNV64block ( const void *in,
                          long int length,
                          uint64_t * const out );
  extern int FNV64initBasis ( FNV64context * const,
                            uint64_t basis );
  extern int FNV64result ( FNV64context * const,
                           uint64_t * const out );
#else
  extern int FNV64string ( const char *in,
                           uint8_t out[FNV64size] );
  extern int FNV64block ( const void *in,
                          long int length,
                          uint8_t out[FNV64size] );
  extern int FNV64initBasis ( FNV64context * const,
                            const uint8_t basis[FNV64size] );
  extern int FNV64result ( FNV64context * const,
                           uint8_t out[FNV64size] );
#endif /* FNV_64bitIntegers */

#ifdef __cplusplus
}
#endif

#endif /* _FNV64_H_ */
   <CODE ENDS>

   <CODE BEGINS>
/***************************** FNV64.c ******************************/
/******************** See RFC NNNN for details **********************/
/* Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
```

```
 */

/* This file implements the FNV (Fowler, Noll, Vo) non-cryptographic
 * hash function FNV-1a for 64-bit hashes.
 */

#ifndef _FNV64_C_
#define _FNV64_C_

#include "fnv-private.h"
#include "FNV64.h"

/*****************************************************************
 *        START VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC       *
 *****************************************************************/
#ifdef FNV_64bitIntegers

/* 64 bit FNV_prime = 2^40 + 2^8 + 0xb3 */
#define FNV64prime 0x00000100000001B3
#define FNV64basis 0xCBF29CE484222325

/* FNV64 hash a null terminated string  (64 bit)
 *****************************************************************/
int FNV64string ( const char *in, uint64_t * const out )
{
uint64_t    temp;
uint8_t     ch;

if ( in && out )
    {
    temp = FNV64basis;
    while ( (ch = *in++) )
        temp = FNV64prime * ( temp ^ ch );
#ifdef FNV_BigEndian
    FNV64reverse ( out, temp );
#else
    *out = temp;
#endif
    return fnvSuccess;
    }
return fnvNull; /* Null input pointer */
}   /* end FNV64string */

/* FNV64 hash a counted block  (64 bit)
 *****************************************************************/
int FNV64block ( const void *vin,
                 long int length,
                 uint64_t * const out )
{
const uint8_t *in = (const uint8_t*)vin;
```

```
uint64_t    temp;

if ( in && out )
    {
    if ( length < 0 )
        return fnvBadParam;
    for ( temp = FNV64basis; length > 0; length-- )
        temp = FNV64prime * ( temp ^ *in++ );
#ifdef FNV_BigEndian
    FNV64reverse ( out, temp );
#else
    *out = temp;
#endif
    return fnvSuccess;
    }
return fnvNull; /* Null input pointer */
}   /* end FNV64block */

#ifdef FNV_BigEndian

/* Store a Big Endian result back as Little Endian
 ************************************************************/
static void FNV64reverse ( uint64_t *out, uint64_t hash )
{
uint64_t    temp;
int         i;

temp = hash & 0xFF;
for ( i = FNV64size - 1; i > 0; i-- )
    {
    hash >>= 8;
    temp = ( temp << 8 ) + ( hash & 0xFF );
    }
*out = temp;
}   /* end FNV64reverse */

#endif /* FNV_BigEndian */


/*****************************************************************
 *         Set of init, input, and output functions below        *
 *         to incrementally compute FNV64                        *
 *****************************************************************/

/* initialize context  (64 bit)
 ****************************************************************/
int FNV64init ( FNV64context * const ctx )
{
return FNV64initBasis ( ctx, FNV64basis );
}       /* end FNV64init */
```

```
/* initialize context with a provided basis  (64 bit)
 **********************************************************************/
int FNV64initBasis ( FNV64context * const ctx, uint64_t basis )
{
if ( ctx )
    {
    ctx->Hash = basis;
    ctx->Computed = FNVinited+FNV64state;
    return fnvSuccess;
    }
return fnvNull;
}       /* end FNV64initBasis */

/* hash in a counted block  (64 bit)
 **********************************************************************/
int FNV64blockin ( FNV64context * const ctx,
                   const void *vin,
                   long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint64_t    temp;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV64state:
            ctx->Computed = FNVcomputed+FNV64state;
        case FNVcomputed+FNV64state:
            break;
        default:
            return fnvStateError;
        }
    for ( temp = ctx->Hash; length > 0; length-- )
        temp = FNV64prime * ( temp ^ *in++ );
    ctx->Hash = temp;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64input */

/* hash in a zero terminated string not including the zero (64 bit)
 **********************************************************************/
int FNV64stringin ( FNV64context * const ctx,
                    const char *in )
{
uint64_t        temp;
uint8_t         ch;
```

```
if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV64state:
            ctx->Computed = FNVcomputed+FNV64state;
        case FNVcomputed+FNV64state:
            break;
        default:
             return fnvStateError;
         }
    temp = ctx->Hash;
    while ( (ch = *in++) )
        temp = FNV64prime * ( temp ^ ch );
    ctx->Hash = temp;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64stringin */

/* return hash  (64 bit)
 ***********************************************************************/
int FNV64result ( FNV64context * const ctx,
                  uint64_t * const out )
{
if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV64state )
        return fnvStateError;
    ctx->Computed = FNVemptied+FNV64state;
#ifdef FNV_BigEndian
    FNV64reverse ( out, ctx->Hash );
#else
    *out = ctx->Hash;
#endif
    ctx->Hash = 0;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64result */

/*****************************************************************
 *        END VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC        *
 *****************************************************************/
#else    /*  FNV_64bitIntegers */
/*****************************************************************
 *     START VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC    *
 *****************************************************************/

/* 64 bit FNV_prime = 2^40 + 2^8 + 0xb3 */
```

```
/* #define FNV64prime 0x00000100000001B3 */
#define FNV64primeX 0x01B3
#define FNV64shift 8

/* #define FNV64basis 0xCBF29CE484222325 */
#define FNV64basis0 0xCBF2
#define FNV64basis1 0x9CE4
#define FNV64basis2 0x8422
#define FNV64basis3 0x2325

/* FNV64 hash a null terminated string  (32 bit)
 ****************************************************************/
int FNV64string ( const char *in, uint8_t out[FNV64size] )
{
FNV64context     ctx;
int              err;

 if ( ( err = FNV64init (&ctx) ) != fnvSuccess )
    return err;
 if ( ( err = FNV64stringin (&ctx, in) ) != fnvSuccess )
    return err;
return FNV64result (&ctx, out);
}   /* end FNV64string */

/* FNV64 hash a counted block  (32 bit)
 ****************************************************************/
int FNV64block ( const void *in,
                 long int length,
                 uint8_t out[FNV64size] )
{
FNV64context     ctx;
int              err;

 if ( ( err = FNV64init (&ctx) ) != fnvSuccess )
    return err;
 if ( ( err = FNV64blockin (&ctx, in, length) ) != fnvSuccess )
    return err;
return FNV64result (&ctx, out);
}   /* end FNV64block */


/*****************************************************************
 *         Set of init, input, and output functions below        *
 *         to incrementally compute FNV64                        *
 *****************************************************************/

/* initialize context  (32 bit)
 ****************************************************************/
int FNV64init ( FNV64context * const ctx )
{
```

```
if ( ctx )
    {
    ctx->Hash[0] = FNV64basis0;
    ctx->Hash[1] = FNV64basis1;
    ctx->Hash[2] = FNV64basis2;
    ctx->Hash[3] = FNV64basis3;
    ctx->Computed = FNVinited+FNV64state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64init */

/* initialize context  (32 bit)
 ******************************************************************/
int FNV64initBasis ( FNV64context * const ctx,
                     const uint8_t basis[FNV64size] )
{
if ( ctx )
    {
#ifdef FNV_BigEndian
    ctx->Hash[0] = basis[1] + ( basis[0]<<8 );
    ctx->Hash[1] = basis[3] + ( basis[2]<<8 );
    ctx->Hash[2] = basis[5] + ( basis[4]<<8 );
    ctx->Hash[3] = basis[7] + ( basis[6]<<8 );
#else
    ctx->Hash[0] = basis[0] + ( basis[1]<<8 );
    ctx->Hash[1] = basis[2] + ( basis[3]<<8 );
    ctx->Hash[2] = basis[4] + ( basis[5]<<8 );
    ctx->Hash[3] = basis[6] + ( basis[7]<<8 );
#endif
    ctx->Computed = FNVinited+FNV64state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64initBasis */

/* hash in a counted block  (32 bit)
 ******************************************************************/
int FNV64blockin ( FNV64context * const ctx,
                   const void *vin,
                   long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t   temp[FNV64size/2];
uint32_t   temp2[2];
int        i;

if ( ctx && in )
    {
    if ( length < 0 )
```

```
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV64state:
            ctx->Computed = FNVcomputed+FNV64state;
        case FNVcomputed+FNV64state:
            break;
        default:
            return fnvStateError;
        }
    for ( i=0; i<FNV64size/2; ++i )
         temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV64prime * ( temp ^ *in++ ); */
        temp2[1] = temp[3] << FNV64shift;
        temp2[0] = temp[2] << FNV64shift;
        temp[3] = FNV64primeX * ( temp[3] ^ *in++ );
        temp[2] *= FNV64primeX;
        temp[1] = temp[1] * FNV64primeX + temp2[1];
        temp[0] = temp[0] * FNV64primeX + temp2[0];
        temp[2] += temp[3] >> 16;
        temp[3] &= 0xFFFF;
        temp[1] += temp[2] >> 16;
        temp[2] &= 0xFFFF;
        temp[0] += temp[1] >> 16;
        temp[1] &= 0xFFFF;
        }
    for ( i=0; i<FNV64size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64blockin */

/* hash in a string  (32 bit)
 ***************************************************************/
int FNV64stringin ( FNV64context * const ctx,
                    const char *in )
{
uint32_t   temp[FNV64size/2];
uint32_t   temp2[2];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV64state:
```

```
            ctx->Computed = FNVcomputed+FNV64state;
        case FNVcomputed+FNV64state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV64size/2; ++i )
         temp[i] = ctx->Hash[i];
    while ( ( ch = (uint8_t)*in++ ) != 0)
        {
        /* temp = FNV64prime * ( temp ^ ch ); */
        temp2[1] = temp[3] << FNV64shift;
        temp2[0] = temp[2] << FNV64shift;
        temp[3] = FNV64primeX * ( temp[3] ^ *in++ );
        temp[2] *= FNV64primeX;
        temp[1] = temp[1] * FNV64primeX + temp2[1];
        temp[0] = temp[0] * FNV64primeX + temp2[0];
        temp[2] += temp[3] >> 16;
        temp[3] &= 0xFFFF;
        temp[1] += temp[2] >> 16;
        temp[2] &= 0xFFFF;
        temp[0] += temp[1] >> 16;
        temp[1] &= 0xFFFF;
        }
    for ( i=0; i<FNV64size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64stringin */

/* return hash  (32 bit)
 ***************************************************************************/
int FNV64result ( FNV64context * const ctx,
                  uint8_t out[FNV64size] )
{
int    i;

if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV64state )
        return fnvStateError;
    for ( i=0; i<FNV64size/2; ++i )
        {
#ifdef FNV_BigEndian
        out[7-2*i] = ctx->Hash[i];
        out[6-2*i] = ctx->Hash[i] >> 8;
#else
        out[2*i] = ctx->Hash[i];
        out[2*i+1] = ctx->Hash[i] >> 8;
```

```
#endif
        ctx -> Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV64state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64result */

#endif    /*  FNV_64bitIntegers */
/*********************************************************************
 *        END VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC      *
 *********************************************************************/

#endif    /* _FNV64_C_ */
```

<CODE ENDS>

The header and C source for 64-bit FNV-1a returning a byte vector.

<CODE BEGINS>

```
/****************************** FNV64B.h *****************************/
/****************** See RFC NNNN for details. ***********************/
/*
 * Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

#ifndef _FNV64_H_
#define _FNV64_H_

/*
 *  Description:
 *      This file provides headers for the 64-bit version of the FNV-1a
 *      non-cryptographic hash algorithm.
 */

#include "FNVconfig.h"

#include <stdint.h>
#define FNV64size (64/8)

/* If you do not have the ISO standard stdint.h header file, then you
 * must typedef the following types:
 *
 *    type             meaning
 *  uint64_t        unsigned 64 bit integer (ifdef FNV_64bitIntegers)
 *  uint32_t        unsigned 32 bit integer
 *  uint16_t        unsigned 16 bit integer
 *  uint8_t         unsigned 8 bit integer (i.e., unsigned char)
```

```
 */

#ifndef _FNV_ErrCodes_
#define _FNV_ErrCodes_
/*********************************************************************
 *  All FNV functions provided return as integer as follows:
 *       0 -> success
 *      >0 -> error as listed below
 */
enum {    /* success and errors */
    fnvSuccess = 0,
    fnvNull,            /* Null pointer parameter */
    fnvStateError,      /* called Input after Result, etc. */
    fnvBadParam         /* passed a bad parameter */
};
#endif /* _FNV_ErrCodes_ */

/*
 *  This structure holds context information for an FNV64 hash
 */
#ifdef FNV_64bitIntegers
    /* version if 64 bit integers supported */

typedef struct FNV64Bcontext_s {
        int Computed;  /* state */
        uint64_t Hash;
} FNV64Bcontext;

#else
    /* version if 64 bit integers NOT supported */

typedef struct FNV64Bcontext_s {
        int Computed;  /* state */
        uint16_t Hash[FNV64size/2];
} FNV64Bcontext;

#endif /* FNV_64bitIntegers */

/*
 *  Function Prototypes
 *    FNV64Bstring: hash a zero terminated string not including
 *                  the terminating zero
 *    FNV64Bblock: FNV64 hash a specified length byte vector
 *    FNV64Binit: initializes an FNV64 context
 *    FNV64BinitBasis: initializes an FNV64 context with a
 *                     provided basis
 *    FNV64Bblockin: hash in a specified length byte vector
 *    FNV64Bstringin: hash in a zero terminated string not
 *                    incluing the zero
 *    FNV64Bresult: returns the hash value
```

```
 *
 *    Hash is returned as a vector of 8-bit integers
 */

#ifdef __cplusplus
extern "C" {
#endif

/* FNV64 */
extern int FNV64Bstring ( const char *in,
                          uint8_t out[FNV64size] );
extern int FNV64Bblock ( const void *in,
                         long int length,
                         uint8_t out[FNV64size] );
extern int FNV64Binit ( FNV64Bcontext * const );
extern int FNV64BinitBasis ( FNV64Bcontext * const,
                             const uint8_t basis[FNV64size] );
extern int FNV64Bblockin ( FNV64Bcontext * const,
                           const void * in,
                           long int length );
extern int FNV64Bstringin ( FNV64Bcontext * const,
                            const char * in );
extern int FNV64Bresult ( FNV64Bcontext * const,
                          uint8_t out[FNV64size] );

#ifdef __cplusplus
}
#endif

#endif /* _FNV64_H_ */
```

<CODE ENDS>

<CODE BEGINS>

```
/***************************** FNV64B.c *****************************/
/******************** See RFC NNNN for details **********************/
/* Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

/* This file implements the FNV (Fowler, Noll, Vo) non-cryptographic
 * hash function FNV-1a for 64-bit hashes.
 */

#ifndef _FNV64_C_
#define _FNV64_C_

#include "fnv-private.h"
#include "FNV64.h"
```

```
/*********************************************************************
 *        START VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC          *
 *********************************************************************/
#ifdef FNV_64bitIntegers

/* 64 bit FNV_prime = 2^40 + 2^8 + 0xb3 */
#define FNV64prime 0x00000100000001B3
#define FNV64basis 0xCBF29CE484222325

/* FNV64 hash a null terminated string  (64 bit)
 *********************************************************************/
int FNV64Bstring ( const char *in, uint64_t * const out )
{
uint64_t    temp;
uint8_t     ch;

if ( in && out )
    {
    temp = FNV64basis;
    while ( (ch = *in++) )
        temp = FNV64prime * ( temp ^ ch );
#ifdef FNV_BigEndian
    FNV64reverse ( out, temp );
#else
    *out = temp;
#endif
    return fnvSuccess;
    }
return fnvNull; /* Null input pointer */
}   /* end FNV64string */

/* FNV64 hash a counted block  (64 bit)
 *********************************************************************/
int FNV64Bblock ( const void *vin,
                  long int length,
                  uint64_t * const out )
{
const uint8_t *in = (const uint8_t*)vin;
uint64_t    temp;

if ( in && out )
    {
    if ( length < 0 )
        return fnvBadParam;
    for ( temp = FNV64basis; length > 0; length-- )
        temp = FNV64prime * ( temp ^ *in++ );
#ifdef FNV_BigEndian
    FNV64reverse ( out, temp );
#else
    *out = temp;
```

```
#endif
    return fnvSuccess;
    }
return fnvNull; /* Null input pointer */
}   /* end FNV64block */

#ifdef FNV_BigEndian

/* Store a Big Endian result back as Little Endian
 ************************************************************/
static void FNV64Breverse ( uint64_t *out, uint64_t hash )
{
uint64_t    temp;
int         i;

temp = hash & 0xFF;
for ( i = FNV64size - 1; i > 0; i-- )
    {
    hash >>= 8;
    temp = ( temp << 8 ) + ( hash & 0xFF );
    }
*out = temp;
}   /* end FNV64reverse */

#endif /* FNV_BigEndian */


/*********************************************************************
 *         Set of init, input, and output functions below            *
 *         to incrementally compute FNV64                            *
 *********************************************************************/

/* initialize context  (64 bit)
 ********************************************************************/
int FNV64Binit ( FNV64Bcontext * const ctx )
{
return FNV64initBasis ( ctx, FNV64basis );
}       /* end FNV64Binit */

/* initialize context with a provided basis  (64 bit)
 ************************************************************/
int FNV64BinitBasis ( FNV64Bcontext * const ctx, uint64_t basis )
{
if ( ctx )
    {
    ctx->Hash = basis;
    ctx->Computed = FNVinited+FNV64state;
    return fnvSuccess;
    }
return fnvNull;
```

```
}       /* end FNV64BinitBasis */

/* hash in a counted block  (64 bit)
 ********************************************************************/
int FNV64Bblockin ( FNV64Bcontext * const ctx,
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint64_t    temp;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV64state:
            ctx->Computed = FNVcomputed+FNV64state;
        case FNVcomputed+FNV64state:
            break;
        default:
            return fnvStateError;
        }
    for ( temp = ctx->Hash; length > 0; length-- )
        temp = FNV64prime * ( temp ^ *in++ );
    ctx->Hash = temp;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64Binput */

/* hash in a zero terminated string not including the zero (64 bit)
 ********************************************************************/
int FNV64Bstringin ( FNV64Bcontext * const ctx,
                     const char *in )
{
uint64_t        temp;
uint8_t         ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV64state:
            ctx->Computed = FNVcomputed+FNV64state;
        case FNVcomputed+FNV64state:
            break;
        default:
             return fnvStateError;
```

```
        }
    temp = ctx->Hash;
    while ( (ch = *in++) )
        temp = FNV64prime * ( temp ^ ch );
    ctx->Hash = temp;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64Bstringin */

/* return hash  (64 bit)
 ***********************************************************************/
int FNV64Bresult ( FNV64Bcontext * const ctx,
                   uint64_t * const out )
{
if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV64state )
        return fnvStateError;
    ctx->Computed = FNVemptied+FNV64state;
#ifdef FNV_BigEndian
    FNV64reverse ( out, ctx->Hash );
#else
    *out = ctx->Hash;
#endif
    ctx->Hash = 0;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64Bresult */

/*****************************************************************
 *        END VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC         *
 *****************************************************************/
#else    /*  FNV_64bitIntegers */
/*****************************************************************
 *     START VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC     *
 *****************************************************************/

/* 64 bit FNV_prime = 2^40 + 2^8 + 0xb3 */
/* #define FNV64prime 0x00000100000001B3 */
#define FNV64primeX 0x01B3
#define FNV64shift 8

/* #define FNV64basis 0xCBF29CE484222325 */
#define FNV64basis0 0xCBF2
#define FNV64basis1 0x9CE4
#define FNV64basis2 0x8422
#define FNV64basis3 0x2325
```

```
/* FNV64 hash a null terminated string  (32 bit)
 ***************************************************************/
int FNV64Bstring ( const char *in, uint8_t out[FNV64size] )
{
FNV64Bcontext     ctx;
int               err;

 if ( ( err = FNV64init (&ctx) ) != fnvSuccess )
    return err;
 if ( ( err = FNV64stringin (&ctx, in) ) != fnvSuccess )
    return err;
return FNV64result (&ctx, out);
}   /* end FNV64Bstring */

/* FNV64 hash a counted block  (32 bit)
 ***************************************************************/
int FNV64Bblock ( const void *in,
                  long int length,
                  uint8_t out[FNV64size] )
{
FNV64Bcontext     ctx;
int              err;

 if ( ( err = FNV64init (&ctx) ) != fnvSuccess )
    return err;
 if ( ( err = FNV64blockin (&ctx, in, length) ) != fnvSuccess )
    return err;
return FNV64result (&ctx, out);
}   /* end FNV64Bblock */


/*****************************************************************
 *         Set of init, input, and output functions below        *
 *         to incrementally compute FNV64                        *
 *****************************************************************/

/* initialize context  (32 bit)
 ***************************************************************/
int FNV64Binit ( FNV64Bcontext * const ctx )
{
if ( ctx )
    {
    ctx->Hash[0] = FNV64basis0;
    ctx->Hash[1] = FNV64basis1;
    ctx->Hash[2] = FNV64basis2;
    ctx->Hash[3] = FNV64basis3;
    ctx->Computed = FNVinited+FNV64state;
    return fnvSuccess;
    }
return fnvNull;
```

```
}   /* end FNV64Binit */

/* initialize context  (32 bit)
 ***************************************************************/
int FNV64BinitBasis ( FNV64Bcontext * const ctx,
                      const uint8_t basis[FNV64size] )
{
if ( ctx )
    {
#ifdef FNV_BigEndian
    ctx->Hash[0] = basis[1] + ( basis[0]<<8 );
    ctx->Hash[1] = basis[3] + ( basis[2]<<8 );
    ctx->Hash[2] = basis[5] + ( basis[4]<<8 );
    ctx->Hash[3] = basis[7] + ( basis[6]<<8 );
#else
    ctx->Hash[0] = basis[0] + ( basis[1]<<8 );
    ctx->Hash[1] = basis[2] + ( basis[3]<<8 );
    ctx->Hash[2] = basis[4] + ( basis[5]<<8 );
    ctx->Hash[3] = basis[6] + ( basis[7]<<8 );
#endif
    ctx->Computed = FNVinited+FNV64state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64BinitBasis */

/* hash in a counted block  (32 bit)
 ***************************************************************/
int FNV64Bblockin ( FNV64Bcontext * const ctx,
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t   temp[FNV64size/2];
uint32_t   temp2[2];
int        i;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV64state:
            ctx->Computed = FNVcomputed+FNV64state;
        case FNVcomputed+FNV64state:
            break;
        default:
            return fnvStateError;
        }
```

```
    for ( i=0; i<FNV64size/2; ++i )
         temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV64prime * ( temp ^ *in++ ); */
        temp2[1] = temp[3] << FNV64shift;
        temp2[0] = temp[2] << FNV64shift;
        temp[3] = FNV64primeX * ( temp[3] ^ *in++ );
        temp[2] *= FNV64primeX;
        temp[1] = temp[1] * FNV64primeX + temp2[1];
        temp[0] = temp[0] * FNV64primeX + temp2[0];
        temp[2] += temp[3] >> 16;
        temp[3] &= 0xFFFF;
        temp[1] += temp[2] >> 16;
        temp[2] &= 0xFFFF;
        temp[0] += temp[1] >> 16;
        temp[1] &= 0xFFFF;
        }
    for ( i=0; i<FNV64size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64Bblockin */

/* hash in a string  (32 bit)
 ***************************************************************/
int FNV64Bstringin ( FNV64Bcontext * const ctx,
                     const char *in )
{
uint32_t   temp[FNV64size/2];
uint32_t   temp2[2];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV64state:
            ctx->Computed = FNVcomputed+FNV64state;
        case FNVcomputed+FNV64state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV64size/2; ++i )
         temp[i] = ctx->Hash[i];
    while ( ( ch = (uint8_t)*in++ ) != 0)
        {
```

```
        /* temp = FNV64prime * ( temp ^ ch ); */
        temp2[1] = temp[3] << FNV64shift;
        temp2[0] = temp[2] << FNV64shift;
        temp[3] = FNV64primeX * ( temp[3] ^ *in++ );
        temp[2] *= FNV64primeX;
        temp[1] = temp[1] * FNV64primeX + temp2[1];
        temp[0] = temp[0] * FNV64primeX + temp2[0];
        temp[2] += temp[3] >> 16;
        temp[3] &= 0xFFFF;
        temp[1] += temp[2] >> 16;
        temp[2] &= 0xFFFF;
        temp[0] += temp[1] >> 16;
        temp[1] &= 0xFFFF;
        }
    for ( i=0; i<FNV64size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64Bstringin */

/* return hash  (32 bit)
 ***********************************************************************/
int FNV64Bresult ( FNV64Bcontext * const ctx,
                   uint8_t out[FNV64size] )
{
int    i;

if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV64state )
        return fnvStateError;
    for ( i=0; i<FNV64size/2; ++i )
        {
#ifdef FNV_BigEndian
        out[7-2*i] = ctx->Hash[i];
        out[6-2*i] = ctx->Hash[i] >> 8;
#else
        out[2*i] = ctx->Hash[i];
        out[2*i+1] = ctx->Hash[i] >> 8;
#endif
        ctx -> Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV64state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV64Bresult */

#endif    /*  FNV_64bitIntegers */
```

```
/*********************************************************************
 *        END VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC       *
 *********************************************************************/

#endif    /* _FNV64_C_ */
```

<CODE ENDS>

6.1.3 FNV128 C Code

The header and C source for 128-bit FNV-1a returning a byte vector.

<CODE BEGINS>

```
/***************************** FNV128.h *************************/
/***************** See RFC NNNN for details. ********************/
/*
 * Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

#ifndef _FNV128_H_
#define _FNV128_H_

/*
 *  Description:
 *      This file provides headers for the 128-bit version of the
 *      FNV-1a non-cryptographic hash algorithm.
 */

#include "FNVconfig.h"

#include <stdint.h>
#define FNV128size (128/8)

/* If you do not have the ISO standard stdint.h header file, then you
 * must typedef the following types:
 *
 *    type              meaning
 *  uint64_t         unsigned 64 bit integer (ifdef FNV_64bitIntegers)
 *  uint32_t         unsigned 32 bit integer
 *  uint16_t         unsigned 16 bit integer
 *  uint8_t          unsigned 8 bit integer (i.e., unsigned char)
 */

#ifndef _FNV_ErrCodes_
#define _FNV_ErrCodes_
/*********************************************************************
 *  All FNV functions provided return as integer as follows:
```

```
 *       0 -> success
 *      >0 -> error as listed below
 */
enum {    /* success and errors */
    fnvSuccess = 0,
    fnvNull,            /* Null pointer parameter */
    fnvStateError,      /* called Input after Result or before Init */
    fnvBadParam         /* passed a bad parameter */
};
#endif /* _FNV_ErrCodes_ */

/*
 *  This structure holds context information for an FNV128 hash
 */
#ifdef FNV_64bitIntegers
    /* version if 64 bit integers supported */
typedef struct FNV128context_s {
        int Computed;  /* state */
        uint32_t Hash[FNV128size/4];
} FNV128context;

#else
    /* version if 64 bit integers NOT supported */

typedef struct FNV128context_s {
        int Computed;  /* state */
        uint16_t Hash[FNV128size/2];
} FNV128context;

#endif /* FNV_64bitIntegers */

/*
 *  Function Prototypes
 *    FNV128string: hash a zero terminated string not including
 *                  the terminating zero
 *    FNV128block: FNV128 hash a specified length byte vector
 *    FNV128init: initializes an FNV128 context
 *    FNV128initBasis: initializes an FNV128 context with a
 *                     provided basis
 *    FNV128blockin: hash in a specified length byte vector
 *    FNV128stringin: hash in a zero terminated string not
 *                    including the zero
 *    FNV128result: returns the hash value
 *
 *    Hash is returned as an array of 8-bit integers
 */

#ifdef __cplusplus
extern "C" {
#endif
```

```
/* FNV128 */
extern int FNV128string ( const char *in,
                          uint8_t out[FNV128size] );
extern int FNV128block ( const void *in,
                         long int length,
                         uint8_t out[FNV128size] );
extern int FNV128init ( FNV128context * const );
extern int FNV128initBasis ( FNV128context * const,
                             const uint8_t basis[FNV128size] );
extern int FNV128blockin ( FNV128context * const,
                           const void *in,
                           long int length );
extern int FNV128stringin ( FNV128context * const,
                            const char *in );
extern int FNV128result ( FNV128context * const,
                          uint8_t out[FNV128size] );

#ifdef __cplusplus
}
#endif

#endif /* _FNV128_H_ */
```

<CODE ENDS>

<CODE BEGINS>

```
/***************************** FNV128.c ****************************/
/******************** See RFC NNNN for details *********************/
/* Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights
 * See fnv-private.h for terms of use and redistribution.
 */

/* This file implements the FNV (Fowler, Noll, Vo) non-cryptographic
 * hash function FNV-1a for 128-bit hashes.
 */

#ifndef _FNV128_C_
#define _FNV128_C_

#include "fnv-private.h"
#include "FNV128.h"

/* common code for 64 and 32 bit modes */

/* FNV128 hash a null terminated string  (64/32 bit)
 ***************************************************************/
int FNV128string ( const char *in, uint8_t out[FNV128size] )
{
FNV128context    ctx;
int              err;
```

```
err = FNV128init ( &ctx );
if ( err != fnvSuccess ) return err;
err = FNV128stringin ( &ctx, in );
if ( err != fnvSuccess ) return err;
return FNV128result (&ctx, out);
}   /* end FNV128string */

/* FNV128 hash a counted block  (64/32 bit)
 ********************************************************************/
int FNV128block ( const void *in,
                  long int length,
                  uint8_t out[FNV128size] )
{
FNV128context    ctx;
int              err;

err = FNV128init ( &ctx );
if ( err != fnvSuccess ) return err;
err = FNV128blockin ( &ctx, in, length );
if ( err != fnvSuccess ) return err;
return FNV128result ( &ctx, out );
}   /* end FNV128block */


/*********************************************************************
 *        START VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC          *
 *********************************************************************/
#ifdef FNV_64bitIntegers

/* 128 bit FNV_prime = 2^88 + 2^8 + 0x3b */
/* 0x00000000 01000000 00000000 0000013B */
#define FNV128primeX 0x013B
#define FNV128shift 24

/* 0x6C62272E 07BB0142 62B82175 6295C58D */
#define FNV128basis0 0x6C62272E
#define FNV128basis1 0x07BB0142
#define FNV128basis2 0x62B82175
#define FNV128basis3 0x6295C58D

/*********************************************************************
 *         Set of init, input, and output functions below            *
 *         to incrementally compute FNV128                           *
 *********************************************************************/

/* initialize context  (64 bit)
 ********************************************************************/
int FNV128init ( FNV128context * const ctx )
{
if ( ctx )
```

```
    {
    ctx->Hash[0] = FNV128basis0;
    ctx->Hash[1] = FNV128basis1;
    ctx->Hash[2] = FNV128basis2;
    ctx->Hash[3] = FNV128basis3;
    ctx->Computed = FNVinited+FNV128state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128init */

/* initialize context with a provided basis  (64 bit)
 ****************************************************************/
int FNV128initBasis ( FNV128context * const ctx,
                      const uint8_t basis[FNV128size] )
{
int       i;
const uint8_t   *ui8p;
uint32_t  temp;

if ( ctx )
    {
#ifdef FNV_BigEndian
    ui8p = basis;
    for ( i=0; i < FNV128size/4; ++i )
        {
        temp = (*ui8p++)<<8;
        temp = (temp + *ui8p++)<<8;
            temp = (temp + *ui8p++)<<8;
        ctx->Hash[i] = temp +  *ui8p;
        }
#else
    ui8p = basis + ( FNV128size/4 - 1 );
    for ( i=0; i < FNV128size/4; ++i )
        {
            temp = (*ui8p--)<<8;
            temp = (temp + *ui8p--)<<8;
            temp = (temp + *ui8p--)<<8;
        ctx->Hash[i] = temp +  *ui8p;
        }
#endif
    ctx->Computed = FNVinited+FNV128state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128initBasis */

/* hash in a counted block  (64 bit)
 ****************************************************************/
int FNV128blockin ( FNV128context * const ctx,
```

```
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint64_t    temp[FNV128size/4];
uint64_t    temp2[2];
int         i;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV128state:
            ctx->Computed = FNVcomputed+FNV128state;
        case FNVcomputed+FNV128state:
            break;
        default:
            return fnvStateError;
        }
    for ( i=0; i<FNV128size/4; ++i )
         temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV128prime * ( temp ^ *in++ ); */
        temp2[1] = temp[3] << FNV128shift;
        temp2[0] = temp[2] << FNV128shift;
        temp[3] = FNV128primeX * ( temp[3] ^ *in++ );
        temp[2] *= FNV128primeX;
        temp[1] = temp[1] * FNV128primeX + temp2[1];
        temp[0] = temp[0] * FNV128primeX + temp2[0];
        temp[2] += temp[3] >> 32;
        temp[3] &= 0xFFFFFFFF;
        temp[1] += temp[2] >> 32;
        temp[2] &= 0xFFFFFFFF;
        temp[0] += temp[1] >> 32;
        temp[1] &= 0xFFFFFFFF;
        }
    for ( i=0; i<FNV128size/4; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128input */

/* hash in a string  (64 bit)
 ****************************************************************/
int FNV128stringin ( FNV128context * const ctx, const char *in )
{
```

```
uint64_t   temp[FNV128size/4];
uint64_t   temp2[2];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV128state:
            ctx->Computed = FNVcomputed+FNV128state;
        case FNVcomputed+FNV128state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV128size/4; ++i )
         temp[i] = ctx->Hash[i];
    while ( (ch = (uint8_t)*in++) )
        {
        /* temp = FNV128prime * ( temp ^ ch ); */
        temp2[1] = temp[3] << FNV128shift;
        temp2[0] = temp[2] << FNV128shift;
        temp[3] = FNV128primeX * ( temp[3] ^ *in++ );
        temp[2] *= FNV128primeX;
        temp[1] = temp[1] * FNV128primeX + temp2[1];
        temp[0] = temp[0] * FNV128primeX + temp2[0];
        temp[2] += temp[3] >> 32;
        temp[3] &= 0xFFFFFFFF;
        temp[1] += temp[2] >> 32;
        temp[2] &= 0xFFFFFFFF;
        temp[0] += temp[1] >> 32;
        temp[1] &= 0xFFFFFFFF;
        }
    for ( i=0; i<FNV128size/4; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128stringin */

/* return hash  (64 bit)
 ***********************************************************************/
int FNV128result ( FNV128context * const ctx, uint8_t out[FNV128size] )
{
int     i;

if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV128state )
```

```
        return fnvStateError;
    for ( i=0; i<FNV128size/4; ++i )
        {
#ifdef FNV_BigEndian
        out[15-4*i] = ctx->Hash[i];
        out[14-4*i] = ctx->Hash[i] >> 8;
        out[13-4*i] = ctx->Hash[i] >> 16;
        out[12-4*i] = ctx->Hash[i] >> 24;
#else
        out[4*i] = ctx->Hash[i];
        out[4*i+1] = ctx->Hash[i] >> 8;
        out[4*i+2] = ctx->Hash[i] >> 16;
        out[4*i+3] = ctx->Hash[i] >> 24;
#endif
        ctx -> Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV128state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128result */

/*****************************************************************
 *        END VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC        *
 *****************************************************************/
#else    /*  FNV_64bitIntegers */
/*****************************************************************
 *     START VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC    *
 *****************************************************************/

/* 128 bit FNV_prime = 2^88 + 2^8 + 0x3b */
/* 0x00000000 01000000 00000000 0000013B */
#define FNV128primeX 0x013B
#define FNV128shift 8

/* 0x6C62272E 07BB0142 62B82175 6295C58D */
uint16_t FNV128basis[FNV128size/2] =
         { 0x6C62, 0x272E, 0x07BB, 0x0142,
           0x62B8, 0x2175, 0x6295, 0xC58D };

/*****************************************************************
 *         Set of init, input, and output functions below        *
 *         to incrementally compute FNV128                       *
 *****************************************************************/

/* initialize context  (32 bit)
 *****************************************************************/
int FNV128init ( FNV128context *ctx )
{
int     i;
```

```
if ( ctx )
    {
    for ( i=0; i<FNV128size/2; ++i )
        ctx->Hash[i] = FNV128basis[i];
    ctx->Computed = FNVinited+FNV128state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128init */

/* initialize context with a provided basis  (32 bit)
 ***********************************************************************/
int FNV128initBasis ( FNV128context *ctx,
                      const uint8_t basis[FNV128size] )
{
int      i;
const uint8_t  *ui8p;
uint32_t temp;

if ( ctx )
    {
#ifdef FNV_BigEndian
    ui8p = basis;
    for ( i=0; i < FNV128size/2; ++i )
        {
        temp = *ui8p++;
        ctx->Hash[i] = ( temp<<8 ) + (*ui8p++);
        }
#else
    ui8p = basis + (FNV128size/2 - 1);
    for ( i=0; i < FNV128size/2; ++i )
        {
        temp = *ui8p--;
        ctx->Hash[i] = ( temp<<8 ) +  (*ui8p--);
        }
#endif
    ctx->Computed = FNVinited+FNV128state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128initBasis */

/* hash in a counted block  (32 bit)
 ***********************************************************************/
int FNV128blockin ( FNV128context *ctx,
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t   temp[FNV128size/2];
```

```
uint32_t   temp2[3];
int        i;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV128state:
            ctx->Computed = FNVcomputed+FNV128state;
        case FNVcomputed+FNV128state:
            break;
        default:
            return fnvStateError;
        }
    for ( i=0; i<FNV128size/2; ++i )
         temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV128prime * ( temp ^ *in++ ); */
        temp[7] ^= *in++;
        temp2[2] = temp[7] << FNV128shift;
        temp2[1] = temp[6] << FNV128shift;
        temp2[0] = temp[5] << FNV128shift;
        for ( i=0; i<8; ++i )
            temp[i] *= FNV128primeX;
        temp[2] += temp2[2];
        temp[1] += temp2[1];
        temp[0] += temp2[0];
        for ( i=7; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
        }
    for ( i=0; i<FNV128size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128blockin */

/* hash in a string  (32 bit)
 ***************************************************************/
int FNV128stringin ( FNV128context *ctx,
                     const char *in )
{
uint32_t   temp[FNV128size/2];
uint32_t   temp2[3];
```

```
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV128state:
            ctx->Computed = FNVcomputed+FNV128state;
        case FNVcomputed+FNV128state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV128size/2; ++i )
         temp[i] = ctx->Hash[i];
    while ( (ch = (uint8_t)*in++) )
        {
        /* temp = FNV128prime * ( temp ^ *in++ ); */
        temp[7] ^= ch;
        temp2[2] = temp[7] << FNV128shift;
        temp2[1] = temp[6] << FNV128shift;
        temp2[0] = temp[5] << FNV128shift;
        for ( i=0; i<8; ++i )
            temp[i] *= FNV128primeX;
        temp[2] += temp2[2];
        temp[1] += temp2[1];
        temp[0] += temp2[0];
        for ( i=7; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
         }
    for ( i=0; i<FNV128size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128stringin */

/* return hash  (32 bit)
 ***************************************************************/
int FNV128result ( FNV128context *ctx,
                   uint8_t out[FNV128size] )
{
int    i;

if ( ctx && out )
    {
```

```
    if ( ctx->Computed != FNVcomputed+FNV128state )
        return fnvStateError;
    for ( i=0; i<FNV128size/2; ++i )
        {
#ifdef FNV_BigEndian
        out[15-2*i] = ctx->Hash[i];
        out[14-2*i] = ctx->Hash[i] >> 8;
#else
        out[2*i] = ctx->Hash[i];
        out[2*i+1] = ctx->Hash[i] >> 8;
#endif
        ctx -> Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV128state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV128result */

#endif    /*  Have64bitIntegers */
/*****************************************************************
 *        END VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC      *
 *****************************************************************/

#endif    /* _FNV128_C_ */
```

<CODE ENDS>

6.1.4 FNV256 C Code

The header and C source for 256-bit FNV-1a returning a byte vector.

<CODE BEGINS>

```
/*************************** FNV256.h ************************/
/***************** See RFC NNNN for details. ******************/
/*
 * Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

#ifndef _FNV256_H_
#define _FNV256_H_

/*
 *  Description:
 *      This file provides headers for the 256-bit version of the
 *      FNV-1a non-cryptographic hash algorithm.
 */
```

```
#include "FNVconfig.h"

#include <stdint.h>
#define FNV256size (256/8)

/* If you do not have the ISO standard stdint.h header file, then you
 * must typedef the following types:
 *
 *    type              meaning
 *  uint64_t         unsigned 64 bit integer (ifdef FNV_64bitIntegers)
 *  uint32_t         unsigned 32 bit integer
 *  uint16_t         unsigned 16 bit integer
 *  uint8_t          unsigned 8 bit integer (i.e., unsigned char)
 */

#ifndef _FNV_ErrCodes_
#define _FNV_ErrCodes_
/*********************************************************************
 *  All FNV functions provided return as integer as follows:
 *       0 -> success
 *      >0 -> error as listed below
 */
enum {    /* success and errors */
    fnvSuccess = 0,
    fnvNull,            /* Null pointer parameter */
    fnvStateError,      /* called Input after Result or before Init */
    fnvBadParam         /* passed a bad parameter */
};
#endif /* _FNV_ErrCodes_ */

/*
 *  This structure holds context information for an FNV256 hash
 */
#ifdef FNV_64bitIntegers
    /* version if 64 bit integers supported */
typedef struct FNV256context_s {
        int Computed;  /* state */
        uint32_t Hash[FNV256size/4];
} FNV256context;

#else
    /* version if 64 bit integers NOT supported */

typedef struct FNV256context_s {
        int Computed;  /* state */
        uint16_t Hash[FNV256size/2];
} FNV256context;

#endif /* FNV_64bitIntegers */
```

```
/*
 *  Function Prototypes
 *    FNV256string: hash a zero terminated string not including
 *                  the zero
 *    FNV256block: FNV256 hash a specified length byte vector
 *    FNV256init: initializes an FNV256 context
 *    FNV256initBasis:  initializes an FNV256 context with a provided
 *                      basis
 *    FNV256blockin: hash in a specified length byte vector
 *    FNV256stringin: hash in a zero terminated string not
 *                    including the zero
 *    FNV256result: returns the hash value
 *
 *    Hash is returned as an array of 8-bit integers
 */

#ifdef __cplusplus
extern "C" {
#endif

/* FNV256 */
extern int FNV256string ( const char *in,
                          uint8_t out[FNV256size] );
extern int FNV256block ( const void *in,
                         long int length,
                         uint8_t out[FNV256size] );
extern int FNV256init ( FNV256context *);
extern int FNV256initBasis ( FNV256context * const,
                             const uint8_t basis[FNV256size] );
extern int FNV256blockin ( FNV256context * const,
                           const void *in,
                           long int length );
extern int FNV256stringin ( FNV256context * const,
                            const char *in );
extern int FNV256result ( FNV256context * const,
                          uint8_t out[FNV256size] );

#ifdef __cplusplus
}
#endif

#endif /* _FNV256_H_ */
   <CODE ENDS>

   <CODE BEGINS>
/***************************** FNV256.c ****************************/
/******************** See RFC NNNN for details *********************/
/* Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights
 * See fnv-private.h for terms of use and redistribution.
```

```
 */

/* This file implements the FNV (Fowler, Noll, Vo) non-cryptographic
 * hash function FNV-1a for 256-bit hashes.
 */

#ifndef _FNV256_C_
#define _FNV256_C_

#include "fnv-private.h"
#include "FNV256.h"

/* common code for 64 and 32 bit modes */

/* FNV256 hash a null terminated string  (64/32 bit)
 ***************************************************************/
int FNV256string ( const char *in, uint8_t out[FNV256size] )
{
FNV256context    ctx;
int              err;

if ( (err = FNV256init ( &ctx )) != fnvSuccess )
    return err;
if ( (err = FNV256stringin ( &ctx, in )) != fnvSuccess )
    return err;
return FNV256result ( &ctx, out );
}   /* end FNV256string */

/* FNV256 hash a counted block  (64/32 bit)
 ***************************************************************/
int FNV256block ( const void *in,
                  long int length,
                  uint8_t out[FNV256size] )
{
FNV256context    ctx;
int              err;

if ( (err = FNV256init ( &ctx )) != fnvSuccess )
    return err;
if ( (err = FNV256blockin ( &ctx, in, length)) != fnvSuccess )
    return err;
return FNV256result ( &ctx, out );
}   /* end FNV256block */


/***************************************************************
 *        START VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC         *
 ***************************************************************/
#ifdef FNV_64bitIntegers
```

```
/* 256 bit FNV_prime = 2^168 + 2^8 + 0x63 */
/* 0x0000000000000000 0000010000000000
     0000000000000000 0000000000000163 */
#define FNV256primeX 0x0163
#define FNV256shift 8

/* 0xDD268DBCAAC55036 2D98C384C4E576CC
     C8B1536847B6BBB3 1023B4C8CAEE0535 */
uint32_t FNV256basis[FNV256size/4] = {
         0xDD268DBC, 0xAAC55036, 0x2D98C384, 0xC4E576CC,
         0xC8B15368, 0x47B6BBB3, 0x1023B4C8, 0xCAEE0535 };

/*****************************************************************
 *         Set of init, input, and output functions below        *
 *         to incrementally compute FNV256                       *
 *****************************************************************/

/* initialize context  (64 bit)
 *****************************************************************/
int FNV256init ( FNV256context *ctx )
{
int      i;

if ( ctx )
    {
    for ( i=0; i<FNV256size/4; ++i )
        ctx->Hash[i] = FNV256basis[i];
    ctx->Computed = FNVinited+FNV256state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256init */

/* initialize context with a provided basis  (64 bit)
 *****************************************************************/
int FNV256initBasis ( FNV256context* const ctx,
                      const uint8_t basis[FNV256size] )
{
int      i;
const uint8_t  *ui8p;
uint32_t  temp;

if ( ctx )
    {
#ifdef FNV_BigEndian
    ui8p = basis;
    for ( i=0; i < FNV256size/4; ++i )
        {
            temp = (*ui8p++)<<8;
            temp = (temp + *ui8p++)<<8;
```

```
            temp = (temp + *ui8p++)<<8;
        ctx->Hash[i] = temp +  *ui8p;
        }
#else
    ui8p = basis + (FNV256size/4 - 1);
    for ( i=0; i < FNV256size/4; ++i )
        {
            temp = (*ui8p--)<<8;
            temp = (temp + *ui8p--)<<8;
            temp = (temp + *ui8p--)<<8;
        ctx->Hash[i] = temp +  *ui8p;
        }
#endif
    ctx->Computed = FNVinited+FNV256state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256initBasis */

/* hash in a counted block  (64 bit)
 ******************************************************************/
int FNV256blockin ( FNV256context *ctx,
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint64_t    temp[FNV256size/4];
uint64_t    temp2[3];
int         i;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV256state:
            ctx->Computed = FNVcomputed+FNV256state;
        case FNVcomputed+FNV256state:
            break;
        default:
            return fnvStateError;
        }
    for ( i=0; i<FNV256size/4; ++i )
         temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV256prime * ( temp ^ *in++ ); */
        temp[7] ^= *in++;
        temp2[2] = temp[7] << FNV256shift;
```

```
        temp2[1] = temp[6] << FNV256shift;
        temp2[0] = temp[5] << FNV256shift;
        for ( i=0; i<FNV256size/4; ++i )
            temp[i] *= FNV256primeX;
        temp[2] += temp2[2];
        temp[1] += temp2[1];
        temp[0] += temp2[0];
        for ( i=FNV256size/4-1; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
       }
    for ( i=0; i<FNV256size/4; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256input */

/* hash in a string  (64 bit)
 ****************************************************************/
int FNV256stringin ( FNV256context *ctx, const char *in )
{
uint64_t   temp[FNV256size/4];
uint64_t   temp2[3];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV256state:
            ctx->Computed = FNVcomputed+FNV256state;
        case FNVcomputed+FNV256state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV256size/4; ++i )
         temp[i] = ctx->Hash[i];
    while ( (ch = (uint8_t)*in++) )
        {
        /* temp = FNV256prime * ( temp ^ ch ); */
        temp[7] ^= ch;
        temp2[2] = temp[7] << FNV256shift;
        temp2[1] = temp[6] << FNV256shift;
        temp2[0] = temp[5] << FNV256shift;
        for ( i=0; i<FNV256size/4; ++i )
```

```
            temp[i] *= FNV256primeX;
        temp[2] += temp2[2];
        temp[1] += temp2[1];
        temp[0] += temp2[0];
        for ( i=FNV256size/4-1; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
       }
    for ( i=0; i<FNV256size/4; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256stringin */

/* return hash  (64 bit)
 ***********************************************************************/
int FNV256result ( FNV256context *ctx, uint8_t out[FNV256size] )
{
int     i;

if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV256state )
        return fnvStateError;
    for ( i=0; i<FNV256size/4; ++i )
        {
#ifdef FNV_BigEndian
        out[31-4*i] = ctx->Hash[i];
        out[31-4*i] = ctx->Hash[i] >> 8;
        out[31-4*i] = ctx->Hash[i] >> 16;
        out[31-4*i] = ctx->Hash[i] >> 24;
#else
        out[4*i] = ctx->Hash[i];
        out[4*i+1] = ctx->Hash[i] >> 8;
        out[4*i+2] = ctx->Hash[i] >> 16;
        out[4*i+3] = ctx->Hash[i] >> 24;
#endif
        ctx -> Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV256state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256result */

/***************************************************************
 *        END VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC         *
```

```
 ****************************************************************/
#else    /*  FNV_64bitIntegers */
/****************************************************************
 *      START VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC    *
 ****************************************************************/

/* version for when you only have 32-bit arithmetic
 ****************************************************************/

/* 256 bit FNV_prime = 2^168 + 2^8 + 0x63 */
/* 0x00000000 00000000 00000100 00000000
     00000000 00000000 00000000 00000163 */
#define FNV256primeX 0x0163
#define FNV256shift 8

/* 0xDD268DBCAAC55036 2D98C384C4E576CC
     C8B1536847B6BBB3 1023B4C8CAEE0535 */
uint16_t FNV256basis[FNV256size/2] = {
         0xDD26, 0x8DBC, 0xAAC5, 0x5036,
         0x2D98, 0xC384, 0xC4E5, 0x76CC,
         0xC8B1, 0x5368, 0x47B6, 0xBBB3,
         0x1023, 0xB4C8, 0xCAEE, 0x0535 };

/****************************************************************
 *         Set of init, input, and output functions below        *
 *         to incrementally compute FNV256                       *
 ****************************************************************/

/* initialize context  (32 bit)
 ****************************************************************/
int FNV256init ( FNV256context *ctx )
{
int     i;

if ( ctx )
    {
    for ( i=0; i<FNV256size/2; ++i )
        ctx->Hash[i] = FNV256basis[i];
    ctx->Computed = FNVinited+FNV256state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256init */

/* initialize context with a provided basis  (32 bit)
 ****************************************************************/
int FNV256initBasis ( FNV256context *ctx,
                      const uint8_t basis[FNV256size] )
{
int      i;
```

```
const uint8_t  *ui8p;
uint32_t temp;

if ( ctx )
    {
#ifdef FNV_BigEndian
    ui8p = basis;
    for ( i=0; i < FNV256size/2; ++i )
        {
        temp = *ui8p++;
        ctx->Hash[i] = ( temp<<8 ) + (*ui8p++);
        }
#else
    ui8p = basis + FNV256size/2 -1;
    for ( i=0; i < FNV256size/2; ++i )
        {
        temp = *ui8p--;
        ctx->Hash[i] = ( temp<<8 ) +  (*ui8p--);
        }
#endif
    ctx->Computed = FNVinited+FNV256state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256initBasis */

/* hash in a counted block  (32 bit)
 ****************************************************************/
int FNV256blockin ( FNV256context *ctx,
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t   temp[FNV256size/2];
uint32_t   temp2[6];
int        i;

if ( ctx && in )
    {
    if ( length < 0 )
        return fnvBadParam;
    switch ( ctx->Computed )
        {
        case FNVinited+FNV256state:
            ctx->Computed = FNVcomputed+FNV256state;
        case FNVcomputed+FNV256state:
            break;
        default:
            return fnvStateError;
        }
```

```
    for ( i=0; i<FNV256size/2; ++i )
        temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV256prime * ( temp ^ *in++ ); */
        temp[15] ^= *in++;
        for ( i=0; i<6; ++i )
            temp2[i] = temp[10+i] << FNV256shift;
        for ( i=0; i<FNV256size/2; ++i )
            temp[i] *= FNV256primeX;
        for ( i=0; i<6; ++i )
            temp[10+i] += temp2[i];
        for ( i=15; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
        }
    for ( i=0; i<FNV256size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256blockin */

/* hash in a string  (32 bit)
 ****************************************************************/
int FNV256stringin ( FNV256context *ctx, const char *in )
{
uint32_t   temp[FNV256size/2];
uint32_t   temp2[6];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV256state:
            ctx->Computed = FNVcomputed+FNV256state;
        case FNVcomputed+FNV256state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV256size/2; ++i )
         temp[i] = ctx->Hash[i];
    while ( ( ch = (uint8_t)*in++ ) != 0)
        {
        /* temp = FNV256prime * ( temp ^ *in++ ); */
```

```
        temp[15] ^= ch;
        for ( i=0; i<6; ++i )
            temp2[i] = temp[10+i] << FNV256shift;
        for ( i=0; i<FNV256size/2; ++i )
            temp[i] *= FNV256primeX;
        for ( i=0; i<6; ++i )
            temp[10+i] += temp2[i];
        for ( i=15; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
         }
    for ( i=0; i<FNV256size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256stringin */

/* return hash  (32 bit)
 ********************************************************************/
int FNV256result ( FNV256context *ctx, uint8_t out[FNV256size] )
{
int    i;

if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV256state )
        return fnvStateError;
    for ( i=0; i<FNV256size/2; ++i )
        {
#ifdef FNV_BigEndian
        out[31-2*i] = ctx->Hash[i];
        out[30-2*i] = ctx->Hash[i] >> 8;
#else
        out[2*i] = ctx->Hash[i];
        out[2*i+1] = ctx->Hash[i] >> 8;
#endif
        ctx->Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV256state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV256result */

#endif    /*  Have64bitIntegers */
/*********************************************************************
 *        END VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC      *
```

```
 ****************************************************************/

#endif    /* _FNV256_C_ */
   <CODE ENDS>
```

6.1.5 FNV512 C Code

The header and C source for 512-bit FNV-1a returning a byte vector.

```
   <CODE BEGINS>
/***************************** FNV512.h *************************/
/***************** See RFC NNNN for details. ********************/
/*
 * Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

#ifndef _FNV512_H_
#define _FNV512_H_

/*
 *  Description:
 *      This file provides headers for the 512-bit version of the
 *      FNV-1a non-cryptographic hash algorithm.
 */

#include "FNVconfig.h"

#include <stdint.h>
#define FNV512size (512/8)

/* If you do not have the ISO standard stdint.h header file, then you
 * must typedef the following types:
 *
 *    type              meaning
 *  uint64_t         unsigned 64 bit integer (ifdef FNV_64bitIntegers)
 *  uint32_t         unsigned 32 bit integer
 *  uint16_t         unsigned 16 bit integer
 *  uint8_t          unsigned 8 bit integer (i.e., unsigned char)
 */

#ifndef _FNV_ErrCodes_
#define _FNV_ErrCodes_
/*****************************************************************
 *  All FNV functions provided return as integer as follows:
 *       0 -> success
 *      >0 -> error as listed below
```

```
 */
enum {    /* success and errors */
    fnvSuccess = 0,
    fnvNull,            /* Null pointer parameter */
    fnvStateError,      /* called Input after Result or before Init */
    fnvBadParam         /* passed a bad parameter */
};
#endif /* _FNV_ErrCodes_ */

/*
 *  This structure holds context information for an FNV512 hash
 */
#ifdef FNV_64bitIntegers
    /* version if 64 bit integers supported */
typedef struct FNV512context_s {
        int Computed;  /* state */
        uint32_t Hash[FNV512size/4];
} FNV512context;

#else
    /* version if 64 bit integers NOT supported */

typedef struct FNV512context_s {
        int Computed;  /* state */
        uint16_t Hash[FNV512size/2];
} FNV512context;

#endif /* FNV_64bitIntegers */

/*
 *  Function Prototypes
 *    FNV512string: hash a zero terminated string not including
 *                  the terminating zero
 *    FNV512block: FNV512 hash a specified length byte vector
 *    FNV512init: initializes an FNV512 context
 *    FNV512initBasis: initializes an FNV512 context with a
 *                     provided basis
 *    FNV512blockin: hash in a specified length byte vector
 *    FNV512stringin: hash in a zero terminated string not
 *                    including the zero
 *    FNV512result: returns the hash value
 *
 *    Hash is returned as an array of 8-bit integers
 */

#ifdef __cplusplus
extern "C" {
#endif

/* FNV512 */
```

```
extern int FNV512string ( const char *in,
                          uint8_t out[FNV512size] );
extern int FNV512block ( const void *in,
                         long int length,
                         uint8_t out[FNV512size] );
extern int FNV512init ( FNV512context *);
extern int FNV512initBasis ( FNV512context * const,
                             const uint8_t basis[FNV512size] );
extern int FNV512blockin ( FNV512context *,
                           const void *in,
                           long int length );
extern int FNV512stringin ( FNV512context *,
                            const char *in );
extern int FNV512result ( FNV512context *,
                          uint8_t out[FNV512size] );

#ifdef __cplusplus
}
#endif

#endif /* _FNV512_H_ */
```

<CODE ENDS>

<CODE BEGINS>

```
/***************************** FNV512.c ****************************/
/******************** See RFC NNNN for details *********************/
/* Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights
 * See fnv-private.h for terms of use and redistribution.
 */

/* This file implements the FNV (Fowler, Noll, Vo) non-cryptographic
 * hash function FNV-1a for 512-bit hashes.
 */

#ifndef _FNV512_C_
#define _FNV512_C_

#include "fnv-private.h"
#include "FNV512.h"

/* common code for 64 and 32 bit modes */

/* FNV512 hash a null terminated string  (64/32 bit)
 ***************************************************************/
int FNV512string ( const char *in, uint8_t out[FNV512size] )
{
FNV512context    ctx;
int              err;
```

```
if ( (err = FNV512init ( &ctx )) != fnvSuccess )
    return err;
if ( (err = FNV512stringin ( &ctx, in )) != fnvSuccess )
    return err;
return FNV512result ( &ctx, out );
}   /* end FNV512string */

/* FNV512 hash a counted block  (64/32 bit)
 ****************************************************************/
int FNV512block ( const void *in,
                  long int length,
                  uint8_t out[FNV512size] )
{
FNV512context    ctx;
int              err;

if ( (err = FNV512init ( &ctx )) != fnvSuccess )
    return err;
if ( (err = FNV512blockin ( &ctx, in, length)) != fnvSuccess )
    return err;
return FNV512result ( &ctx, out );
}   /* end FNV512block */


/*****************************************************************
 *        START VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC         *
 *****************************************************************/
#ifdef FNV_64bitIntegers

/*
  512 bit FNV_prime = 2^344 + 2^8 + 0x57 =
   0x0000000000000000 0000000000000000
     0000000001000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000157 */
#define FNV512primeX 0x0157
#define FNV512shift 24

/* 0xB86DB0B1171F4416 DCA1E50F309990AC
     AC87D059C9000000 0000000000000D21
     E948F68A34C192F6 2EA79BC942DBE7CE
     182036415F56E34B AC982AAC4AFE9FD9 */

uint32_t FNV512basis[FNV512size/4] = {
         0xB86DB0B1, 0x171F4416, 0xDCA1E50F, 0x209990AC,
         0xAC87D059, 0x9C000000, 0x00000000, 0x00000D21,
         0xE948F68A, 0x34C192F6, 0x2EA79BC9, 0x42DBE7CE,
         0x18203641, 0x5F56E34B, 0xAC982AAC, 0x4AFE9FD9 };

/*****************************************************************
```

```
 *         Set of init, input, and output functions below           *
 *         to incrementally compute FNV512                          *
 ********************************************************************/

/* initialize context  (64 bit)
 ********************************************************************/
int FNV512init ( FNV512context *ctx )
{
if ( ctx )
    {
    for ( i=0; i<FNV512size/4; ++i )
        ctx->Hash[i] = FNV512basis[i];
    ctx->Computed = FNVinited+FNV512state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV512init */

/* initialize context with a provided basis  (64 bit)
 ********************************************************************/
int FNV512initBasis ( FNV512context* const ctx,
                      const uint8_t basis[FNV512size] )
{
int      i;
const uint8_t  *ui8p;
uint32_t  temp;

if ( ctx )
    {
#ifdef FNV_BigEndian
    ui8p = basis;
    for ( i=0; i < FNV512size/4; ++i )
        {
            temp = (*ui8p++)<<8;
            temp = (temp + *ui8p++)<<8;
            temp = (temp + *ui8p++)<<8;
        ctx->Hash[i] = temp +  *ui8p;
        }
#else
    ui8p = basis + (FNV512size/4 - 1);
    for ( i=0; i < FNV512size/4; ++i )
        {
            temp = (*ui8p--)<<8;
            temp = (temp + *ui8p--)<<8;
            temp = (temp + *ui8p--)<<8;
        ctx->Hash[i] = temp +  *ui8p;
        }
#endif
    ctx->Computed = FNVinited+FNV512state;
    return fnvSuccess;
```

```
    }
return fnvNull;
}   /* end FNV512initBasis */

/* hash in a counted block  (64 bit)
 ********************************************************************/
int FNV512blockin ( FNV512context *ctx,
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint64_t    temp[FNV512size/4];
uint64_t    temp2[3];

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV512state:
            ctx->Computed = FNVcomputed+FNV128state;
        case FNVcomputed+FNV512state:
            break;
        default:
            return fnvStateError;
        }
    if ( length < 0 )
        return fnvBadParam;
    for ( i=0; i<FNV512size/4; ++i )
         temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV512prime * ( temp ^ *in++ ); */
        temp[7] ^= *in++;
        temp2[2] = temp[7] << FNV512shift;
        temp2[1] = temp[6] << FNV512shift;
        temp2[0] = temp[5] << FNV512shift;
        for ( i=0; i<FNV512size/4; ++i )
            temp[i] *= FNV512primeX;
        temp[2] += temp2[2];
        temp[1] += temp2[1];
        temp[0] += temp2[0];
        for ( i=FNV512size/4-1; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
       }
    for ( i=0; i<FNV512size/4; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
```

```
    }
return fnvNull;
}   /* end FNV512input */

/* hash in a string  (64 bit)
 ****************************************************************/
inf FNV512stringin ( FNV512context *ctx, const char *in )
{
uint64_t   temp[FNV512size/4];
uint64_t   temp2[2];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV512state:
            ctx->Computed = FNVcomputed+FNV512state;
        case FNVcomputed+FNV512state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV512size/4; ++i )
         temp[i] = ctx->Hash[i];
    while ( ch = (uint8_t)*in++ )
        {
        /* temp = FNV512prime * ( temp ^ ch ); */
        temp[7] ^= ch;
        temp2[2] = temp[7] << FNV128shift;
        temp2[1] = temp[6] << FNV128shift;
        temp2[0] = temp[5] << FNV128shift;
        for ( i=0; i<FNV512size/4; ++i )
            temp[i] *= FNV512prime;
        temp[2] += temp2[2];
        temp[1] += temp2[1];
        temp[0] += temp2[0];
        for ( i=FNVsize512/4-1; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
       }
    for ( i=0; i<FNV512size/4; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV512stringin */
```

```
/* return hash  (64 bit)
 ************************************************************************/
int FNV512result ( FNV512context *ctx, uint8_t out[FNV512size] )
{
if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV512state )
        return fnvStateError;
    for ( i=0; i<FNV512size/4; ++i )
        {
#ifdef FNV_BigEndian
        out[15-2*i] = ctx->Hash[i];
        out[14-2*i] = ctx->Hash[i] >> 8;
#else
        out[2*i] = ctx->Hash[i];
        out[2*i+1] = ctx->Hash[i] >> 8;
#endif
        ctx -> Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV512state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV512result */

/*****************************************************************
 *        END VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC         *
 *****************************************************************/
#else    /*  FNV_64bitIntegers */
/*****************************************************************
 *      START VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC    *
 *****************************************************************/

/* version for when you only have 32-bit arithmetic
 ************************************************************************/

/*
  512 bit FNV_prime = 2^344 + 2^8 + 0x57 =
   0x0000000000000000 0000000000000000
     0000000001000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000157 */
#define FNV512primeX 0x0157
#define FNV512shift 8

/* 0xB86DB0B1171F4416 DCA1E50F309990AC
     AC87D059C9000000 0000000000000D21
     E948F68A34C192F6 2EA79BC942DBE7CE
     182036415F56E34B AC982AAC4AFE9FD9 */
```

```
uint16_t FNV512basis[FNV512size/2] = {
     0xB86D, 0xB0B1, 0x171F, 0x4416, 0xDCA1, 0xE50F, 0x3099, 0x90AC,
     0xAC87, 0xD059, 0xC900, 0x0000, 0x0000, 0x0000, 0x0000, 0x0D21,
     0xE948, 0xF68A, 0x34C1, 0x92F6, 0x2EA7, 0x9BC9, 0x42DB, 0xE7CE,
     0x1820, 0x3641, 0x5F56, 0xE34B, 0xAC98, 0x2AAC, 0x4AFE, 0x9FD9 };


/*****************************************************************
 *         Set of init, input, and output functions below        *
 *         to incrementally compute FNV512                       *
 *****************************************************************/

/* initialize context  (32 bit)
 *****************************************************************/
int FNV512init ( FNV512context *ctx )
{
int     i;

if ( ctx )
    {
    for ( i=0; i<FNV512size/2; ++i )
        ctx->Hash[i] = FNV512basis[i];
    ctx->Computed = FNVinited+FNV512state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV512init */

/* initialize context with a provided basis  (32 bit)
 *****************************************************************/
int FNV512initBasis ( FNV512context *ctx,
                      const uint8_t basis[FNV512size] )
{
int      i;
const uint8_t  *ui8p;
uint32_t temp;

if ( ctx )
    {
#ifdef FNV_BigEndian
    ui8p = basis;
    for ( i=0; i < FNV512size/2; ++i )
        {
        temp = *ui8p++;
        ctx->Hash[i] = ( temp<<8 ) + (*ui8p++);
        }
#else
    ui8p = basis + ( FNV512size/2 - 1 );
    for ( i=0; i < FNV512size/2; ++i )
        {
```

```
        temp = *ui8p--;
        ctx->Hash[i] = ( temp<<8 ) + (*ui8p--);
        }
#endif
    ctx->Computed = FNVinited+FNV512state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV512initBasis */

/* hash in a counted block  (32 bit)
 ******************************************************************/
int FNV512blockin ( FNV512context *ctx,
                    const void *vin,
                    long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t   temp[FNV512size/2];
uint32_t   temp2[6];
int        i;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV512state:
            ctx->Computed = FNVcomputed+FNV512state;
        case FNVcomputed+FNV512state:
            break;
        default:
            return fnvStateError;
        }
    if ( length < 0 )
        return fnvBadParam;
    for ( i=0; i<FNV512size/2; ++i )
        temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV512prime * ( temp ^ *in++ ); */
        temp[15] ^= *in++;
        for ( i=0; i<6; ++i )
            temp2[i] = temp[10+i] << FNV512shift;
        for ( i=0; i<FNV512size/2; ++i )
            temp[i] *= FNV512primeX;
        for ( i=0; i<6; ++i )
            temp[10+i] += temp2[i];
        for ( i=15; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
```

```
            }
        }
    for ( i=0; i<FNV512size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV512blockin */

/* hash in a string  (32 bit)
 ****************************************************************/
int FNV512stringin ( FNV512context *ctx, const char *in )
{
uint32_t   temp[FNV512size/2];
uint32_t   temp2[6];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV512state:
            ctx->Computed = FNVcomputed+FNV512state;
        case FNVcomputed+FNV512state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV512size/2; ++i )
         temp[i] = ctx->Hash[i];
    while ( (ch = (uint8_t)*in++) )
        {
        /* temp = FNV512prime * ( temp ^ *in++ ); */
        temp[15] ^= ch;
        for ( i=0; i<6; ++i )
            temp2[i] = temp[10+i] << FNV512shift;
        for ( i=0; i<FNV512size/2; ++i )
            temp[i] *= FNV512primeX;
        for ( i=0; i<6; ++i )
            temp[10+i] += temp2[i];
        for ( i=15; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
         }
    for ( i=0; i<FNV512size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
```

```
    }
return fnvNull;
}   /* end FNV512stringin */

/* return hash  (32 bit)
 ****************************************************************/
int FNV512result ( FNV512context *ctx, unsigned char out[16] )
{
int    i;

if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV512state )
        return fnvStateError;
    for ( i=0; i<FNV512size/2; ++i )
        {
#ifdef FNV_BigEndian
        out[31-2*i] = ctx->Hash[i];
        out[30-2*i] = ctx->Hash[i] >> 8;
#else
        out[2*i] = ctx->Hash[i];
        out[2*i+1] = ctx->Hash[i] >> 8;
#endif
        ctx->Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV512state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV512result */

#endif    /*  FNV_64bitIntegers */
/*****************************************************************
 *        END VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC      *
 *****************************************************************/

#endif    /* _FNV512_C_ */
```

<CODE ENDS>

6.1.6 FNV1024 C Code

The header and C source for 1024-bit FNV-1a returning a byte vector.

<CODE BEGINS>

```
/************************** FNV1024.h ************************/
/***************** See RFC NNNN for details. ******************/
/*
 * Copyright (c) 2016 IETF Trust and the persons identified as
```

```
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */

#ifndef _FNV1024_H_
#define _FNV1024_H_

/*
 *  Description:
 *      This file provides headers for the 1024-bit version of the
 *      FNV-1a non-cryptographic hash algorithm.
 */

#include "FNVconfig.h"

#include <stdint.h>
#define FNV1024size (1024/8)

/* If you do not have the ISO standard stdint.h header file, then you
 * must typedef the following types:
 *
 *    type              meaning
 *  uint64_t         unsigned 64 bit integer (ifdef FNV_64bitIntegers)
 *  uint32_t         unsigned 32 bit integer
 *  uint16_t         unsigned 16 bit integer
 *  uint8_t          unsigned 8 bit integer (i.e., unsigned char)
 */

#ifndef _FNV_ErrCodes_
#define _FNV_ErrCodes_
/*****************************************************************
 *  All FNV functions provided return as integer as follows:
 *       0 -> success
 *      >0 -> error as listed below
 */
enum {    /* success and errors */
    fnvSuccess = 0,
    fnvNull,            /* Null pointer parameter */
    fnvStateError,      /* called Input after Result or before Init */
    fnvBadParam         /* passed a bad parameter */
};
#endif /* _FNV_ErrCodes_ */

/*
 *  This structure holds context information for an FNV1024 hash
 */
#ifdef FNV_64bitIntegers
    /* version if 64 bit integers supported */
typedef struct FNV1024context_s {
        int Computed;  /* state */
```

```
        uint32_t Hash[FNV1024size/4];
} FNV1024context;

#else
    /* version if 64 bit integers NOT supported */

typedef struct FNV1024context_s {
        int Computed;  /* state */
        uint16_t Hash[FNV1024size/2];
} FNV1024context;

#endif /* FNV_64bitIntegers */

/*
 *  Function Prototypes
 *    FNV1024string: hash a zero terminated string not including
 *                   the terminating zero
 *    FNV1024block: FNV1024 hash a specified length byte vector
 *    FNV1024init: initializes an FNV1024 context
 *    FNV1024initBasis: initializes an FNV1024 context with a
 *                      provided basis
 *    FNV1024blockin: hash in a specified length byte vector
 *    FNV1024stringin: hash in a zero terminated string not
 *                     including the zero
 *    FNV1024result: returns the hash value
 *
 *    Hash is returned as an array of 8-bit integers
 */

#ifdef __cplusplus
extern "C" {
#endif

/* FNV1024 */
extern int FNV1024string ( const char *in,
                           unsigned char out[FNV1024size] );
extern int FNV1024block ( const void *in,
                          long int length,
                          unsigned char out[FNV1024size] );
extern int FNV1024init ( FNV1024context *);
extern int FNV1024initBasis ( FNV1024context * const,
                              const uint8_t basis[FNV1024size] );
extern int FNV1024blockin ( FNV1024context *,
                            const void *in,
                            long int length );
extern int FNV1024stringin ( FNV1024context *,
                             const char *in );
extern int FNV1024result ( FNV1024context *,
                           unsigned char out[FNV1024size] );
```

```
#ifdef __cplusplus
}
#endif

#endif /* _FNV1024_H_ */
```

<CODE ENDS>

<CODE BEGINS>

```
/****************************** FNV1024.c ***************************/
/******************** See RFC NNNN for details **********************/
/* Copyright (c) 2016, 2017 IETF Trust and the persons identified as
 * authors of the code.  All rights
 * See fnv-private.h for terms of use and redistribution.
 */

/* This file implements the FNV (Fowler, Noll, Vo) non-cryptographic
 * hash function FNV-1a for 1024-bit hashes.
 */

#ifndef _FNV1024_C_
#define _FNV1024_C_

#include "fnv-private.h"
#include "FNV1024.h"

/* common code for 64 and 32 bit modes */

/* FNV1024 hash a null terminated string  (64/32 bit)
 ***********************************************************************/
int FNV1024string ( const char *in, uint8_t out[FNV1024size] )
{
FNV1024context    ctx;
int              err;

if ( (err = FNV1024init ( &ctx )) != fnvSuccess)
    return err;
if ( (err = FNV1024stringin ( &ctx, in )) != fnvSuccess)
    return err;
return FNV1024result ( &ctx, out );
}   /* end FNV1024string */

/* FNV1024 hash a counted block  (64/32 bit)
 ***********************************************************************/
int FNV1024block ( const void *in,
                   long int length,
                   uint8_t out[FNV1024size] )
{
FNV1024context    ctx;
int              err;
```

```
if ( (err = FNV1024init ( &ctx )) != fnvSuccess)
    return err;
if ( (err = FNV1024blockin ( &ctx, in, length)) != fnvSuccess)
    return err;
return FNV1024result ( &ctx, out );
}   /* end FNV1024block */


/*****************************************************************
 *        START VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC      *
 *****************************************************************/
#ifdef FNV_64bitIntegers

/*
 1024 bit FNV_prime = 2^680 + 2^8 + 0x8d =
   0x0000000000000000 0000010000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 000000000000018D
#define FNV1024primeX 0x018D
#define FNV1024shift 24

/* 0x0000000000000000 005F7A76758ECC4D
     32E56D5A591028B7 4B29FC4223FDADA1
     6C3BF34EDA3674DA 9A21D90000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 000000000004C6D7
     EB6E73802734510A 555F256CC005AE55
     6BDE8CC9C6A93B21 AFF4B16C71EE90B3 */

uint32_t FNV1024basis[FNV1024size/4] = {
  0x00000000, 0x00000000, 0x005F7A76, 0x758ECC4D,
  0x32E56D5A, 0x591028B7, 0x4B29FC42, 0x23FDADA1,
  0x6C3BF34E, 0xDA3674DA, 0x9A21D900, 0x00000000,
  0x00000000, 0x00000000, 0x00000000, 0x00000000,
  0x00000000, 0x00000000, 0x00000000, 0x00000000,
  0x00000000, 0x00000000, 0x00000000, 0x0004C6D7,
  0xEB6E7380, 0x2734510A, 0x555F256C, 0xC005AE55,
  0x6BDE8CC9, 0xC6A93B21, 0xAFF4B16C, 0x71EE90B3
 };

/*****************************************************************
 *         Set of init, input, and output functions below        *
 *         to incrementally compute FNV1024                       *
 *****************************************************************/

/* initialize context  (64 bit)
```

```
 ***************************************************************/
int FNV1024init ( FNV1024context *ctx )
{
if ( ctx )
    {
    for ( i=0; i<FNV1024size/4; ++i )
        ctx->Hash[i] = FNV1024basis[i];
    ctx->Computed = FNVinited+FNV1024state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024init */

/* initialize context with a provided basis  (64 bit)
 ***************************************************************/
int FNV1024initBasis ( FNV1024context* const ctx,
                       const uint8_t basis[FNV1024size] )
{
int      i;
uint8_t  *ui8p;
uint32_t  temp;

if ( ctx )
    {
#ifdef FNV_BigEndian
    ui8p = basis;
    for ( i=0; i < FNV1024size/4; ++i )
        {
            temp = (*ui8p++)<<8;
            temp = (temp + *ui8p++)<<8;
            temp = (temp + *ui8p++)<<8;
        ctx->Hash[i] = temp +  *ui8p;
        }
#else
    ui8p = basis + (FNV1024size/4 - 1);
    for ( i=0; i < FNV1024size/4; ++i )
        {
            temp = (*ui8p--)<<8;
            temp = (temp + *ui8p--)<<8;
            temp = (temp + *ui8p--)<<8;
        ctx->Hash[i] = temp +  *ui8p;
        }
#endif
    ctx->Computed = FNVinited+FNV1024state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024initBasis */

/* hash in a counted block  (64 bit)
```

```
 ***************************************************************/
int FNV1024blockin ( FNV1024context *ctx,
                     const void *vin,
                     long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint64_t    temp[FNV1024size/4];
uint64_t    temp2[3];

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV1024state:
            ctx->Computed = FNVcomputed+FNV128state;
        case FNVcomputed+FNV1024state:
            break;
        default:
            return fnvStateError;
        }
    if ( length < 0 )
        return fnvBadParam;
    for ( i=0; i<FNV1024size/4; ++i )
         temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV1024prime * ( temp ^ *in++ ); */
        temp[7] ^= *in++;
        temp2[2] = temp[7] << FNV1024shift;
        temp2[1] = temp[6] << FNV1024shift;
        temp2[0] = temp[5] << FNV1024shift;
        for ( i=0; i<FNV1024size/4; ++i )
            temp[i] *= FNV1024primeX;
        temp[2] += temp2[2];
        temp[1] += temp2[1];
        temp[0] += temp2[0];
        for ( i=FNV1024size/4-1; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
       }
    for ( i=0; i<FNV1024size/4; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024input */

/* hash in a string  (64 bit)
```

```
 ***************************************************************/
inf FNV1024stringin ( FNV1024context *ctx, const char *in )
{
uint64_t   temp[FNV1024size/4];
uint64_t   temp2[2];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV1024state:
            ctx->Computed = FNVcomputed+FNV1024state;
        case FNVcomputed+FNV1024state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV1024size/4; ++i )
         temp[i] = ctx->Hash[i];
    while ( ch = (uint8_t)*in++ )
        {
        /* temp = FNV1024prime * ( temp ^ ch ); */
        temp[7] ^= ch;
        temp2[2] = temp[7] << FNV128shift;
        temp2[1] = temp[6] << FNV128shift;
        temp2[0] = temp[5] << FNV128shift;
        for ( i=0; i<FNV1024size/4; ++i )
            temp[i] *= FNV1024prime;
        temp[2] += temp2[2];
        temp[1] += temp2[1];
        temp[0] += temp2[0];
        for ( i=FNVsize1024/4-1; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
       }
    for ( i=0; i<FNV1024size/4; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024stringin */

/* return hash  (64 bit)
 ***************************************************************/
int FNV1024result ( FNV1024context *ctx, uint8_t out[FNV1024size] )
{
```

```
if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV1024state )
        return fnvStateError;
    for ( i=0; i<FNV1024size/4; ++i )
        {
#ifdef FNV_BigEndian
        out[15-2*i] = ctx->Hash[i];
        out[14-2*i] = ctx->Hash[i] >> 8;
#else
        out[2*i] = ctx->Hash[i];
        out[2*i+1] = ctx->Hash[i] >> 8;
#endif
        ctx -> Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV1024state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024result */

/*****************************************************************
 *        END VERSION FOR WHEN YOU HAVE 64 BIT ARITHMETIC        *
 *****************************************************************/
#else    /*  FNV_64bitIntegers */
/*****************************************************************
 *      START VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC   *
 *****************************************************************/

/* version for when you only have 32-bit arithmetic
 *****************************************************************/

/*
 1024 bit FNV_prime = 2^680 + 2^8 + 0x8d =
   0x0000000000000000 0000010000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 000000000000018D */
#define FNV1024primeX 0x018D
#define FNV1024shift 8

/* 0x0000000000000000 005F7A76758ECC4D
     32E56D5A591028B7 4B29FC4223FDADA1
     6C3BF34EDA3674DA 9A21D90000000000
     0000000000000000 0000000000000000
     0000000000000000 0000000000000000
     0000000000000000 000000000004C6D7
     EB6E73802734510A 555F256CC005AE55
```

```
     6BDE8CC9C6A93B21 AFF4B16C71EE90B3 */

uint16_t FNV1024basis[FNV1024size/2] = {
  0x0000, 0x0000, 0x0000, 0x0000, 0x005F, 0x7A76, 0x758E, 0xCC4D,
  0x32E5, 0x6D5A, 0x5910, 0x28B7, 0x4B29, 0xFC42, 0x23FD, 0xADA1,
  0x6C3B, 0xF34E, 0xDA36, 0x74DA, 0x9A21, 0xD900, 0x0000, 0x0000,
  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
  0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0004, 0xC6D7,
  0xEB6E, 0x7380, 0x2734, 0x510A, 0x555F, 0x256C, 0xC005, 0xAE55,
  0x6BDE, 0x8CC9, 0xC6A9, 0x3B21, 0xAFF4, 0xB16C, 0x71EE, 0x90B3
};


/**********************************************************************
 *         Set of init, input, and output functions below             *
 *         to incrementally compute FNV1024                           *
 **********************************************************************/

/* initialize context  (32 bit)
 **********************************************************************/
int FNV1024init ( FNV1024context *ctx )
{
int     i;

if ( ctx )
    {
    for ( i=0; i<FNV1024size/2; ++i )
        ctx->Hash[i] = FNV1024basis[i];
    ctx->Computed = FNVinited+FNV1024state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024init */

/* initialize context with a provided basis  (32 bit)
 **********************************************************************/
int FNV1024initBasis ( FNV1024context *ctx,
                       const uint8_t basis[FNV1024size] )
{
int      i;
const uint8_t  *ui8p;
uint32_t temp;

if ( ctx )
    {
#ifdef FNV_BigEndian
    ui8p = basis;
    for ( i=0; i < FNV1024size/2; ++i )
        {
```

```
        temp = *ui8p++;
        ctx->Hash[i] = ( temp<<8 ) + (*ui8p++);
        }
#else
    ui8p = basis + ( FNV1024size/2 - 1 );
    for ( i=0; i < FNV1024size/2; ++i )
        {
        temp = *ui8p--;
        ctx->Hash[i] = ( temp<<8 ) + (*ui8p--);
        }
#endif
    ctx->Computed = FNVinited+FNV1024state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024initBasis */

/* hash in a counted block  (32 bit)
 ******************************************************************/
int FNV1024blockin ( FNV1024context *ctx,
                     const void *vin,
                     long int length )
{
const uint8_t *in = (const uint8_t*)vin;
uint32_t   temp[FNV1024size/2];
uint32_t   temp2[6];
int        i;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV1024state:
            ctx->Computed = FNVcomputed+FNV1024state;
        case FNVcomputed+FNV1024state:
            break;
        default:
            return fnvStateError;
        }
    if ( length < 0 )
        return fnvBadParam;
    for ( i=0; i<FNV1024size/2; ++i )
        temp[i] = ctx->Hash[i];
    for ( ; length > 0; length-- )
        {
        /* temp = FNV1024prime * ( temp ^ *in++ ); */
        temp[15] ^= *in++;
        for ( i=0; i<6; ++i )
            temp2[i] = temp[10+i] << FNV1024shift;
        for ( i=0; i<FNV1024size/2; ++i )
```

```
            temp[i] *= FNV1024primeX;
        for ( i=0; i<6; ++i )
            temp[10+i] += temp2[i];
        for ( i=15; i>0; --i )
            {
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
        }
    for ( i=0; i<FNV1024size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024blockin */

/* hash in a string  (32 bit)
 ***************************************************************/
int FNV1024stringin ( FNV1024context *ctx, const char *in )
{
uint32_t   temp[FNV1024size/2];
uint32_t   temp2[6];
int        i;
uint8_t    ch;

if ( ctx && in )
    {
    switch ( ctx->Computed )
        {
        case FNVinited+FNV1024state:
            ctx->Computed = FNVcomputed+FNV1024state;
        case FNVcomputed+FNV1024state:
            break;
        default:
             return fnvStateError;
         }
    for ( i=0; i<FNV1024size/2; ++i )
         temp[i] = ctx->Hash[i];
    while ( (ch = (uint8_t)*in++) )
        {
        /* temp = FNV1024prime * ( temp ^ *in++ ); */
        temp[15] ^= ch;
        for ( i=0; i<6; ++i )
            temp2[i] = temp[10+i] << FNV1024shift;
        for ( i=0; i<FNV1024size/2; ++i )
            temp[i] *= FNV1024primeX;
        for ( i=0; i<6; ++i )
            temp[10+i] += temp2[i];
        for ( i=15; i>0; --i )
            {
```

```
            temp[i-1] += temp[i] >> 16;
            temp[i] &= 0xFFFF;
            }
         }
    for ( i=0; i<FNV1024size/2; ++i )
        ctx->Hash[i] = temp[i];
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024stringin */

/* return hash  (32 bit)
 ********************************************************************/
int FNV1024result ( FNV1024context *ctx, unsigned char out[16] )
{
int    i;

if ( ctx && out )
    {
    if ( ctx->Computed != FNVcomputed+FNV1024state )
        return fnvStateError;
    for ( i=0; i<FNV1024size/2; ++i )
        {
#ifdef FNV_BigEndian
        out[31-2*i] = ctx->Hash[i];
        out[30-2*i] = ctx->Hash[i] >> 8;
#else
        out[2*i] = ctx->Hash[i];
        out[2*i+1] = ctx->Hash[i] >> 8;
#endif
        ctx->Hash[i] = 0;
        }
    ctx->Computed = FNVemptied+FNV1024state;
    return fnvSuccess;
    }
return fnvNull;
}   /* end FNV1024result */

#endif    /*  FNV_64bitIntegers */
/*********************************************************************
 *        END VERSION FOR WHEN YOU ONLY HAVE 32-BIT ARITHMETIC       *
 *********************************************************************/

#endif    /* _FNV1024_C_ */
   <CODE ENDS>
```

6.2 FNV Test Code

Here is a test driver:

<CODE BEGINS>

```
/*************************** MAIN.c ***************************/
/****************** See RFC NNNN for details. *****************/
/*
 * Copyright (c) 2016 IETF Trust and the persons identified as
 * authors of the code.  All rights reserved.
 * See fnv-private.h for terms of use and redistribution.
 */
/* to do a thorough test you need to run will all four
 combinations of the following defined/undefined */

// #define FNV_64bitIntegers
// #define FNV_BigEndian

#include <stdio.h>
#include <string.h>

#include "fnv-private.h"
#include "FNV.h"

/* global variables */
char    *funcName;
char    *errteststring = "foo";
int     Terr; /* Total errors */
#define NTestBytes 3
uint8_t errtestbytes[NTestBytes] = { (uint8_t)1,
    (uint8_t)2, (uint8_t)3 };

#define NTstrings 3
char    *teststring[NTstrings] = { "", "a", "foobar" };

/*****************************************************************
 * local prototypes
 *****************************************************************/
int TestR ( char *, int should, int was );
void TestNValue ( char *subfunc,
                  char *string,
                  int N,
                  uint8_t *should,
                  uint8_t *was );
void HexPrint ( int i, unsigned char *p );
void Test32 ();
void Test32Value ( char *subfunc, char *string,
                   uint32_t was, uint32_t should );
void Test64 ();
#ifdef FNV_64bitIntegers
```

```
void Test64Value ( char *subfunc, char *string,
                   uint64_t should, uint64_t was );
#else
#define uint64_t foobar
#endif /* FNB_64bitIntegers */
void Test128 ();
void Test256 ();
void Test512 ();
void Test1024 ();

void TestNValue ( char *subfunc,
                  char *string,
                  int N,
                  uint8_t was[N],
                  uint8_t should[N] );

/*****************************************************************
 * main
 *****************************************************************/
int main (int argc, const char * argv[])
{
#ifdef FNV_64bitIntegers
    printf ("Have 64-bit Integers. ");
#else
    printf ("Do not have 64-bit integers. ");
#endif
#ifdef FNV_BigEndian
    printf ("Calculating for Big Endian.0);
#else
    printf ("Not calculating for Big Endian.0);
#endif
    funcName = "Testing TestR ";
         /* test the Test Return function */
    TestR ( "should fail", 1, 2 );
    TestR ( "should not have failed", 0, 0 );

    Test32();
    Test64();
    Test128();
    Test256();
    Test512();
    Test1024();

    printf ("Type return to exit.0);
    (void)getchar();
    printf ("Goodbye!0);

    return 0;
}     /* end main */
```

```
/* Test status returns
 ****************************************************************/
int TestR ( char *name, int expect, int actual )
{
if ( expect != actual )
    {
    printf ( "%s%s returned %i instead of %i.0,
              funcName, name, actual, expect );
    ++Terr;
    }
return actual;
}    /* end TestR */

/* Return true if the bytes are in reverse order from each other */
int revcmp(uint8_t *buf1, uint8_t *buf2, int N) {
  int i;
  uint8_t *bufc = buf2 + N;
  for ( i = 0; i < N / 2; i++ )
    if (*buf1++ != *--bufc)
      return 0;
  return 1;
}

/* General byte vector return error test
 ****************************************************************/
void TestNValue ( char *subfunc,
                  char *string,
                  int N,
                  uint8_t *was,
                  uint8_t *should )
{
#ifdef FNV_BigEndian
if ( revcmp ( was, should, N) == 0)
#else
if ( memcmp ( was, should, N) != 0)
#endif
    {
    printf ( "%s %s of '%s' computed ", funcName, subfunc, string );
    HexPrint ( N, was );
    printf ( ", should have been " );
    HexPrint ( N, should );
    printf ( ".0 );
    ++Terr;
    }
}   /* end TestNValue */

/* print some hex
 ****************************************************************/
 void HexPrint ( int count, unsigned char *ptr )
{
```

```
int   i;

for ( i = 0; i < count; ++i )
  printf ( "%02X", ptr[i] );
}   /* end HexPrint */


/*************************************************************
 * FNV32 test
 *************************************************************/
void Test32 ()
{
int            i, err;
long int       iLen;
uint32_t       eUint32;
FNV32context   eContext;
uint32_t    FNV32svalues[NTstrings] = {
    0x811c9dc5, 0xe40c292c, 0xbf9cf968 };
uint32_t    FNV32bvalues[NTstrings] = {
    0x050c5d1f, 0x2b24d044, 0x0c1c9eb8 };

/* test Test32Value */
funcName = "Test32Value";
Test32Value ( "should fail", "test", FNV32svalues[1], FNV32svalues[2] );

funcName = "FNV32";

/* test error checks */
Terr = 0;
TestR ( "init", fnvSuccess, FNV32init (&eContext) );
    TestR ( "string", fnvNull,
            FNV32string ( (char *)0, &eUint32 ) );
    TestR ( "string", fnvNull,
            FNV32string ( errteststring, (uint32_t *)0 ) );
    TestR ( "block", fnvNull,
            FNV32block ( (uint8_t *)0, 1, &eUint32 ) );
    TestR ( "block", fnvBadParam,
            FNV32block ( errtestbytes, -1, &eUint32 ) );
    TestR ( "block", fnvNull,
            FNV32block ( errtestbytes, 1, (uint32_t *)0 ) );
    TestR ( "init", fnvNull,
            FNV32init ( (FNV32context *)0 ) );
    TestR ( "initBasis", fnvNull,
            FNV32initBasis ( (FNV32context *)0, 42 ) );
    TestR ( "blockin", fnvNull,
            FNV32blockin ( (FNV32context *)0,
                           errtestbytes, NTestBytes ) );
    TestR ( "blockin", fnvNull,
            FNV32blockin ( &eContext, (uint8_t *)0,
                           NTestBytes ) );
```

```
    TestR ( "blockin", fnvBadParam,
            FNV32blockin ( &eContext, errtestbytes, -1 ) );
    eContext.Computed = FNVclobber+FNV32state;
    TestR ( "blockin", fnvStateError,
            FNV32blockin ( &eContext, errtestbytes,
                           NTestBytes ) );
    TestR ( "stringin", fnvNull,
            FNV32stringin ( (FNV32context *)0, errteststring ) );
    TestR ( "stringin", fnvNull,
            FNV32stringin ( &eContext, (char *)0 ) );
    TestR ( "stringin", fnvStateError,
            FNV32stringin ( &eContext, errteststring ) );
    TestR ( "result", fnvNull,
            FNV32result ( (FNV32context *)0, &eUint32 ) );
    TestR ( "result", fnvNull,
            FNV32result ( &eContext, (uint32_t *)0  ) );
    TestR ( "result", fnvStateError,
            FNV32result ( &eContext, &eUint32  ) );
if ( Terr )
    printf ( "%s test of error checks failed %i times.0,
             funcName, Terr );
else
    printf ( "%s test of error checks passed0, funcName );

/* test actual results */
Terr = 0;
for ( i = 0; i < NTstrings; ++i )
    {
    err = TestR ( "string", fnvSuccess,
                  FNV32string ( teststring[i], &eUint32 ) );
    if ( err == fnvSuccess )
        Test32Value ( "string", teststring[i], eUint32,
                      FNV32svalues[i] );
    err = TestR ( "block", fnvSuccess,
                  FNV32block ( (uint8_t *)teststring[i],
                               (unsigned long)(strlen(teststring[i])+1),
                               &eUint32 ) );
    if ( err == fnvSuccess )
        Test32Value ( "block", teststring[i], eUint32,
                      FNV32bvalues[i] );
    /* now try testing the incremental stuff */
    err = TestR ( "init", fnvSuccess, FNV32init ( &eContext ) );
    if ( err ) break;
    iLen = strlen ( teststring[i] );
    err = TestR ( "blockin", fnvSuccess,
                  FNV32blockin ( &eContext,
                                 (uint8_t *)teststring[i],
                                 iLen/2 ) );
    if ( err ) break;
    err = TestR ( "stringin", fnvSuccess,
```

```
                FNV32stringin ( &eContext,
                              teststring[i] + iLen/2 ) );
    err = TestR ( "result", fnvSuccess,
                FNV32result ( &eContext, &eUint32 ) );
    if ( err ) break;
    Test32Value ( " incremental", teststring[i], eUint32,
                FNV32svalues[i] );
    }
if ( Terr )
    printf ( "%s test of return values failed %i times.0,
             funcName, Terr );
else
    printf ( "%s test of return values passed.0, funcName );
}    /* end Test32 */

/* start Test32Value
 ****************************************************************/
void Test32Value ( char *subfunc,
                   char *string,
                   uint32_t was,
                   uint32_t should )
{
  TestNValue(subfunc, string, sizeof(uint32_t), (uint8_t*)&was,
                                                (uint8_t*)&should);
}   /* end Test32Value */


#ifdef FNV_64bitIntegers
/***************************************************************
 * Code for FNV64 using 64-bit integers
 ***************************************************************/

void Test64 ()
{
int    i, err;
uint64_t   eUint64 = 42;
FNV64context   eContext;
uint64_t   FNV64svalues[NTstrings] = {
 0xcbf29ce484222325, 0xaf63dc4c8601ec8c, 0x85944171f73967e8 };
uint64_t   FNV64bvalues[NTstrings] = {
 0xaf63bd4c8601b7df, 0x089be207b544f1e4, 0x34531ca7168b8f38 };

funcName = "FNV64";

/* test error checks */
Terr = 0;
TestR ( "init", fnvSuccess, FNV64init (&eContext) );
    TestR ( "string", fnvNull,
             FNV64string ( (char *)0, &eUint64 ) );
    TestR ( "string", fnvNull,
```

```
            FNV64string ( errteststring, (uint64_t *)0 ) );
    TestR ( "block", fnvNull,
            FNV64block ( (uint8_t *)0, 1, &eUint64 ) );
    TestR ( "block", fnvBadParam,
            FNV64block ( errtestbytes, -1, &eUint64 ) );
    TestR ( "block", fnvNull,
            FNV64block ( errtestbytes, 1, (uint64_t *)0 ) );
    TestR ( "init", fnvNull,
            FNV64init ( (FNV64context *)0 ) );
    TestR ( "initBasis", fnvNull,
            FNV64initBasis ( (FNV64context *)0, 42 ) );
    TestR ( "blockin", fnvNull,
            FNV64blockin ( (FNV64context *)0,
                           errtestbytes, NTestBytes ) );
    TestR ( "blockin", fnvNull,
            FNV64blockin ( &eContext, (uint8_t *)0,
                           NTestBytes ) );
    TestR ( "blockin", fnvBadParam,
            FNV64blockin ( &eContext, errtestbytes, -1 ) );
    eContext.Computed = FNVclobber+FNV64state;
    TestR ( "blockin", fnvStateError,
            FNV64blockin ( &eContext, errtestbytes,
                           NTestBytes ) );
    TestR ( "stringin", fnvNull,
            FNV64stringin ( (FNV64context *)0, errteststring ) );
    TestR ( "stringin", fnvNull,
            FNV64stringin ( &eContext, (char *)0 ) );
    TestR ( "stringin", fnvStateError,
            FNV64stringin ( &eContext, errteststring ) );
    TestR ( "result", fnvNull,
            FNV64result ( (FNV64context *)0, &eUint64 ) );
    TestR ( "result", fnvNull,
            FNV64result ( &eContext, (uint64_t *)0  ) );
    TestR ( "result", fnvStateError,
            FNV64result ( &eContext, &eUint64  ) );
if ( Terr )
    printf ( "%s test of error checks failed %i times.0,
             funcName, Terr );
else
    printf ( "%s test of error checks passed0, funcName );

/* test actual results */
Terr = 0;
for ( i = 0; i < NTstrings; ++i )
    {
    err = TestR ( "string", fnvSuccess,
                  FNV64string ( teststring[i], &eUint64 ) );
    if ( err == fnvSuccess )
        Test64Value ( "string", teststring[i], eUint64,
                      FNV64svalues[i] );
```

```
    err = TestR ( "block", fnvSuccess,
                  FNV64block ( (uint8_t *)teststring[i],
                               (unsigned long)(strlen(teststring[i])+1),
                               &eUint64 ) );
    if ( err == fnvSuccess )
         Test64Value ( "block", teststring[i], eUint64,
                       FNV64bvalues[i] );
    /* now try testing the incremental stuff */
    err = TestR ( "init", fnvSuccess, FNV64init ( &eContext ) );

    }
if ( Terr )
    printf ( "%s test of return values failed %i times.0,
             funcName, Terr );
else
    printf ( "%s test of return values passed.0, funcName );
}    /* end Test64 */

/* start Test64Value
 ****************************************************************/
 void Test64Value ( char *subfunc,
                    char *string,
                    uint64_t should,
                    uint64_t was )
{
  TestNValue(subfunc, string, sizeof(uint64_t), (uint8_t*)&was,
.                                          (uint8_t*)&should);
}   /* end Test64Value */
#else
void Test64 ()
{
  /* TBD */
}
#endif /* FNV_64bitIntegers */


/****************************************************************
 * Code for FNV128 using 64-bit integers
 ****************************************************************/

void Test128 ()
{
//int            i, err;
uint8_t        eUint128[FNV128size];
FNV128context  eContext;

funcName = "FNV128";

/* test error checks */
Terr = 0;
```

```
TestR ( "init", fnvSuccess, FNV128init (&eContext) );
    TestR ( "string", fnvNull,
            FNV128string ( (char *)0, eUint128 ) );
    TestR ( "string", fnvNull,
            FNV128string ( errteststring, (uint8_t *)0 ) );
    TestR ( "block", fnvNull,
            FNV128block ( (uint8_t *)0, 1, eUint128 ) );
    TestR ( "block", fnvBadParam,
            FNV128block ( errtestbytes, -1, eUint128 ) );
    TestR ( "block", fnvNull,
            FNV128block ( errtestbytes, 1, (uint8_t *)0 ) );
    TestR ( "init", fnvNull,
            FNV128init ( (FNV128context *)0 ) );
    TestR ( "initBasis", fnvNull,
            FNV128initBasis ( (FNV128context *)0, eUint128 ) );
    TestR ( "blockin", fnvNull,
            FNV128blockin ( (FNV128context *)0,
                            errtestbytes, NTestBytes ) );
    TestR ( "blockin", fnvNull,
            FNV128blockin ( &eContext, (uint8_t *)0,
                            NTestBytes ) );
    TestR ( "blockin", fnvBadParam,
            FNV128blockin ( &eContext, errtestbytes, -1 ) );
    eContext.Computed = FNVclobber+FNV128state;
    TestR ( "blockin", fnvStateError,
            FNV128blockin ( &eContext, errtestbytes,
                            NTestBytes ) );
    TestR ( "stringin", fnvNull,
            FNV128stringin ( (FNV128context *)0, errteststring ) );
    TestR ( "stringin", fnvNull,
            FNV128stringin ( &eContext, (char *)0 ) );
    TestR ( "stringin", fnvStateError,
            FNV128stringin ( &eContext, errteststring ) );
    TestR ( "result", fnvNull,
            FNV128result ( (FNV128context *)0, eUint128 ) );
    TestR ( "result", fnvNull,
            FNV128result ( &eContext, (uint8_t *)0  ) );
    TestR ( "result", fnvStateError,
            FNV128result ( &eContext, eUint128  ) );
if ( Terr )
    printf ( "%s test of error checks failed %i times.0,
             funcName, Terr );
else
    printf ( "%s test of error checks passed0, funcName );

/* test actual results */
Terr = 0;
/* tbd */
}    /* end Test128 */
```

```
/*****************************************************************
 * Code for FNV256 using 64-bit integers
 *****************************************************************/

void Test256 ()
{
//int            i, err;
uint8_t        eUint256[FNV256size];
FNV256context  eContext;

funcName = "FNV256";

/* test error checks */
Terr = 0;
TestR ( "init", fnvSuccess, FNV256init (&eContext) );
    TestR ( "string", fnvNull,
            FNV256string ( (char *)0, eUint256 ) );
    TestR ( "string", fnvNull,
            FNV256string ( errteststring, (uint8_t *)0 ) );
    TestR ( "block", fnvNull,
            FNV256block ( (uint8_t *)0, 1, eUint256 ) );
    TestR ( "block", fnvBadParam,
            FNV256block ( errtestbytes, -1, eUint256 ) );
    TestR ( "block", fnvNull,
            FNV256block ( errtestbytes, 1, (uint8_t *)0 ) );
    TestR ( "init", fnvNull,
            FNV256init ( (FNV256context *)0 ) );
    TestR ( "initBasis", fnvNull,
            FNV256initBasis ( (FNV256context *)0, eUint256 ) );
    TestR ( "blockin", fnvNull,
            FNV256blockin ( (FNV256context *)0,
                            errtestbytes, NTestBytes ) );
    TestR ( "blockin", fnvNull,
            FNV256blockin ( &eContext, (uint8_t *)0,
                            NTestBytes ) );
    TestR ( "blockin", fnvBadParam,
            FNV256blockin ( &eContext, errtestbytes, -1 ) );
    eContext.Computed = FNVclobber+FNV256state;
    TestR ( "blockin", fnvStateError,
            FNV256blockin ( &eContext, errtestbytes,
                            NTestBytes ) );
    TestR ( "stringin", fnvNull,
            FNV256stringin ( (FNV256context *)0, errteststring ) );
    TestR ( "stringin", fnvNull,
            FNV256stringin ( &eContext, (char *)0 ) );
    TestR ( "stringin", fnvStateError,
            FNV256stringin ( &eContext, errteststring ) );
    TestR ( "result", fnvNull,
            FNV256result ( (FNV256context *)0, eUint256 ) );
    TestR ( "result", fnvNull,
```

```
            FNV256result ( &eContext, (uint8_t *)0  ) );
    TestR ( "result", fnvStateError,
            FNV256result ( &eContext, eUint256  ) );
if ( Terr )
    printf ( "%s test of error checks failed %i times.0,
             funcName, Terr );
else
    printf ( "%s test of error checks passed0, funcName );

/* test actual results */
Terr = 0;
/* tbd */
}    /* end Test256 */


/*****************************************************************
 * Code for FNV512 using 64-bit integers
 *****************************************************************/

void Test512 ()
{
//int            i, err;
uint8_t        eUint512[FNV512size];
FNV512context  eContext;

funcName = "FNV512";

/* test error checks */
Terr = 0;
TestR ( "init", fnvSuccess, FNV512init (&eContext) );
    TestR ( "string", fnvNull,
            FNV512string ( (char *)0, eUint512 ) );
    TestR ( "string", fnvNull,
            FNV512string ( errteststring, (uint8_t *)0 ) );
    TestR ( "block", fnvNull,
            FNV512block ( (uint8_t *)0, 1, eUint512 ) );
    TestR ( "block", fnvBadParam,
            FNV512block ( errtestbytes, -1, eUint512 ) );
    TestR ( "block", fnvNull,
            FNV512block ( errtestbytes, 1, (uint8_t *)0 ) );
    TestR ( "init", fnvNull,
            FNV512init ( (FNV512context *)0 ) );
    TestR ( "initBasis", fnvNull,
            FNV512initBasis ( (FNV512context *)0, eUint512 ) );
    TestR ( "blockin", fnvNull,
            FNV512blockin ( (FNV512context *)0,
                            errtestbytes, NTestBytes ) );
    TestR ( "blockin", fnvNull,
            FNV512blockin ( &eContext, (uint8_t *)0,
                            NTestBytes ) );
```

```
    TestR ( "blockin", fnvBadParam,
            FNV512blockin ( &eContext, errtestbytes, -1 ) );
    eContext.Computed = FNVclobber+FNV512state;
    TestR ( "blockin", fnvStateError,
            FNV512blockin ( &eContext, errtestbytes,
                            NTestBytes ) );
    TestR ( "stringin", fnvNull,
            FNV512stringin ( (FNV512context *)0, errteststring ) );
    TestR ( "stringin", fnvNull,
            FNV512stringin ( &eContext, (char *)0 ) );
    TestR ( "stringin", fnvStateError,
            FNV512stringin ( &eContext, errteststring ) );
    TestR ( "result", fnvNull,
            FNV512result ( (FNV512context *)0, eUint512 ) );
    TestR ( "result", fnvNull,
            FNV512result ( &eContext, (uint8_t *)0  ) );
    TestR ( "result", fnvStateError,
            FNV512result ( &eContext, eUint512  ) );
if ( Terr )
    printf ( "%s test of error checks failed %i times.0,
             funcName, Terr );
else
    printf ( "%s test of error checks passed0, funcName );

/* test actual results */
Terr = 0;
/* tbd */
}    /* end Test512 */


/*****************************************************************
 * Code for FNV1024 using 64-bit integers
 *****************************************************************/

void Test1024 ()
{
//int            i, err;
uint8_t        eUint1024[FNV1024size];
FNV1024context  eContext;

funcName = "FNV1024";

/* test error checks */
Terr = 0;
TestR ( "init", fnvSuccess, FNV1024init (&eContext) );
    TestR ( "string", fnvNull,
            FNV1024string ( (char *)0, eUint1024 ) );
    TestR ( "string", fnvNull,
            FNV1024string ( errteststring, (uint8_t *)0 ) );
    TestR ( "block", fnvNull,
```

```
            FNV1024block ( (uint8_t *)0, 1, eUint1024 ) );
    TestR ( "block", fnvBadParam,
            FNV1024block ( errtestbytes, -1, eUint1024 ) );
    TestR ( "block", fnvNull,
            FNV1024block ( errtestbytes, 1, (uint8_t *)0 ) );
    TestR ( "init", fnvNull,
            FNV1024init ( (FNV1024context *)0 ) );
    TestR ( "initBasis", fnvNull,
            FNV1024initBasis ( (FNV1024context *)0, eUint1024 ) );
    TestR ( "blockin", fnvNull,
            FNV1024blockin ( (FNV1024context *)0,
                        errtestbytes, NTestBytes ) );
    TestR ( "blockin", fnvNull,
            FNV1024blockin ( &eContext, (uint8_t *)0,
                        NTestBytes ) );
    TestR ( "blockin", fnvBadParam,
            FNV1024blockin ( &eContext, errtestbytes, -1 ) );
    eContext.Computed = FNVclobber+FNV1024state;
    TestR ( "blockin", fnvStateError,
            FNV1024blockin ( &eContext, errtestbytes,
                        NTestBytes ) );
    TestR ( "stringin", fnvNull,
            FNV1024stringin ( (FNV1024context *)0, errteststring ) );
    TestR ( "stringin", fnvNull,
            FNV1024stringin ( &eContext, (char *)0 ) );
    TestR ( "stringin", fnvStateError,
            FNV1024stringin ( &eContext, errteststring ) );
    TestR ( "result", fnvNull,
            FNV1024result ( (FNV1024context *)0, eUint1024 ) );
    TestR ( "result", fnvNull,
            FNV1024result ( &eContext, (uint8_t *)0  ) );
    TestR ( "result", fnvStateError,
            FNV1024result ( &eContext, eUint1024  ) );
if ( Terr )
    printf ( "%s test of error checks failed %i times.0,
             funcName, Terr );
else
    printf ( "%s test of error checks passed0, funcName );

/* test actual results */
Terr = 0;
/* tbd */
}    /* end Test1024 */
   <CODE ENDS>
```

7. Security Considerations

This document is intended to provide convenient open source access by the Internet community to the FNV non-cryptographic hash. No assertion of suitability for cryptographic applications is made for the FNV hash algorithms.

7.1 Why is FNV Non-Cryptographic?

A full discussion of cryptographic hash requirements and strength is beyond the scope of this document. However, here are three characteristics of FNV that would generally be considered to make it non-cryptographic:

1. Sticky State - A cryptographic hash should not have a state in which it can stick for a plausible input pattern. But, in the very unlikely event that the FNV hash variable becomes zero and the input is a sequence of zeros, the hash variable will remain at zero until there is a non-zero input byte and the final hash value will be unaffected by the length of that sequence of zero input bytes. Of course, for the common case of fixed length input, this would usually not be significant because the number of non-zero bytes would vary inversely with the number of zero bytes and for some types of input, runs of zeros do not occur. Furthermore, the inclusion of even a little unpredictable input may be sufficient to stop an adversary from inducing a zero hash variable.

2. Diffusion - Every output bit of a cryptographic hash should be an equally complex function of every input bit. But it is easy to see that the least significant bit of a direct FNV hash is the XOR of the least significant bits of every input byte and does not depend on any other input bit. While more complex, the second through seventh least significant bits of an FNV hash have a similar weakness; only the top bit of the bottom byte of output, and higher order bits, depend on all input bits. If these properties are considered a problem, they can be easily fixed by XOR folding (see Section 3).

3. Work Factor - Depending on intended use, it is frequently desirable that a hash function should be computationally expensive for general purpose and graphics processors since these may be profusely available through elastic cloud services or botnets. This is to slow down testing of possible inputs if the output is known. But FNV is designed to be very inexpensive on a general-purpose processor. (See Appendix A.)

Nevertheless, none of the above have proven to be a problem in actual practice for the many applications of FNV.

7.2 Inducing Collisions

While use of a cryptographic hash should be considered when active adversaries are a factor, the following attack can be made much more difficult with very minor changes in the use of FNV.

If FNV is being used in a known way for hash tables in a network server or the like, for example some part of a web server, an adversary could send requests calculated to cause hash table collisions and induce substantial processing delays. As mentioned in Section 2.2, use of an offset_basis not knownable by the adversary will substantially eliminate this problem.

8. IANA Considerations

This document requires no IANA Actions. RFC Ediotor: Please delete this section before publication.

Normative References

[RFC20] - Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, October 1969, <http://www.rfc-editor.org/info/rfc20>.

Informative References

[FNV] - FNV web site:
http://www.isthe.com/chongo/tech/comp/fnv/index.html

[IEEE] - http://www.ieee.org

[IPv6flow] - https://researchspace.auckland.ac.nz/bitstream/handle/2292/13240/flowhashRep.pdf

[RFC2460] - Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998, <http://www.rfc-editor.org/info/rfc2460>.

[RFC3174] - Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC6194] - Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, March 2011.

[RFC6234] - Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011.

[RFC6437] - Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, November 2011, <http://www.rfc-editor.org/info/rfc6437>.

Acknowledgements

The contributions of the following are gratefully acknowledged:

Roman Donchenko, Frank Ellermann, Tony Finch, Bob Moskowitz, Gayle Noble, Stefan Santesson, and Mukund Sivaraman.

Appendix A: Work Comparison with SHA-1

This section provides a simplistic rough comparison of the level of effort required per input byte to compute FNV-1a and SHA-1 [RFC3174].

Ignoring transfer of control and conditional tests and equating all logical and arithmetic operations, FNV requires 2 operations per byte, an XOR and a multiply.

SHA-1 is a relatively weak cryptographic hash producing a 160-bit hash. It has been partially broken [RFC6194]. It is actually designed to accept a bit vector input although almost all computer uses apply it to an integer number of bytes. It processes blocks of 512 bits (64 bytes) and we estimate the effort involved in SHA-1 processing a full block. Ignoring SHA-1 initial set up, transfer of control, and conditional tests, but counting all logical and arithmetic operations, including counting indexing as an addition, SHA-1 requires 1,744 operations per 64 bytes block or 27.25 operations per byte. So by this rough measure, it is a little over 13 times the effort of FNV for large amounts of data. However, FNV is commonly used for small inputs. Using the above method, for inputs of N bytes, where N is <= 55 so SHA-1 will take one block (SHA-1 includes padding and an 8-byte length at the end of the data in the last block), the ratio of the effort for SHA-1 to the effort for FNV will be 872/N. For example, with an 8 byte input, SHA-1 will take 109 times as much effort as FNV.

Stronger cryptographic functions than SHA-1 generally have an even higher work factor.

Appendix B: Previous IETF Reference to FNV

FNV-1a was referenced in draft-ietf-tls-cached-info-08.txt that has since expired. It was later decided that it would be better to use a cryptographic hash for that application.

Below is the Java code for FNV64 from that TLS draft included by the kind permission of the author:

<CODE BEGINS>

```
 /**
 * Java code sample, implementing 64 bit FNV-1a
 * By Stefan Santesson
 */

import java.math.BigInteger;

public class FNV {


   static public BigInteger getFNV1aToByte(byte[] inp) {

       BigInteger m = new BigInteger("2").pow(64);
       BigInteger fnvPrime = new BigInteger("1099511628211");
       BigInteger fnvOffsetBasis =
               new BigInteger("14695981039346656037");

       BigInteger digest = fnvOffsetBasis;

       for (byte b : inp) {
           digest = digest.xor(BigInteger.valueOf((int) b & 255));
           digest = digest.multiply(fnvPrime).mod(m);
       }
       return digest;

   }
}
```

<CODE ENDS>

Appendix C: A Few Test Vectors

Below are a few test vectors in the form of ASCII strings and their FNV32 and FNV64 hashes using the FNV-1a algorithm.

Strings without null (zero byte) termination:

String	FNV32	FNV64
""	0x811c9dc5	0xcbf29ce484222325
"a"	0xe40c292c	0xaf63dc4c8601ec8c
"foobar"	0xbf9cf968	0x85944171f73967e8

Strings including null (zero byte) termination:

String	FNV32	FNV64
""	0x050c5d1f	0xaf63bd4c8601b7df
"a"	0x2b24d044	0x089be207b544f1e4
"foobar"	0x0c1c9eb8	0x34531ca7168b8f38

Appendix Z: Change Summary

RFC Editor Note: Please delete this appendix on publication.

From -00 to -01

1. Add Security Considerations section on why FNV is non-cryptographic.
2. Add Appendix A on a work factor comparison with SHA-1.
3. Add Appendix B concerning previous IETF draft referenced to FNV.
4. Minor editorial changes.

From -01 to -02

1. Correct FNV_Prime determination criteria and add note as to why s < 5 and s > 10 are not considered.
2. Add acknowledgements list.
3. Add a couple of references.
4. Minor editorial changes.

From -02 to -03

1. Replace direct reference to US-ASCII standard with reference to RFC 20.
2. Update dates and version number.
3. Minor editing changes.

From -03 to -04

1. Change reference to RFC 20 back to a reference to the ANSI 1968 ASCII standard.
2. Minor addition to Section 6, point 3.

3. Update dates and version number.

4. Minor editing changes.

From -04 to -05

1. Add Twitter as a use example and IPv6 flow hash study reference.

2. Update dates and version number.

From -05 to -06

1. Add code subsections.

2. Update dates and version number.

From -06 to -07 to -08

1. Update Author info.

2. Minor edits.

From -08 to -09

1. Change reference for ASCII to [RFC20].

2. Add more details on history of the string used to compute offset_basis.

3. Re-write "Work Factor" part of Section 6 to be more precise.

4. Minor editorial changes.

From -09 to -10

1. Inclusion of initial partial version of code and some documentation about the code, Section 6.

2. Insertion of new Section 4 on hashing values.

From -10 to -11

Changes based on code improvements primarily from Tony Hansen who has been added as an author. Changes based on comments from Mukund Sivaraman and Roman Donchenko.

From -11 to -12

Keep alive update.

From -12 to -13

Fixed bug in pseudocode in Section 2.3.

Change code to eliminate the BigEndian flag and so there are separate byte vector output routines for FNV32 and FNV64, equivalent to the other routines, and integer output routines for cases where Endianness consistency is not required.

From -13 to -14 to -15 to -16 to -17

Keep alive updates. Update an author address.

Author's Address

Glenn Fowler
Google

Email: glenn.s.fowler@gmail.com

Landon Curt Noll
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134 USA

Telephone: +1-408-424-1102
Email: http://www.isthe.com/chongo/address.html
URL: http://www.isthe.com/chongo/index.html

Kiem-Phong Vo
Google

Email: phongvo@gmail.com

Donald Eastlake
Huawei Technologies
1424 Pro Shop Court
Davenport, FL 33896 USA

Telephone: +1-508-333-2270
EMail: d3e3e3@gmail.com

Tony Hansen
AT&T Laboratories
200 Laurel Ave. South
Middletown, NJ 07748
USA

Email: tony@att.com

Internet Engineering Task Force
Internet-Draft
Intended status: Experimental
Expires: November 11, 2011

P. Hallam-Baker
Comodo Group Inc.
R. Stradling
Comodo CA Ltd.
B. Laurie
Google Inc.

May 10, 2011

DNS Certification Authority Authorization (CAA) Resource Record
draft-hallambaker-donotissue-04

Abstract

The Certification Authority Authorization (CAA) DNS Resource Record allows a DNS domain name holder to specify the certificate signing certificate(s) authorized to issue certificates for that domain. CAA resource records allow a public Certification Authority to implement additional controls to reduce the risk of unintended certificate mis-issue.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 11, 2011.

Copyright Notice

Table of Contents

1. Definitions

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Defined Terms

The following terms are used in this document:

Abstract Syntax Notation One (ASN.1) A notation for describing abstract types and values, as specified in X.680 [X.680].

Authorization Entry An authorization assertion that grants or denies a specific set of permissions to a specific group of entities.

Canonical Domain Name A Domain Name that is not an alias.

Canonical Domain Name Value The value of a Canonical Domain Name. The value resulting from applying alias transformations to a Domain Name that is not canonical.

Certificate An X.509 Certificate, as specified in RFC 5280 [RFC5280].

Certification Policy (CP) Specifies the criteria that a Certification Authority undertakes to meet in its issue of certificates.

Certification Practices Statement (CPS) Specifies the means by which the criteria of the Certification Policy are met. In most cases this will be the document against which the operations of the Certification Authority are audited.

Certification Authority (CA) An entity that issues Certificates in accordance with a specified Certification Policy.

Distinguished Encoding Rules (DER) A set of rules for encoding ASN.1 objects, as specified in X.690 [X.690].

Domain The set of resources associated with a DNS Domain Name.

Domain Name A DNS Domain name as specified in RFC 1035 [RFC1035] and revisions.

Domain Name System (DNS) The Internet naming system specified in RFC 1035 [RFC1035] and revisions.

DNS Security (DNSSEC) Extensions to the DNS that provide authentication services as specified in RFC 4033 [RFC4033] and revisions.

Extended Issuer Authorization Set The most specific Issuer Authorization Set that is active for a domain. This is either the Issuer Authorization Set for the domain itself, or if that is empty, the Issuer Authorization Set for the corresponding Public Delegation Point.

Issuer Authorization Set The set of Authorization Entries for a domain name that are flagged for use by Issuers. Analogous to an Access Control List but with no ordering specified.

Public Delegation Point A Domain Name that is obtained from a public DNS registry as defined by a Certification Policy.

Public Key Infrastructure X.509 (PKIX) Standards and specifications issued by the IETF that apply the X.509 [X.509] certificate standards specified by the ITU to Internet applications as specified in RFC 5280 [RFC5280] and related documents.

Resource Record (RR) A set of attributes bound to a Domain Name.

Relying Party A party that makes use of an application whose operation depends on use of a Certificate for making a security decision.

Relying Application An application whose operation depends on use of a Certificate for making a security decision.

Relying Party Authorization Set The set of Authorization Entries for a domain name that are flagged for use by Relying Party Applications. Analogous to an Access Control List but with no ordering specified.

2. Introduction

The Certification Authority Authorization (CAA) DNS Resource Record allows a DNS domain name holder to specify the Certification Authorities authorized to issue certificates for that domain. Publication of CAA resource records allow a public Certification Authority (CA) to implement additional controls to reduce the risk of unintended certificate mis-issue.

Conformance with a published CAA record is a necessary but not sufficient condition for issue of a certificate. Before issuing a certificate, a PKIX CA is required to validate the request according to the policies set out in its Certificate Policy Statement. In the case of a public CA that validates certificate requests as a third party, the certificate will be typically issued under a public root certificate embedded in one or more relevant reliant applications.

Criteria for inclusion of embedded root certificates in applications are outside the scope of this document but typically require the CA to publish a Certificate Practices Statement (CPS) that specifies how the requirements of the Certificate Policy (CP) are achieved and provide an annual audit statement of their performance against their CPS performed by an independent third party auditor.

It is the intention of the authors to propose the CAA record defined in this document as the basis for CA validation requirements to be proposed in organizations that publish validation requirements.

CAA records only describe the current state of Certification Authority certificate issue authority. Since a certificate is typically valid for at least a year, it is possible that a certificate that is not conformant with the CAA records currently published was conformant with the CAA records published at the time that it was issued. Thus Relying Applications MUST NOT use failure to conform to currently published CAA records as a rejection criteria for certificates unless the published records are flagged as being intended for that use.

2.1. The CAA RR type

A CAA RR (257) publishes a CAA property entry that corresponds to the specified domain name. Multiple property entries MAY be associated with the same domain name by publiching multiple CAA RRs at that domain name. Each property entry MAY be tagged with one or more of the following flag values:

Critical If set, indicates that the corresponding property entry tag MUST be understood if the semantics of the CAA record are to be correctly understood by the specified audience.

Issuers MUST NOT issue certificates for a domain if the Extended Issuer Authorization Set contains unknown property entry tags that are flagged as critical.

Relying Parties MUST NOT attempt to enforce CAA records if the Relying Party Authorization Set contains unknown property entry tags that are flagged as critical

Must be Zero This bit is reserved for future use.

Issuers MUST NOT issue certificates for a domain if the Extended Issuer Authorization Set contains property entries with the Must Be Zero Tag Set.

Relying Parties MUST NOT attempt to enforce CAA records if the Relying Party Authorization Set contains property entries with the Must Be Zero Tag Set.

Relying Party Specifies that the corresponding Property Entry is to be used by Relying Party Applications and forms part of the Relying Party Authorization Set for the domain.

Issuer Specifies that the corresponding Property Entry is to be used by Issuers and forms part of the Issuer Authorization Set for the domain.

The following properties are defined:

policy <Certificate Policy OID> The policy property entry declares an authorization entry granting authorization to issue under the specified Certificate Policy.

path <Object Digest Identifier> The path property entry declares an authorization entry granting authorization to issue end entity certificates under a trust path that includes the specified signing credential.

An Object Digest Identifier (ODI) is a means of specifying a reference to an object instance by means of a cryptographic digest function. A CAA path property may use an ODI to specify a certificate trust path by means of:

A Certificate Signing Certificate

A Public Signing Key

In either case a path Authorization Entry authorizes an issuer to issue an End Entity certificate to the corresponding domain if and only if it is possible to form a valid certificate path to it from the referenced certificate or key.

2.1.1. Examples of Use.

For convenience the examples are presented in the text format suggested in section Section 3.1.1

The following example informs CAs that certificates must not be issued except under the Default Deny Security 'Example 1' Certificate Policy (1.3.6.1.4.1.35405.666.1). Since the policy is published at the Public Delegation Point, the policy applies to all subordinate domains under example.com.

```
$ORIGIN example.com
.       CAA 1 policy 1.3.6.1.4.1.35405.666.1
```

The following example informs CAs that certificates must not be issued except under the Certificate Authority Root certificate specified in Appendix B.

```
$ORIGIN example.com
.       CAA 1 path MDIGA1UEJQYJYIZIAWUDBAIBBCAXzJgPaoT7Fe
                XaPzKv6mI2D0yilif+7WhzmhMGLe/oBA==
```

A domain MAY authorize multiple CAs to issue certificates at the same time. The following example allows issue under the Default Deny Security certification policy 'Example 1' or 'Example 2':

```
$ORIGIN example.com
.       CAA 1 policy 1.3.6.1.4.1.35405.666.1
.       CAA 1 policy 1.3.6.1.4.1.35405.666.2
```

If Authorization Entries using the path and policy properties are present at a given Domain, compatibility with either is sufficient to authorize the request.

Future versions of this specification MAY use the critical flag to introduce new semantics that MUST be understood for correct processing of the record, preventing Certification Authorities that do not recognize the record from issuing certificates.

In the following example, the property 'tbs' is flagged as critical. The Default Deny Security CA is not authorized to issue under either policy unless the processing rules for the 'tbs' property tag are understood.

```
$ORIGIN example.com
.       CAA 1 policy 1.3.6.1.4.1.35405.666.1
.       CAA 1 policy 1.3.6.1.4.1.35405.666.2
.       CAA 129 tbs MDIGA1UEJQYJYIZIAWUDBAIBBCAXzJgPaoT7Fe
```

XaPzKv6mI2D0yilif+7WhzmhMGLe/oBA==

Enforcement by Relying Party Applications follows the same general principles. A Relying Party Application MUST NOT enforce CAA records unless at least one Property Entry has the Relying Party flag set, that is the Relyin Party Authorization Set is not empty.

In the following example, certificates must not be issued except under the Default Deny Security 'Example 1' Certificate Policy and Relying Party Applications MAY reject certificates presented that do not comply with this requirement:

```
$ORIGIN example.com
.       CAA 3 policy 1.3.6.1.4.1.35405.666.1
```

In the ordinary course of business a Domain administrator may withdraw authorization for issue of new certificates before the previously issued certificates expire.

In the following example, Relying Party Applications are informed that certificates issued under either the policy are to be considered to be authorized but new certificates can only be issued under the first.

```
$ORIGIN example.com
.       CAA 3 policy 1.3.6.1.4.1.35405.666.1
.       CAA 2 policy 1.3.6.1.4.1.35405.666.2
```

2.2. Certification Authority Processing

Before issue of a certificate a compliant CA MUST check for publication of a relevant CAA Resource Record(s) and if such record(s) are published, that the certificate requested is consistent with them. If the certificate requested is not consistent with the relevant CAA RRs, the CA MUST NOT issue the certificate.

The Issuer Authorization Set for a domain name consists of the set of all CAA Authorization Entries declared for the canonical form of the specified domain.

The Extended Issuer Authorization Set for a domain name consists of the Issuer Authorization Set for that domain name if it is non-empty. Otherwise the Extended Issuer Authorization Set for a domain name consists of the Issuer Authorization Set for the corresponding Public Delegation Point for that domain name.

If the Extended Issuer Authorization Set for a domain name is not empty, a Certification Authority MUST NOT issue a certificate unless

it conforms to at least one authorization entry in the Extended Issuer Authorization Set.

Note that while it MUST be possible to form a certificate validation path that contains at least one certificate that is so specified, it MAY also be possible to form valid certificate paths that are not.

For example, a CA that has updated its root certificate to extend the expiry date is entitled to issue certificates for domains where the CAA record only specifies the older root certificate provided that the older root certificate has not actually expired and it is thus possible to form a valid certificate path.

2.2.1. Canonical Domain Name

The DNS defines the CNAME and DNAME mechanisms for specifying domain name aliases. The canonical name of a DNS name is the name that results from performing all DNS alias operations.

A Certification Authority MUST perform CNAME and DNAME processing as defined in the DNS specifications 1035 [RFC1035].

2.2.2. Use of DNS Security

Use of DNSSEC to authenticate CAA RRs is strongly recommended but not required. A CA MUST NOT issue certificates if doing so would conflict with the corresponding extended issuer authorization set whether the corresponding DNS records are signed or not.

Use of DNSSEC allows a CA to acquire and archive a non-repudiable proof that they were authorized to issue certificates for the domain.

2.2.3. Archive

A compliant CA SHOULD maintain an archive of the DNS transactions used to verify CAA eligibility.

In particular a CA SHOULD ensure that where DNSSEC data is available that the corresponding signature and NSEC/NSEC3 records are preserved so as to enable later compliance audits.

2.3. Relying Party Application Processing

Relying Party Applications MAY enforce CAA issue restrictions at their option, provided that the Relying Party Authorization set is not empty.

The consequences of determining that a certificate is not compatible

with the specified CAA relying party restrictions are outside the scope of this document.

Domains that opt to flag records for use by Relying Party Applications SHOULD be aware that the Property Entries supported in this version of the specification are only designed to support the requirements of enforcing issuer restrictions. While these Property Entries MAY be sufficient to enable enforcement by Relying Party Applications in some circumstances, they are not intended to provide complete requirements coverage for this purpose.

Domains containing CAA issue restrictions intended for use by Relying Party Applications SHOULD be authenticated using DNSSEC or other equivalent means.

If DNSSEC is deployed in a domain Relying Party Applications MUST treat failure to authenticate signatures of CAA records or absence of CAA records whose presence is indicated as being equivalent to an inconsistent CAA record.

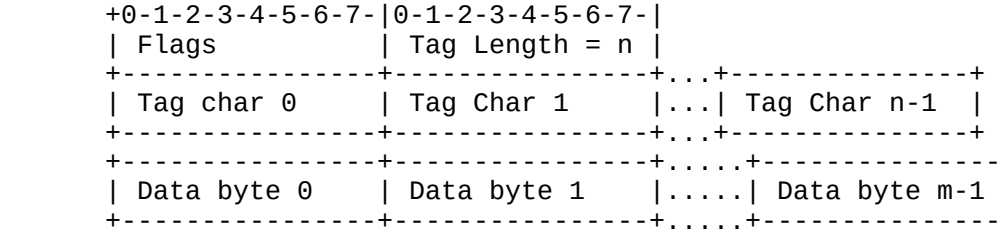
3. Mechanism

3.1. Syntax

A CAA RR contains a single property entry consisting of a tag value pair. Each tag represents a property of the CAA record. The value of a CAA property is that specified in the corresponding value field.

A domain name MAY have multiple CAA RRs associated with it and a given property MAY be specified more than once.

The CAA data field contains one property entry. A property entry consists of the following data fields:



Where n is the length specified in the tag length field and m is the remaining octets in the data field (m = d - n - 2) where d is the length of the data section.

The data fields are defined as follows:

Flags One octet containing the following fields:

Bit 0: Critical Flag If the value is set (1), the critical flag is asserted and the property MUST be understood if the CAA record is to be correctly processed.

A Certification Authority MUST NOT issue certificates for any Domain that contains a CAA critical property for an unknown or unsupported property type.

Bit 5: Must Be Zero Bit 5 is reserved and MUST be set to zero. Processors that encounter a CAA record containing a property with this bit set MUST treat the record set as if the critical property was asserted for an unknown record.

Bit 6: Relying Application Use If set, the property entry contains an Authorization Entry that forms part of the Relying Application Authorization Set for the corresponding domain.

Bit 7: Issuer Use If set, the property entry contains an Authorization Entry that forms part of the Issuer Application Authorization Set for the corresponding domain.

Note that according to the conventions set out in RFC 1035 [RFC1035] Bit 0 is the Most Significant Bit and Bit 7 is the Least Significant. Thus a flags value of 0x51 indicates a tag length of 5 octets and that the property entry is not critical and is not to be used for relying party processing.

Tag Length A single octet containing an unsigned integer specifying the tag length in octets. The tag length MUST be at least 1 and SHOULD be no more than 15.

Tag The property identifier, a sequence of ASCII characters.

Tag values MAY contain ASCII characters a through z and the numbers 0 through 9. Tag values MUST NOT contain any other characters. Matching of tag values is case insensitive.

Value A sequence of octets representing the property value. Property values are encoded as binary values and MAY employ sub-formats.

The length of the value field is specified implicitly as the remaining length of the enclosing Resource Record data field.

3.1.1. Canonical Presentation Format

The canonical presentation format of the CAA record is as follows:

CAA <flags> <tag> <data>

Where:

flags Is an unsigned integer between 0 and 15.

tag Is a non-zero sequence of ASCII letter and numbers in lower case.

data Is the Base64 Encoding [RFC4648] of the value field.

3.1.1.1. Policy OID Encoding Options

For convenience of administration, implementations MAY support ASN.1 Policy OID encoding at their option.

The Base64 encoding of data never contains the period character '.', while the encoding of ASN.1 OID values specified in IETF GSER encoding [RFC3642] will always incorporate at least one period character.

It follows that a data decoder MAY unambiguously interpret data specified in the Base64 or GSER format without the need for additional disambiguation.

Implementations MAY choose to allow use of both formats in both file and presentation formats.

3.1.2. policy Property value

The policy property value specifies an Authorization Entry by means of an ASN.1 OID specifying a Certification Policy. A Certification Authority is authorized to issue Certificates under a policy Authorization Entry if and only if

The Certification Authority has the right to issue certificates under the specified policy, AND

The certificate request is compliant with the requirements of the specified policy, AND

The certificate request meets all the criteria under the Certification Policy under which the certificate is to be issued.

Each policy property specifies a single ASN.1 OID value consisting of the ASN.1 type, length specifier and OID data.

The policy property applies to the specified policy OID and all policy OIDs that fall within the same OID arc. If the OID arc 1.3.6.1.4.1.35405.666 is specified, then the policy OIDs 1.3.6.1.4.1.35405.666, 1.3.6.1.4.1.35405.666.1, 1.3.6.1.4.1.35405.666.2 etc. are all authorized.

The Certificate that is issued MAY incorporate the specified policy OID itself but is not required to provided that the issue of the certificate is consistent with the requirements of the specified policy.

For example, a CA that offers two levels of Certification Policy such that the higher level of assurance included all the requirements of the lower one MAY rely on a policy property specifying the lower assurance policy as authorization for issue under the higher assurance policy but not vice-versa.

3.1.3. path Property value

The path property value specifies an Authorization Entry by means of a Certificate Signer Certificate or a Certificate Signing key. A Certification Authority is authorized to issue Certificates under a path Authorization Entry if and only if

A valid PKIX trust path can be formed from the specified Certificate Signer Certificate or a Certificate Signing key to the certificate that is to be issued, AND

The certificate request meets all the criteria under the Certification Policy under which the certificate is to be issued.

4. Security Considerations

CAA Records provide an accountability control. They are intended to deter rather than prevent undesired behavior.

While a Certification Authority can choose to ignore published CAA records, doing so increases the both the probability that they will mis-issue a certificate and the consequences of doing so. Once it is known that a CA observes CAA records, malicious registration requests will target disproportionately target the negligent CAs that do not,

and so the mis-issue rate amongst the negligent CAs will increase. Since the CA could clearly have avoided the mis-issue by performing CAA processing, the likelihood of sanctions against the negligent CA is increased. Failure to observe CAA issue restrictions provides an objective criteria for excluding issuers from embedded roots of trust.

In contrast, a Certification Authority that processes CAA records correctly can reasonably claim that any residual mis-issue event could have been avoided had the Domain Name holder published appropriate CAA records.

4.1. Mis-Issue by Authorized Certification Authority

Use of CAA records does not provide protection against mis-issue by an authorized Certification Authority.

Domain name holders SHOULD ensure that the CAs they authorize to issue certificates for their domains employ appropriate controls to ensure that certificates are only issued to authorized parties within their organization.

Such controls are most appropriately determined by the domain name holder and the authorized CA(s) directly and are thus out of scope of this document.

4.2. Suppression or spoofing of CAA records

Suppression of the CAA record or insertion of a bogus CAA record could enable an attacker to obtain a certificate from a CA that was not authorized to issue for that domain name.

4.2.1. Applications

Applications performing CAA checking SHOULD mitigate the risk of suppresion or spoofing of CAA records by means of DNSSEC validation where present. In cases where DNSSEC validation is not available, CAA checking is of limited security value.

4.2.2. Certification Authorities

Since a certificate issued by a CA can be valid for several years, the consequences of a spoofing or suppression attack are much greater for Certification Authorities and so additional countermeasures are justified.

A CA MUST mitigate this risk by employing DNSSEC verification whenever possible and rejecting certificate requests in any case

where it is not possible to verify the non-existence or contents of a relevant CAA record.

In cases where DNSSEC is not deployed in a corresponding domain, a CA SHOULD attempt to mitigate this risk by employing appropriate DNS security controls. For example all portions of the DNS lookup process SHOULD be performed against the authoritative name server. Cached data MUST NOT be relied on but MAY be used to support additional anti-spoofing or anti-suppression controls.

4.3. Denial of Service

Introduction of a malformed or malicious CAA RR could in theory enable a Denial of Service attack.

This specific threat is not considered to add significantly to the risk of running an insecure DNS service.

4.4. Abuse of the Critical Flag

A Certification Authority could make use of the critical flag to trick customers into publishing records which prevent competing Certification Authorities from issuing certificates even though the customer intends to authorize multiple providers.

In practice, such an attack would be of minimal effect since any competent competitor that found itself unable to issue certificates due to lack of support for a property marked critical is going to investigate the cause and report the reason to the customer who was deceived. It is thus unlikely that the attack would succeed and the attempt might lay the perpetrator open to civil or criminal sanctions.

5. IANA Considerations

5.1. Registration of the CAA Resource Record Type

IANA has assigned Resource Record Type 257 for the CAA Resource Record Type and added the line depicted below to the registry named Resource Record (RR) TYPEs and QTYPEs as defined in BCP 42 RFC 5395 [RFC5395] and located at http://www.iana.org/assignments/dns-parameters.

	Value and meaning	Reference
CAA	257 Certification Authority Restriction	[RFCXXXX]

5.2. Certification Authority Authorization Properties

IANA has created the Certification Authority Authorization Properties registry with the following initial values:

	Meaning	Reference
path	Authorization Entry by Signature Path	[RFCXXXX]
policy	Authorization Entry by Certificate Policy	[RFCXXXX]

Addition of tag identifiers requires a public specification and expert review as set out in RFC5395 [RFC5395]

6. References

6.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5395] Eastlake, D., "Domain Name System (DNS) IANA Considerations", RFC 5395, November 2008.

[X.509] International Telecommunication Union, "ITU-T Recommendation X.509 (11/2008): Information technology - Open systems interconnection - The Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, November 2008.

[X.680] International Telecommunication Union, "ITU-T Recommendation X.680 (11/2008): Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, November 2008.

[X.690] International Telecommunication Union, "ITU-T Recommendation X.690 (11/2008): Information technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, November 2008.

6.2. Non Normative References

[NIST-ALGS]
National Institute of Standards and Technology, "Cryptographic Algorithm Registration", March 2009.

[RFC3642] Legg, S., "Common Elements of Generic String Encoding Rules (GSER) Encodings", RFC 3642, October 2003.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

Appendix A. Object Digest Identifier Calculation

An Object Digest is an ASN.1 structure with three components:

An ASN.1 Object Identifier specifying the object type of the referenced object

An ASN.1 Object Identifier specifying the digest algorithm

An ASN.1 DER [X.690] encoded data field containing the digest value of the referenced object processed using the specified digest algorithm.

```
DNSCAA DEFINITIONS ::=

BEGIN

 ObjectDigestIdentifier ::= SEQUENCE {
   type             OBJECT IDENTIFIER,
   digestAlgorithm  OBJECT IDENTIFIER,
   digest           OCTET STRING
 }

END
```

The Object Digest Identifier construction is designed to facilitate implementation in applications that already require ASN.1 handling mechanisms (i.e. most cryptographic applications) without causing an undue coding burden in cases where ASN.1 code is not already supported. Appendix C provides all the necessary information to create a fully compliant Object Digest Identifier implementation.

A.1. Example: CA Certificate A

The ODI of CA Certificate A (specified in Appendix B.1) is calculated as follows:

ASN.1 Sequence tag: "3032"

ASN.1 OID id-at-cACertificate (2.5.4.37): "0603550425"

ASN.1 OID sha256 (2.16.840.1.101.3.4.2.1): "0609608648016503040201"

SHA-256 Digest Value: "042017cc980f6a84fb15e5da3f32afea62360f4ca29627feed68739a13062defe804"

The ODI in BASE64 format is MDIGA1UEJQYJYIZIAWUDBAIBBCAXzJgPaoT7FeXaPzKv6mI2D0yilif+7WhzmhMGLe/oBA==.

A.2. Example: CA Certificate A

The ODI of the signing key of CA Certificate A (specified in Appendix B.1) is calculated as follows:

ASN.1 Sequence tag

ASN.1 OID ’CA Signing Key’

ASN.1 OID ’SHA-256’

SHA-256 Digest Value

Appendix B. Example Certificates

The following certificates are used in the examples.

B.1. CA Certificate A

CA Certificate A is a self signed certificate signed with a 2048 bit RSA key:

```
-----BEGIN CERTIFICATE-----
MIIDATCCAeugAwIBAgIBATALBgkqhkiG9w0BAQUwKDERMA8GA1UEChMIQWNtZSBJ
bmMxEzARBgNVBAMTCkV4YW1wbGUgQ0EwHhcNMTAxMTExMTgxMjAzWhcNMjAxMTA4
MTgxMjAzWjAoMREwDwYDVQQKEwhBY21lIEluYzETMBEGA1UEAxMKRXhhbXBsZSBD
QTCCAR8wCwYJKoZIhvcNAQEBA4IBDgAwggEJAoIBALHvos3yEe0ugR6Ae2rPATXA
pBYGK6BMzGTLkXCg6MZaG9CZpfleZTZ/EgIKBwRJlIXvWdKwjMZ7GBByT+fdMDZp
7zkx64UZ4+CJm98NRjdugxovl8HhscIBXnhCHERgamp0U/f8Ho5W8eAxYLZ1XcIG
mB7mVknvolaN9EqlEmYn+qHexGJPlpWFmR4NKhVAATE6B1a9z5PCmoOgW9p0Vqic
SJ6CdAHKaa7JZS+sqNQDx57H8Q6R9lh52XXmJVVficxBp2K7C+Wvht45t68FG6f1
sXWuWDRYc6iUmOxZbzDDvIoFU0pAXESTdMOWvXKI8ZUaYBoZ7/YnSSTaseiW86sC
AwEAAaM9MDswDgYDVR0PAQEBBAQDAgAEMA8GA1UdEwEBAQQFMAMBAQEwGAYDVR0g
BBEwDzANBgsrBgEEAYKUTYUaATALBgkqhkiG9w0BAQUDggEBAGcNiaQXdyiI9Y5e
Ps+XEYdKiWYvmSnRIfbUZuQWaQpPcj5cHzMe91CUZipGDNJYXwqWhIUtQAAGmtrq
ZGa4F9Yh0cPFAHBXPHXKGeM1hMtAR7Mv9kHu4DFIhb822O0n4DdBIit8FNas5t/5
CbM6crDpWB5hjAsD37U+GZGvTJmag059VWjnjv90NcfCQ6YJ6AA5VKnmrV695VnL
dSPaN9VS5RN6heJqU9tcbqPkAEP3MuJtd1QxB8Q34f9e1kTYXxc/dBJK1RQ0F4nc
Jc4NbJzakvFq+QcbzEqkhDMiXvjDV0JJt+GkFZrsREi6IgQY4DQHPv65OIvbr3uW
329dd+g=
-----END CERTIFICATE-----
```

In binary form, the certificate data is:

```
0000  30 82 03 01 30 82 01 eb  a0 03 02 01 02 02 01 01
0010  30 0b 06 09 2a 86 48 86  f7 0d 01 01 05 30 28 31
0020  11 30 0f 06 03 55 04 0a  13 08 41 63 6d 65 20 49
0030  6e 63 31 13 30 11 06 03  55 04 03 13 0a 45 78 61
0040  6d 70 6c 65 20 43 41 30  1e 17 0d 31 30 31 31 31
0050  31 31 38 31 32 30 33 5a  17 0d 32 30 31 31 30 38
0060  31 38 31 32 30 33 5a 30  28 31 11 30 0f 06 03 55
0070  04 0a 13 08 41 63 6d 65  20 49 6e 63 31 13 30 11
0080  06 03 55 04 03 13 0a 45  78 61 6d 70 6c 65 20 43
0090  41 30 82 01 1f 30 0b 06  09 2a 86 48 86 f7 0d 01
00a0  01 01 03 82 01 0e 00 30  82 01 09 02 82 01 00 b1
00b0  ef a2 cd f2 11 ed 2e 81  1e 80 7b 6a cf 01 35 c0
00c0  a4 16 06 2b a0 4c cc 64  cb 91 70 a0 e8 c6 5a 1b
00d0  d0 99 a5 f9 5e 65 36 7f  12 02 0a 07 04 49 94 85
```

```
00e0  ef 59 d2 b0 8c c6 7b 18  10 72 4f e7 dd 30 36 69
00f0  ef 39 31 eb 85 19 e3 e0  89 9b df 0d 46 37 6e 83
0100  1a 2f 97 c1 e1 b1 c2 01  5e 78 42 1c 44 60 6a 6a
0110  74 53 f7 fc 1e 8e 56 f1  e0 31 60 b6 75 5d c2 06
0120  98 1e e6 56 49 ef a2 56  8d f4 4a a5 12 66 27 fa
0130  a1 de c4 62 4f 96 95 85  99 1e 0d 2a 15 40 01 31
0140  3a 07 56 bd cf 93 c2 9a  83 a0 5b da 74 56 a8 9c
0150  48 9e 82 74 01 ca 69 ae  c9 65 2f ac a8 d4 03 c7
0160  9e c7 f1 0e 91 f6 58 79  d9 75 e6 25 55 5f 89 cc
0170  41 a7 62 bb 0b e5 af 86  de 39 b7 af 05 1b a7 f5
0180  b1 75 ae 58 34 58 73 a8  94 98 ec 59 6f 30 c3 bc
0190  8a 05 53 4a 40 5c 44 93  74 c3 96 bd 72 88 f1 95
01a0  1a 60 1a 19 ef f6 27 49  24 da b1 e8 96 f3 ab 02
01b0  03 01 00 01 a3 3d 30 3b  30 0e 06 03 55 1d 0f 01
01c0  01 01 04 04 03 02 00 04  30 0f 06 03 55 1d 13 01
01d0  01 01 04 05 30 03 01 01  01 30 18 06 03 55 1d 20
01e0  04 11 30 0f 30 0d 06 0b  2b 06 01 04 01 82 94 4d
01f0  85 1a 01 30 0b 06 09 2a  86 48 86 f7 0d 01 01 05
0200  03 82 01 01 00 67 0d 89  a4 17 77 28 88 f5 8e 5e
0210  3e cf 97 11 87 4a 89 66  2f 99 29 d1 21 f6 d4 66
0220  e4 16 69 0a 4f 72 3e 5c  1f 33 1e f7 50 94 66 2a
0230  46 0c d2 58 5f 0a 96 84  85 2d 40 00 06 9a da ea
0240  64 66 b8 17 d6 21 d1 c3  c5 00 70 57 3c 75 ca 19
0250  e3 35 84 cb 40 47 b3 2f  f6 41 ee e0 31 48 85 bf
0260  36 d8 ed 27 e0 37 41 22  2b 7c 14 d6 ac e6 df f9
0270  09 b3 3a 72 b0 e9 58 1e  61 8c 0b 03 df b5 3e 19
0280  91 af 4c 99 9a 83 4e 7d  55 68 e7 8e ff 74 35 c7
0290  c2 43 a6 09 e8 00 39 54  a9 e6 ad 5e bd e5 59 cb
02a0  75 23 da 37 d5 52 e5 13  7a 85 e2 6a 53 db 5c 6e
02b0  a3 e4 00 43 f7 32 e2 6d  77 54 31 07 c4 37 e1 ff
02c0  5e d6 44 d8 5f 17 3f 74  12 4a d5 14 34 17 89 dc
02d0  25 ce 0d 6c 9c da 92 f1  6a f9 07 1b cc 4a a4 84
02e0  33 22 5e f8 c3 57 42 49  b7 e1 a4 15 9a ec 44 48
02f0  ba 22 04 18 e0 34 07 3e  fe b9 38 8b db af 7b 96
0300  df 6f 5d 77 e8
```

The SHA-256 digest of the certificate data is:

17cc980f6a84fb15e5da3f32afea62360f4ca29627feed68739a13062defe804

Appendix C. ASN.1 Values (Non-Normative)

Although the Object Digest Identifier form employs ASN.1 DER encoding only a small subset of ASN.1 features are used and a full ASN.1 stack is not necessary.

This appendix provides sufficient information to implement an Object

Digest Identifier constructor or parser.

C.1. DER Sequence Encoding

In DER encoding, the enclosing SEQUENCE will always be represented by the type identifier x30 followed by the length specifier. Since the total length of the following data fields will almost certainly be less than 127 bytes, the single byte encoding mechanism in which bit 7 is clear and the length value is encoded in the lower 7 bits will be required.

C.2. Object Identifiers for Certificate Types

OIDs have been defined in connection with the X.500 directory for user certificates, certification authority certificates, revocations of certification authority, and revocations of user certificates. The following table lists the OIDs, their DER encoding, and their type identifier and length-prefixed hex format for use in Object Digest Identifiers.

```
id-at  OBJECT IDENTIFIER  ::=  { joint-iso-itu-t(2) ds(5) 4 }

id-at-userCertificate  OBJECT IDENTIFIER  ::=  { id-at 36 }
                                                 -- 06 03 55 04 24
  id-at-cACertificate  OBJECT IDENTIFIER  ::=  { id-at 37 }
                                                 -- 06 03 55 04 25
 TBS-PUBLIC-KEY-VALUE  OBJECT IDENTIFIER  ::=  { ??? }
                                                 -- 06 xx xx xx xx
```

C.3. Object Identifiers for Digest Algorithms

OIDs have been assigned by NIST for the SHA-2 digest algorithms [NIST-ALGS] [RFC4055] Use of the SHA-1 digest algorithm is not recommended due to concerns for the security of the algorithm.

```
hashAlgs  OBJECT IDENTIFIER  ::=  { joint-iso-itu-t(2)
              country(16) us(840) organization(1) gov(101) csor(3)
              nistAlgorithm(4) 2 }

id-sha256  OBJECT IDENTIFIER  ::=  { hashAlgs 1 }
                                     -- 06 09 60 86 48 01 65 03 04 02 01
id-sha384  OBJECT IDENTIFIER  ::=  { hashAlgs 2 }
                                     -- 06 09 60 86 48 01 65 03 04 02 02
id-sha512  OBJECT IDENTIFIER  ::=  { hashAlgs 3 }
                                     -- 06 09 60 86 48 01 65 03 04 02 03
id-sha224  OBJECT IDENTIFIER  ::=  { hashAlgs 4 }
                                     -- 06 09 60 86 48 01 65 03 04 02 04
```

C.4. DER Data Encoding Prefixes

The rules of ASN.1 encoding state that every data value is preceded by a data type identifier and a length identifier. In the case of an Object Digest Identifier the data type identifier is always OCTET STRING (04) and the length for all currently defined digest algorithms will be less than 128 bytes (1024 bits) and thus use the single byte encoding form in which bit 7 is set to 0 and the lower 7 bits specify the length.

The length prefixes for commonly used digest lengths in hexadecimal notation are thus:

160 bits 04 14

224 bits 04 1C

256 bits 04 20

384 bits 04 30

512 bits 04 40

Authors' Addresses

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Rob Stradling
Comodo CA Ltd.

Email: rob.stradling@comodo.com

Ben Laurie
Google Inc.

Email: benl@google.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: December 6, 2012

Y. Oiwa
H. Watanabe
H. Takagi
RISEC, AIST
B. Kihara
T. Hayashi
Lepidum
Y. Ioku
Yahoo! Japan
June 4, 2012

Mutual Authentication Protocol for HTTP
draft-oiwa-http-mutualauth-12

Abstract

This document specifies a mutual authentication method for the Hypertext Transport Protocol (HTTP). This method provides a true mutual authentication between an HTTP client and an HTTP server using password-based authentication. Unlike the Basic and Digest authentication methods, the Mutual authentication method specified in this document assures the user that the server truly knows the user's encrypted password. This prevents common phishing attacks: a phishing attacker controlling a fake website cannot convince a user that he authenticated to the genuine website. Furthermore, even when a user authenticates to an illegitimate server, the server cannot gain any information about the user's password. The Mutual authentication method is designed as an extension to the HTTP protocol, and is intended to replace the existing authentication methods used in HTTP (the Basic method, Digest method, and authentication using HTML forms).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 6, 2012.

Copyright Notice

Table of Contents

1. Introduction

This document specifies a mutual authentication method for Hyper-Text Transport Protocol (HTTP). The method, called "Mutual Authentication Protocol" in this document, provides a true mutual authentication between an HTTP client and an HTTP server, using just a simple password as a credential.

The currently available methods for authentication in HTTP and Web systems have several deficiencies. The Basic authentication method [RFC2617] sends a plaintext password to a server without any protection; the Digest method uses a hash function that suffers from simple dictionary-based off-line attacks, and people have begun to think it is obsolete.

The authentication method proposed in this document solves these problems, substitutes for these existing methods, and serves as a long-term solution to Web authentication security. It has the following main characteristics:

- It provides "true" mutual authentication: in addition to assuring the server that the user knows the password, it also assures the user that the server truly knows the user's encrypted password at the same time. This makes it impossible for fake website owners to persuade users that they have authenticated with the original websites.

- It uses only passwords as the user's credential: unlike public-key-based security algorithms, the method does not rely on secret keys or other cryptographic data that have to be stored inside the users' computers. The proposed method can be used as a drop-in replacement to the current authentication methods like Basic or Digest, while ensuring a much stronger level of security.

- It is secure: when the server fails to authenticate with a user, the protocol will not reveal any bit of the user's password.

Users can discriminate between true and fake Web servers using their own passwords by using the proposed method. Even when a user inputs his/her password to a fake website owned by illegitimate phishers, the user will certainly notice that the authentication has failed. Phishers will not be successful in their authentication attempts, even if they forward the received data from a user to a legitimate server or vice versa. Users can input sensitive data to the web forms after confirming that the mutual authentication has succeeded, without fear of phishing attacks.

The document, along with [I-D.oiwa-http-auth-extension], also

proposes several extensions to the current HTTP authentication framework, to replace current widely-used form-based Web authentication. The extensions provided include:

- Multi-host single authentication within an Internet domain (Section 5),
- non-mandatory, optional authentication on HTTP (Section 8),
- log out from both server and client side (Section 8), and
- finer control for redirection depending on authentication status (Section 8).

1.1. Relations to other technologies

1.1.1. Technologies updated or superceded by this proposal

1.1.1.1. HTTP Basic and Digest authentication

The main purpose of this proposal is obviously providing an upgrade for the two existing HTTP authentication methods, Basic and Digest [RFC2617].

HTTP Basic authentication, as its name suggests, provides very simple authentication mechanism using plain-text password directly upon the HTTP transport. HTTP Digest authentication focuses on mitigating the fundamental weakness of Basic authentication by using MD5-based hashing to the authentication, but that has almost failed to deploy due to improper implementations, interoperability problems, and missing feature implementations before MD5 has deprecated by its cryptographic weakness. Digest also has a fundamental problem that the server-side must posses a password-equivalent to perform authentication, which increases risks of server-side data leakage.

1.1.1.2. HTML Form authentication

Another aim of this protocol is (at least) partially replacing the HTML form authentication. Because of inflexibility of the HTTP Basic authentication, recent Web applications tend to use application-level implementations for user authentication using HTML Forms and Web browser rendering engines. However, that method has many potential security weaknesses as same as the HTTP Basic authentication as it uses plaintext. Considering server-impersonations and existence of human-forging rogue servers (i.e. phishing), script-based implementations of hash-based authentication does not help, because its behavior is completely controlled by the web-page content itself, which is possibly provided by such a rogue server. This also closes

any possibilities for extending HTML forms to implement cryptography, as its user-interface could not be prevented from being imitated using plain-text forms. Using HTTP-level authentication is better in this field, because it is under the control of the client software (Web browsers), which can enforce security checks regardless of server-provided contents.

Of course, we could not ignore the strong reasons of favoring Form authentication over Basic authentication: its flexibility. HTTP authentication framework lacks many features for recent Web applications, mainly for interactions between HTTP-level authentications and application-level management of "authentication sessions". As long as current HTTP-layer (and lower-layer) authentication are used, the new method would share the same problem. To solve this problem, this protocol has a companion mechanism for application-level control of authentication behaviors as a separate draft [I-D.oiwa-http-auth-extension]. By using this additional mechanism, Web applications can implement most of these required features as easy as just calling an already-provided API for them.

1.1.2. Technologies not updated by this proposal

1.1.2.1. Federated identity/authorization management

There are several technologies (protocols, frameworks, or systems) for managing authentications/authorizations involving multiple-parties: some of those examples are OAuth [I-D.ietf-oauth-v2], OpenID Connect [OIDF.Connect.Standard], SAML [OASIS.saml-core-2.0-os] etc. These technologies can be further divided to two categories: federated authentication and authorization delegation, although some of these technologies cover both.

Federated authentication provides so-called "three-legged authentication": provided the result of user authentication to a single entity (identity provider) and the user’s consent, the mechanism can provide other entities assertion of the user’s identity without performing a separate identity management by every entity. Authorization delegation gives a mechanism for transferring a part of the user’s privilege on an entity (resource owners) to another entity without requiring users give away the full credential for the authentication.

Essentially, both of those technologies are transforming a result of conventional, one-by-one (two-legged) authentication into a multi-party privilege management. The purpose of this protocol is to secure the very part of the two-legged authentication, and so it can be naturally combined with existing federated management frameworks for increasing security of the entire system.

Additionally, this protocol can provide a secure peer-to-peer shared key generated during authentication to the higher-layer applications Section 12. These keys can be possibly used by such federating mechanisms in future for simplifying/securing the framework.

1.1.2.2. HTTPS and HTTPS client-certificate authentication

This protocol will not replace the wide-spread and widely-accepted technology of SSL/TLS and HTTPS [RFC2818]. This protocol will be still relying on the HTTPS for the integrity and secrecy of the HTTP payload. This protocol ensures users the integrity and secrecy of the authentication credentials, and authenticity of the talking peer server.

Client certificate (and other public-key-based) authentications have a fair-amount of applications (mainly for high-assurance applications), and there are possible needs for redesigning/updating the whole framework. However, currently public-key-based user-authentication and connection-based user identification is out-of-scope of this proposal.

1.1.2.3. Protocols for local identity-management frameworks

There are several existing frameworks for managing user identity of tightly-managed, closed group of users, such as Kerberos [RFC3961]/ GSS-API [RFC2743] etc. Some of these have defined a bridging protocol for HTTP authentication. This protocol does not currently aim to replace such existing frameworks.

More precisely, requirements for those framework and usual Web user authentication differ fundamentally. In such framework, user authentication in performed first, and the result of the authentication tends to be shared in all applications, sometimes even shared regardless of the underlying protocols. In those systems, it is almost never likely to use a multiple identity to be used inside a single server and inside a single client machine at the same time. In such applications, connection-based or even a machine-based authentication can be used without a trouble. This is not a case for the general Web authentication applications.

1.1.2.4. HTTP and HTTP authentication architecture

Although HTTP and generic HTTP authentication architecture lacks some required features (see above), the whole structure of per-request, per-resource authentication is well-suited for general Web applications compared with connection-based or machine-based authentication/authorization framework (those which tie user identity to either connections or machines). The whole protocol in this

specification is designed on top of the framework of [I-D.ietf-httpbis-p7-auth]. Small extensions to the framework in this specification and [I-D.oiwa-http-auth-extension], which are designed for filling the missing features, are carefully designed so that it can be implemented easily only by the client-side without changing the whole framework.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The terms "encouraged" and "advised" are used for suggestions that do not constitute "SHOULD"-level requirements. People MAY freely choose not to include the suggested items regarding [RFC2119], but complying with those suggestions would be a best practice; it will improve security, interoperability, and/or operational performance.

This document distinguishes the terms "client" and "user" in the following way: A "client" is an entity understanding and talking HTTP and the specified authentication protocol, usually computer software; a "user" is a (usually natural) person who wants to access data resources using "a client".

The term "natural numbers" refers to the non-negative integers (including zero) throughout this document.

This document treats target (codomain) of hash functions to be natural numbers. The notation OCTETS(H(s)) gives a usual octet-string output of hash function H applied to string s.

1.3. Document Structure and Related Documents

The entire document is organized as follows:

- Section 2 presents an overview of the protocol design.
- Sections 3 to 10 define a general framework of the Mutual authentication protocol. This framework is independent of specific cryptographic primitives.
- Section 11 describes properties needed for cryptographic algorithms used with this protocol framework, and defines a few functions which will be shared among such cryptographic algorithms.

- The sections after that contain general normative and informative information about the protocol.
- The appendices contain some information that may help developers to implement the protocol.

In addition, there are two companion documents which are referred from/related to this specification:

- [I-D.oiwa-http-mutualauth-algo]: defines a cryptographic primitives which can be used with this protocol framework. [draft note: it is separated so that it may be replaced with another crypto in future. We need at least one example for testing/ implementing this protocol, so here it is.]
- [I-D.oiwa-http-auth-extension]: defines a small but useful extensions to the current HTTP authentication framework so that it can support application-level semantics of existing Web systems.

2. Protocol Overview

The protocol, as a whole, is designed as a natural extension to the HTTP protocol [I-D.ietf-httpbis-p1-messaging] using a framework defined in [I-D.ietf-httpbis-p7-auth]. Internally, the server and the client will first perform a cryptographic key exchange, using the secret password as a "tweak" to the exchange. The key-exchange will only succeed when the secrets used by the both peers are correctly related (i.e. generated from the same password). Then, both peers will verify the authentication results by confirming the sharing of the exchanged key. This section describes a brief image of the protocol and the exchanged messages.

2.1. Messages Overview

The authentication protocol uses seven kinds of messages to perform mutual authentication. These messages have specific names within this specification.

- Authentication request messages: used by the servers to request clients to start mutual authentication.
  - 401-INIT message: a general message to start the authentication protocol. It is also used as a message indicating an authentication failure.
  - 200-Optional-INIT message: a variant of the 401-INIT message indicating that an authentication is not mandatory.

    * 401-STALE message: a message indicating that it has to start a new authentication trial.

- Authenticated key exchange messages: used by both peers to perform authentication and the sharing of a cryptographic secret.

    * req-KEX-C1 message: a message sent from the client.

    * 401-KEX-S1 message: a message sent from the server as a response to a req-KEX-C1 message.

- Authentication verification messages: used by both peers to verify the authentication results.

    * req-VFY-C message: a message used by the client, requesting that the server authenticates and authorizes the client.

    * 200-VFY-S message: a successful response used by the server, and also asserting that the server is authentic to the client simultaneously.

In addition to the above, either a request or a response without any HTTP headers related to this specification will be hereafter called a "normal request" or a "normal response", respectively.

2.2. Typical Flows of the Protocol

In typical cases, the client access to a resource protected by the Mutual authentication will follow the following protocol sequence.

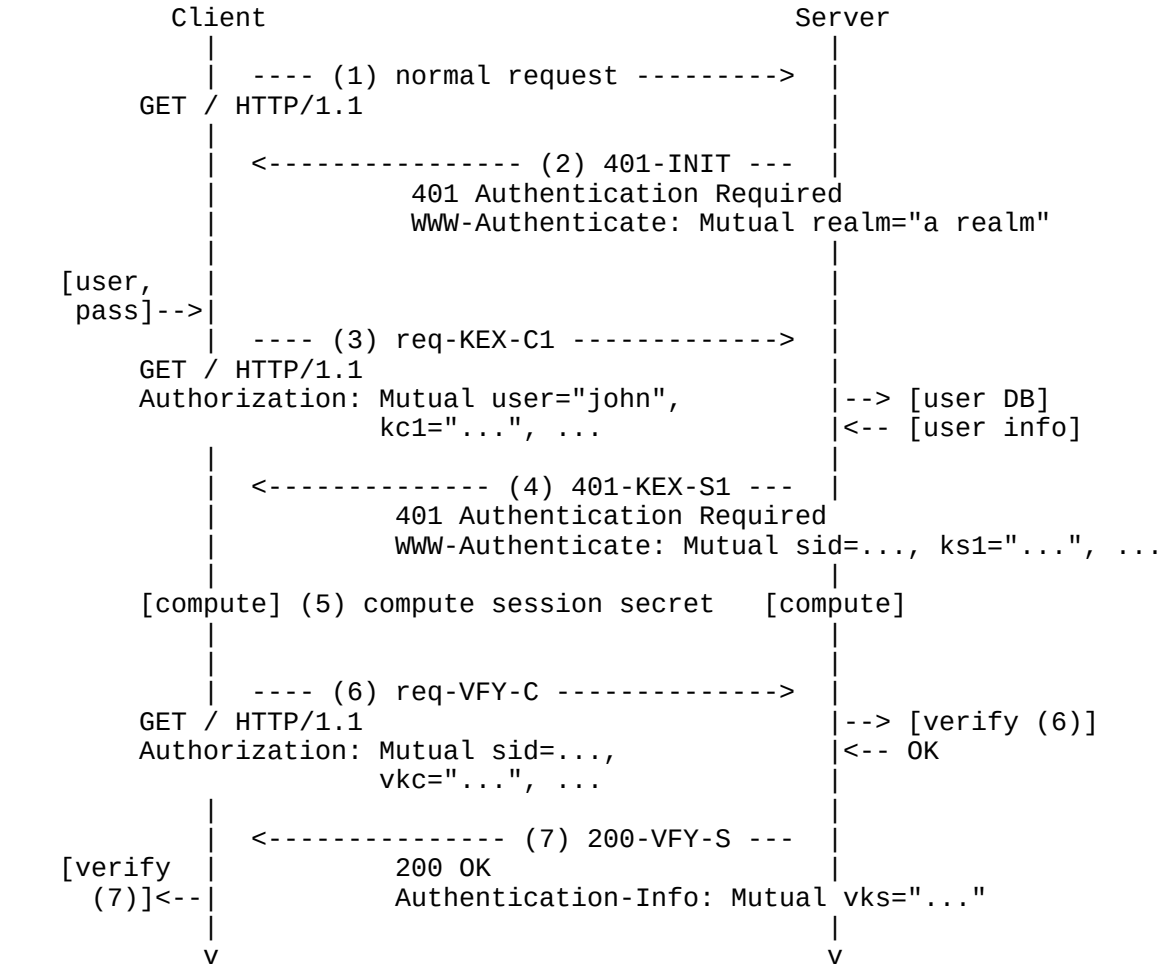


Figure 1: Typical communication flow for first access to resource

- As usual in general HTTP protocol designs, a client will at first request a resource without any authentication attempt (1). If the requested resource is protected by the Mutual authentication, the server will respond with a message requesting authentication (401-INIT) (2).

- The client processes the body of the message, and waits for the user to input the user name and a password. If the user name and the password are available, the client will send a message with the authenticated key exchange (req-KEX-C1) to start the authentication (3).

- If the server has received a req-KEX-C1 message, the server looks up the user's authentication information within its user database. Then the server creates a new session identifier (sid) that will be used to identify sets of the messages that follow it, and responds back with a message containing a server-side authenticated key exchange value (401-KEX-S1) (4).

- At this point (5), both peers calculate a shared "session secret" using the exchanged values in the key exchange messages. Only when both the server and the client have used secret credentials generated from the same password will the session secret values match. This session secret will be used for the actual access authentication after this point.

- The client will send a request with a client-side authentication verification value (req-VFY-C) (6), generated from the client-owned session secret. The server will check the validity of the verification value using its own session secret.

- If the authentication verification value from the client was correct, it means that the client definitely owns the credential based on the expected password (i.e. the client authentication succeeded.) The server will respond with a successful message (200-VFY-S) (7). Contrary to the usual one-way authentication (e.g. HTTP Basic authentication or POP APOP authentication), this message also contains a server-side authentication verification value.

  When the client's verification value is incorrect (e.g. because the user-supplied password was incorrect), the server will respond with the 401-INIT message (the same one as used in (2)) instead.

- The client MUST first check the validity of the server-side authentication verification value contained in the message (7). If the value was equal to the expected one, the server authentication succeeded.

  If it is not the value expected, or if the message does not contain the authentication verification value, it means that the mutual authentication has been broken for some unexpected reason. The client MUST NOT process any body or header values contained in this case. (Note: This case should not happen between a correctly-implemented server and a client.)

2.3. Alternative Flows

As shown above, the typical flow for a first authenticated request requires three request-response pairs. To reduce the protocol overhead, the protocol enables several short-cut flows which require fewer messages.

- (case A) If the client knows that the resource is likely to require the authentication, the client MAY omit the first unauthenticated request (1) and immediately send a key exchange (req-KEX-C1 message). This will reduce one round-trip of messages.

- (case B) If both the client and the server previously shared a session secret associated with a valid session identifier (sid), the client MAY directly send a req-VFY-C message using the existing session identifier and corresponding session secret. This will further reduce one round-trip of messages.

  In such cases, the server MAY have thrown out the corresponding sessions from the session table. In this case, the server will respond with a 401-STALE message, indicating a new key exchange is required. The client SHOULD retry constructing a req-KEX-C1 message in this case.

Figure 2 depicts the shortcut flows described above. Under the appropriate settings and implementations, most of the requests to resources are expected to meet both the criteria, and thus only one round-trip of request/responses will be required in most cases.

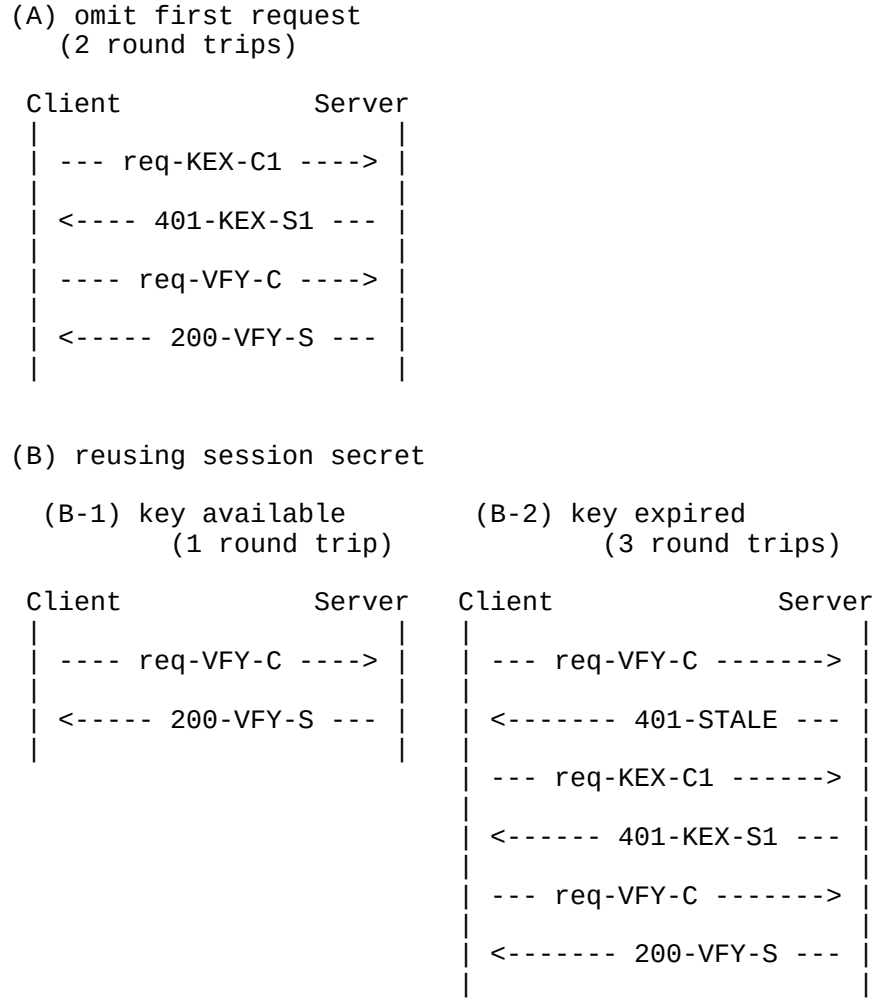


Figure 2: Several alternative flows on protocol

For more details, see Sections 9 and 10.

3. Message Syntax

Throughout this specification, The syntax is denoted in the extended augmented BNF syntax defined in [I-D.ietf-httpbis-p1-messaging] and [RFC5234]. The following elements are quoted from [RFC5234], [I-D.ietf-httpbis-p1-messaging] and [I-D.ietf-httpbis-p7-auth]: DIGIT, ALPHA, SP, auth-scheme, quoted-string, auth-param, header-field, token, challenge, and credential.

The Mutual authentication protocol uses three headers: WWW-Authenticate (in responses with status code 401), Authorization (in requests), and Authentication-Info (in responses other than 401 status). These headers follow a common framework described in [I-D.ietf-httpbis-p7-auth]. The detailed meanings for these headers are contained in Section 4.

The framework in [I-D.ietf-httpbis-p7-auth] defines the syntax for the headers WWW-Authenticate and Authorization as the syntax elements "challenge" and "credentials", respectively. The "auth-scheme" contained in those headers MUST be "Mutual" throughout this protocol specification. The syntax for "challenge" and "credentials" to be used with the "Mutual" auth-scheme SHALL be name-value pairs (#auth-param), not the "b64token" defined in [I-D.ietf-httpbis-p7-auth].

The Authentication-Info: header used in this protocol SHALL contain the value in same syntax as those the "WWW-Authenticate" header, i.e. the "challenge" syntax element.

In HTTP, the WWW-Authenticate header may contain more than one challenges. Client implementations SHOULD be aware of and be capable of handle those cases correctly.

3.1. Values

The parameter values contained in challenge/credentials MUST be parsed strictly conforming to the HTTP semantics (especially un-quoting of the string parameter values). In this protocol, those values are further categorized into the following value types: tokens (bare-token and extensive-token), string, integer, hex-fixed-number, and base64-fixed-number.

For clarity, implementations are encouraged to use the canonical representations specified in the following subsections for sending values. Recipients SHOULD accept both quoted and unquoted representations interchangeably as specified in HTTP.

3.1.1. Tokens

For sustaining both security and extensibility at the same time, this protocol defines a stricter sub-syntax for the "token" to be used. The extensive-token values SHOULD follow the following syntax (after HTTP value parsing):

```
 bare-token       = 1*(DIGIT / ALPHA / "-" / "_")
 extension-token  = "-" bare-token 1*("." bare-token)
 extensive-token  = bare-token / extension-token
```

Figure 3: BNF syntax for token values

The tokens (bare-token and extension-token) are case insensitive; Senders SHOULD send these in lower-case, and receivers MUST accept both upper- and lower-cases. When tokens are used as (partial) inputs to any hash or other mathematical functions, it MUST always be used in lower-case.

Extensive-tokens are used in this protocol where the set of acceptable tokens may include non-standard extensions. Any non-standard extensions of this protocol SHOULD use the extension-tokens with format "-<bare-token>.<domain-name>", where <domain-name> is a validly registered (sub-)domain name on the Internet owned by the party who defines the extensions.

Bare-tokens and extensive-tokens are also used for parameter names (of course in the unquoted form). Requirements for using the extension-token for the parameter names are the same as the above.

The canonical format for bare-tokens and tokens are unquoted tokens.

3.1.2. Strings

All character strings outside ASCII character sets MUST be encoded using the UTF-8 encoding [RFC3629] for the ISO 10646-1 character set [ISO.10646-1.1993], without any leading BOM characters. Both peers are RECOMMENDED to reject any invalid UTF-8 sequences that might cause decoding ambiguities (e.g., containing <"> in the second or later byte of the UTF-8 encoded characters).

If strings are representing a domain name or URI that contains non-ASCII characters, the host parts SHOULD be encoded as it is used in the HTTP protocol layer (e.g. in a Host: header); under current standards it will be the one defined in [RFC5890]. It SHOULD use lower-case ASCII characters.

The canonical format for strings are quoted-string.

3.1.3. Numbers

The following syntax definitions gives a syntax for number-type values:

```
 integer          = "0" / (%x31-39 *DIGIT)      ; no leading zeros
 hex-fixed-number = 1*(2(DIGIT / %x41-46 / %x61-66))
 base64-fixed-number = 1*( ALPHA / DIGIT /
                         "-" / "." / "_" / "~" / "+" / "/" ) *"="
```

Figure 4: BNF syntax for number types

The syntax definition of the integers only allows representations that do not contain extra leading zeros.

The numbers represented as a hex-fixed-number MUST include an even number of characters (i.e. multiples of eight bits). Those values are case-insensitive, and SHOULD be sent in lower-case. When these values are generated from any cryptographic values, they SHOULD have their "natural length": if these are generated from a hash function, these lengths SHOULD correspond to the hash size; if these are representing elements of a mathematical set (or group), its lengths SHOULD be the shortest for representing all the elements in the set. For example, any results of SHA-256 hash function will be represented by 64 characters, and any elements in 2048-bit prime field (modulo a 2048-bit integer) will be represented by 512 characters, regardless of how much 0's will be appear in front of such representations. Session-identifiers and other non-cryptographically generated values are represented in any (even) length determined by the side who generates it first, and the same length SHALL be used throughout the all communications by both peers.

The numbers represented as base64-fixed-number SHALL be generated as follows: first, the number is converted to a big-endian radix-256 binary representation as an octet string. The length of the representation is determined in the same way as mentioned above. Then, the string is encoded using the Base 64 encoding [RFC4648] without any spaces and newlines. Implementations decoding base64-fixed-number SHOULD reject any input data with invalid characters, excess/insufficient paddings, or non-canonical pad bits (See Sections 3.1 to 3.5 of [RFC4648]).

The canonical format for integer and hex-fixed-number are unquoted tokens, and that for base64-fixed-number is quoted-string.

4. Messages

In this section we define the seven kinds of messages used in the authentication protocol along with the formats and requirements of the headers for each message.

To determine which message are expected to be sent, see Sections 9 and 10.

In the descriptions below, the type of allowable values for each header parameter is shown in parenthesis after each parameter name. The "algorithm-determined" type means that the acceptable value for

the parameter is one of the types defined in Section 3, and is determined by the value of the "algorithm" parameter. The parameters marked "mandatory" SHALL be contained in the message. The parameters marked "non-mandatory" MAY either be contained or omitted in the message. Each parameter SHALL appear in each headers exactly once at most.

All credentials and challenges MAY contain any parameters not explicitly specified in the following sections. Recipients who do not understand such parameters MUST silently ignore those. However, all credentials and challenges MUST meet the following criteria:

- For responses, the parameters "reason", any "ks*" (where * stands for any decimal integers), and "vks" are mutually exclusive: any challenge MUST NOT contain two or more parameters among them. They MUST NOT contain any "kc*" and "vkc" parameters.

- For requests, the parameters "kc*" (where * stands for any decimal integers), and "vks" are mutually exclusive and any challenge MUST NOT contain two or more parameters among them. They MUST NOT contain any "ks*" and "vks" parameters.

4.1. 401-INIT and 401-STALE

Every 401-INIT or 401-STALE message SHALL be a valid HTTP 401-status (Authentication Required) message containing one (and only one: hereafter not explicitly noticed) "WWW-Authenticate" header containing a "reason" parameter in the challenge. The challenge SHALL contain all of the parameters marked "mandatory" below, and MAY contain those marked "non-mandatory".

version: (mandatory extensive-token) should be the token "-draft11" in this specification. The behavior is undefined when other values are specified.

algorithm: (mandatory extensive-token) specifies the authentication algorithm to be used. The value MUST be one of the tokens specified in [I-D.oiwa-http-mutualauth-algo] or other supplemental specification documentation.

validation: (mandatory extensive-token) specifies the method of host validation. The value MUST be one of the tokens described in Section 7, or the tokens specified in other supplemental specification documentation.

auth-domain: (non-mandatory string) specifies the authentication domain, the set of hosts for which the authentication credentials are valid. It MUST be one of the strings described in Section 5. If the value is omitted, it is assumed to be the "single-port" type domain in Section 5.

realm: (mandatory string) is a UTF-8 encoded string representing the name of the authentication realm inside the authentication domain. As specified in [I-D.ietf-httpbis-p7-auth], this value MUST always be sent in the quoted-string form.

pwd-hash: (non-mandatory extensive-token) specifies the hash algorithm (hereafter referred to by ph) used for additionally hashing the password. The valid tokens are

* none: ph(p) = p

* md5: ph(p) = MD5(p)

* digest-md5: ph(p) = MD5(username | ":" | realm | ":" | p), the same value as MD5(A1) for "MD5" algorithm in [RFC2617].

* sha1: ph(p) = SHA1(p)

If omitted, the value "none" is assumed. The use of "none" is recommended.

reason: (mandatory extensive-token) SHALL be an extensive-token which describes the possible reason of the failed authentication/authorization. Both servers and clients SHALL understand and support the following three tokens:

* initial: authentication was not tried because there was no Authorization header in the corresponding request.

* stale-session: the provided sid; in the request was either unknown to or expired in the server.

* auth-failed: authentication trial was failed by some reasons, possibly with a bad authentication credentials.

Implementations MAY support the following tokens or any extensive-tokens defined outside this specification. If clients has received any unknown tokens, these SHOULD treat these as if it were "auth-failed" or "initial".

* reauth-needed: server-side application requires a new authentication trial, regardless of the current status.

* invalid-parameters: authentication was not even tried in the server-side because some parameters are not acceptable.

* internal-error: authentication was not even tried in the server-side because there is some troubles on the server-side.

* user-unknown: a special case of auth-failed, suggesting that the provided user-name is invalid. The use of this parameter is NOT RECOMMENDED for security implications, except for special-purpose applications which makes this value sense.

* invalid-credential: ditto, suggesting that the provided user-name was valid but authentication was failed. The use of this parameter is NOT RECOMMENDED as the same as the above.

* authz-failed: authentication was successful, but access to the specified resource is not authorized to the specific authenticated user. (It is different from 403 responses which suggest that the reason of inaccessibility is other that authentication.)

The algorithm specified in this header will determine the types (among those defined in Section 3) and the values for K_c1, K_s1, VK_c and VK_s.

Among these messages, those with the reason parameter of value "stale-session" will be called "401-STALE" messages hereafter, because these have a special meaning in the protocol flow. Messages with any other reason parameters will be called "401-INIT" messages.

4.2. req-KEX-C1

Every req-KEX-C1 message SHALL be a valid HTTP request message containing an "Authorization" header with a credential containing a "kc1" parameter.

The credential SHALL contain the parameters with the following names:

version: (mandatory, extensive-token) should be the token "-draft11" in this specification. The behavior is undefined when other values are specified.

algorithm, validation, auth-domain, realm: MUST be the same value as it is when received from the server.

user: (mandatory, string) is the UTF-8 encoded name of the user. If this name comes from a user input, client software SHOULD prepare the string using SASLprep [RFC4013] before encoding it to UTF-8.

kc1: (mandatory, algorithm-determined) is the client-side key exchange value K_c1, which is specified by the algorithm that is used.

rekey-sid: (non-mandatory, hex-fixed-number): reserved for future extensions (see rekey-method in "200-VFY-S" message).

4.3. 401-KEX-S1

Every 401-KEX-S1 message SHALL be a valid HTTP 401-status (Authentication Required) response message containing a "WWW-Authenticate" header with a challenge containing a "ks1" parameter.

The challenge SHALL contain the parameters with the following names:

version: (mandatory, extensive-token) should be the token "-draft11" in this specification. The behavior is undefined when other values are specified.

algorithm, validation, auth-domain, realm: MUST be the same value as it is when received from the client.

sid: (mandatory, hex-fixed-number) MUST be a session identifier, which is a random integer. The sid SHOULD have uniqueness of at least 80 bits or the square of the maximal estimated transactions concurrently available in the session table, whichever is larger.

See Section 6 for more details.

ks1: (mandatory, algorithm-determined) is the server-side key exchange value K_s1, which is specified by the algorithm.

nc-max: (mandatory, integer) is the maximal value of nonce counts that the server accepts.

nc-window: (mandatory, integer) the number of available nonce slots that the server will accept. The value of the nc-window parameter is RECOMMENDED to be 32 or more.

time: (mandatory, integer) represents the suggested time (in seconds) that the client can reuse the session represented by the sid. It is RECOMMENDED to be at least 60. The value of this parameter is not directly linked to the duration that the server keeps track of the session represented by the sid.

path: (non-mandatory, string) specifies which path in the URI space the same authentication is expected to be applied. The value is a space-separated list of URIs, in the same format as it was specified in domain parameter [RFC2617] for the Digest authentications, and clients are RECOMMENDED to recognize it. The all path elements contained in the parameter MUST be inside the specified auth-domain: if not, clients SHOULD ignore such elements.

4.4. req-VFY-C

Every req-VFY-C message SHALL be a valid HTTP request message containing an "Authorization" header with a credential containing a "vkc" parameter.

The parameters contained in the header are as follows:

version: (mandatory, extensive-token) should be the token "-draft11" in this specification. The behavior is undefined when other values are specified.

algorithm, validation, auth-domain, realm: MUST be the same value as it is when received from the server for the session.

sid: (mandatory, hex-fixed-number) MUST be one of the sid values that was received from the server for the same authentication realm.

nc: (mandatory, integer) is a nonce value that is unique among the requests sharing the same sid. The values of the nonces SHOULD satisfy the properties outlined in Section 6.

vkc: (mandatory, algorithm-determined) is the client-side authentication verification value VK_c, which is specified by the algorithm.

4.5. 200-VFY-S

Every 200-VFY-S message SHALL be a valid HTTP message that is not of the 401 (Authentication Required) status, containing an "Authentication-Info" header with a "vks" parameter.

The parameters contained in the header are as follows:

version: (mandatory, extensive-token) should be the token "-draft11" in this specification. The behavior is undefined when other values are specified.

sid: (mandatory, hex-fixed-number) MUST be the value received from the client.

vks: (mandatory, algorithm-determined) is the server-side authentication verification value VK_s, which is specified by the algorithm.

logout-timeout: (non-mandatory, integer) is the number of seconds after which the client should re-validate the user's password for the current authentication realm. The value 0 means that the client SHOULD automatically forget the user-inputted password for the current authentication realm and revert to the unauthenticated state (i.e. server-initiated logout). This does not, however, mean that the long-term memories for the passwords (such as the password reminders and auto fill-ins) should be removed. If a new timeout value is received for the same authentication realm, it overrides the previous timeout. If logout-timeout parameters are specified both in an Authentication-Info header and an Authentication-Control header ([I-D.oiwa-http-auth-extension]), the client SHOULD

respect the smaller one of those and ignore the other.

rekey-method: (non-mandatory, extensive-token): defining a credential used for reestablishing a new session with a new sid. It must be either omitted or the token "passwords" at the current specification. The bare-tokens "refresh-key" and "refresh-key-global" are reserved for future extensions.

The header MUST be sent before the content body: it MUST NOT be sent in the trailer of a chunked-encoded response. If a "100 Continue" response is sent from the server, the Authentication-Info header SHOULD be included in that response, instead of the final response.

5. Authentication Realms

In this protocol, an "authentication realm" is defined as a set of resources (URIs) for which the same set of user names and passwords is valid for. If the server requests authentication for an authentication realm that the client is already authenticated for, the client will automatically perform the authentication using the already-known secrets. However, for the different authentication realms, the clients SHOULD NOT automatically reuse the usernames and passwords for another realm.

Just like in Basic and Digest access authentication protocols, Mutual authentication protocol supports multiple, separate protection spaces to be set up inside each host. Furthermore, the protocol supports that a single authentication realm spans over several hosts within the same Internet domain.

Each authentication realm is defined and distinguished by the triple of an "authentication algorithm", an "authentication domain", and a "realm" parameter. However, server operators are NOT RECOMMENDED to use the same pair of an authentication domain and a realm for different authentication algorithms.

The realm parameter is a string as defined in Section 4. Authentication domains are described in the remainder of this section.

An authentication domain specifies the range of hosts that the authentication realm spans over. In this protocol, it MUST be one of the following strings.

- Single-server type: The string in format "<scheme>://<host>:<port>", where <scheme>, <host>, and <port> are

the corresponding URI parts of the request URI. Even if the request-URI does not have a port part, the string will include one (i.e. 80 for http and 443 for https). The port part MUST NOT contain leading zeros. Use this when authentication is only valid for specific protocol (such as https).

- Single-host type: The "host" part of the requested URI. This is the default value. Authentication realms within this kind of authentication domain will span over several protocols (i.e. http and https) and ports, but not over different hosts.

- Wildcard-domain type: The string in format "*.<domain-postfix>", where <domain-postfix> is either the host part of the requested URI or any domain in which the requested host is included (this means that the specification "*.example.com" is valid for all of hosts "www.example.com", "web.example.com", "www.sales.example.com" and "example.com"). The domain-postfix sent from the servers MUST be equal to or included in a valid Internet domain assigned to a specific organization: if clients know, by some means such as a blacklist for HTTP cookies [RFC6265], that the specified domain is not to be assigned to any specific organization (e.g. "*.com" or "*.jp"), the clients are RECOMMENDED to reject the authentication request.

In the above specifications, every "scheme", "host", and "domain" MUST be in lower-case, and any internationalized domain names beyond the ASCII character set SHALL be represented in the way they are sent in the underlying HTTP protocol, represented in lower-case characters; i.e. these SHALL be in the form of the LDH labels in IDNA [RFC5890]. All "port"s MUST be in the shortest, unsigned, decimal number notation. Not obeying these requirements will cause failure of valid authentication attempts.

5.1. Resolving Ambiguities

In the above definitions of authentication domains, several domains will overlap each other. Depending on the "path" parameters given in the "401-KEX-S1" message (see Section 4), there may be several candidates when the client is going to send a request including an authentication credential (Steps 3 and 4 of the decision procedure presented in Section 9).

If such choices are required, the following procedure SHOULD be followed.

- If the client has previously sent a request to the same URI, and if it remembers the authentication realm requested by 401-INIT messages at that time, use that realm.

- o In other cases, use one of authentication realms representing the most-specific authentication domains. From the list of possible domain specifications shown above, each one earlier has priority over ones described after that.

  If there are several choices with different domain-postfix specifications, the one that has the longest domain-postfix has priority over ones with a shorter domain-postfix.

- o If there are realms with the same authentication domain, there is no defined priority: the client MAY choose any one of the possible choices.

If possible, server operators are encouraged to avoid such ambiguities by properly setting the "path" parameters.

6. Session Management

In the Mutual authentication protocol, a session represented by an sid is set up using first four messages (first request, 401-INIT, req-KEX-C1 and 401-KEX-S1), and a "session secret" (z) associated with the session is established. After sharing a session secret, this session, along with the secret, can be used for one or more requests for resources protected by the same realm in the same server. Note that session management is only an inside detail of the protocol and usually not visible to normal users. If a session expires, the client and server SHOULD automatically reestablish another session without informing the users.

Sessions and session identifiers are local to each server (defined by scheme, host and port) inside an authentication domain; the clients MUST establish separate sessions for each port of a host to be accessed. Furthermore, sessions and identifiers are also local to each authentication realm, even if these are provided from the same servers. The same session identifiers provided either from different servers or for different realms SHOULD be treated as independent ones.

The server SHOULD accept at least one req-VFY-C request for each session, given that the request reaches the server in a time window specified by the timeout parameter in the 401-KEX-S1 message, and that there are no emergent reasons (such as flooding attacks) to forget the sessions. After that, the server MAY discard any session at any time and MAY send 401-STALE messages for any req-VFY-C requests.

The client MAY send two or more requests using a single session

specified by the sid. However, for all such requests, each value of the nonce (in the nc parameter) MUST satisfy the following conditions:

- It is a natural number.
- The same nonce was not sent within the same session.
- It is not larger than the nc-max value that was sent from the server in the session represented by the sid.
- It is larger than (largest-nc - nc-window), where largest-nc is the maximal value of nc which was previously sent in the session, and nc-window is the value of the nc-window parameter which was received from the server in the session.

The last condition allows servers to reject any nonce values that are "significantly" smaller than the "current" value (defined by the value of nc-window) of the nonce used in the session involved. In other words, servers MAY treat such nonces as "already received". This restriction enables servers to implement duplicated nonce detection in a constant amount of memory (for each session).

Servers MUST check for duplication of the received nonces, and if any duplication is detected, the server MUST discard the session and respond with a 401-STALE message, as outlined in Section 10. The server MAY also reject other invalid nonce values (such as ones above the nc-max limit) by sending a 401-STALE message.

For example, assume the nc-window value of the current session is 32, nc-max is 100, and that the client has already used the following nonce values: {1-20, 22, 24, 30-38, 45-60, 63-72}. Then the nonce values that can be used for next request is one of the following set: {41-44, 61-62, 73-100}. The values {0, 21, 23, 25-29, 39-40} MAY be rejected by the server because they are not above the current "window limit" (40 = 72 - 32).

Typically, clients can ensure the above property by using a monotonically-increasing integer counter that counts from zero upto the value of nc-max.

The values of the nonces and any nonce-related values MUST always be treated as natural numbers within an infinite range. Implementations using fixed-width integers or fixed-precision floating numbers MUST correctly and carefully handle integer overflows. Such implementations are RECOMMENDED to accept any larger values that cannot be represented in the fixed-width integer representations, as long as other limits such as internal header-length restrictions are

not involved. The protocol is designed carefully so that both the clients and servers can implement the protocol using only fixed-width integers, by rounding any overflowed values to the maximum possible value.

7. Validation Methods

The "validation method" specifies a method to "relate" the mutual authentication processed by this protocol with other authentications already performed in the underlying layers and to prevent man-in-the-middle attacks. It decides the value v that is an input to the authentication protocols.

The valid tokens for the validation parameter and corresponding values of v are as follows:

host: hostname validation: The value v will be the ASCII string in the following format: "<scheme>://<host>:<port>", where <scheme>, <host>, and <port> are the URI components corresponding to the currently accessing resource. The scheme and host are in lower-case, and the port is in a shortest decimal representation. Even if the request-URI does not have a port part, v will include one.

tls-cert: TLS certificate validation: The value v will be the octet string of the hash value of the public key certificate used in the underlying TLS [RFC5246] (or SSL) connection. The hash value is defined as the value of the entire signed certificate (specified as "Certificate" in [RFC5280]), hashed by the hash algorithm specified by the authentication algorithm used.

tls-key: TLS shared-key validation: The value v will be the octet string of the shared master secret negotiated in the underlying TLS (or SSL) connection.

If the HTTP protocol is used on a non-encrypted channel (TCP and SCTP, for example), the validation type MUST be "host". If HTTP/TLS [RFC2818] (HTTPS) protocol is used with the server certificates, the validation type MUST be either "tls-cert" or "tls-key". If HTTP/TLS protocol is used with an anonymous Diffie-Hellman key exchange, the validation type MUST be "tls-key" (see the note below).

If the validation type "tls-cert" is used, the server certificate provided on TLS connection MUST be verified to make sure that the

server actually owns the corresponding secret key.

Clients MUST validate this parameter upon reception of the 401-INIT messages.

However, when the client is a Web browser with any scripting capabilities, the underlying TLS channel used with HTTP/TLS MUST provide server identity verification. This means (1) the anonymous Diffie-Hellman key exchange ciphersuite MUST NOT be used, and (2) the verification of the server certificate provided from the server MUST be performed.

For other systems, when the underlying TLS channel used with HTTP/TLS does not perform server identity verification, the client SHOULD ensure that all the responses are validated using the Mutual authentication protocol, regardless of the existence of the 401-INIT responses.

Note: The protocol defines two variants for validation on the TLS connections. The "tls-key" method is more secure. However, there are some situations where tls-cert is more preferable.

- When TLS accelerating proxies are used, it is difficult for the authenticating server to acquire the TLS key information that is used between the client and the proxy. This is not the case for client-side "tunneling" proxies using a CONNECT method extension of HTTP.

- When a black-box implementation of the TLS protocol is used on either peer.

Implementations supporting a Mutual authentication over the HTTPS protocol SHOULD support the "tls-cert" validation. Support for "tls-key" validation is OPTIONAL for both the servers and clients.

8. Authentication Extensions

The HTTP authentication extensions described in [I-D.oiwa-http-auth-extension] is a definitive part of this protocol. Interactive clients (e.g. Web browsers) supporting this protocol are RECOMMENDED to support non-mandatory authentication and the Authentication-Control header defined there, except the "auth-style" parameter. This specification also proposes (however, not mandates) default "auth-style" to be "non-modal". Web applications SHOULD however consider the security impacts of the behaviors of clients that do not support these headers.

Authentication-initializing messages with the Optional-WWW-Authenticate header are used where 401-INIT response is valid. Such a message is called a 200-Optional-INIT message in this document. (It will not replace other 401-type messages such as 401-STALE and 401-KEX-S1.)

9. Decision Procedure for Clients

To securely implement the protocol, the user client must be careful about accepting the authenticated responses from the server. This also holds true for the reception of "normal responses" (responses which do not contain Mutual-related headers) from HTTP servers.

Clients SHOULD implement a decision procedure equivalent to the one shown below. (Unless implementers understand what is required for the security, they should not alter this.) In particular, clients SHOULD NOT accept "normal responses" unless explicitly allowed below. The labels on the steps are for informational purposes only. Action entries within each step are checked in top-to-bottom order, and the first clause satisfied SHOULD be taken.

Step 1 (step_new_request):
: If the client software needs to access a new Web resource, check whether the resource is expected to be inside some authentication realm for which the user has already been authenticated by the Mutual authentication scheme. If yes, go to Step 2. Otherwise, go to Step 5.

Step 2:
: Check whether there is an available sid for the authentication realm you expect. If there is one, go to Step 3. Otherwise, go to Step 4.

Step 3 (step_send_vfy_1):
: Send a req-VFY-C request.

  * If you receive a 401-INIT message with a different authentication realm than expected, go to Step 6.

  * If you receive a 200-Optional-INIT message with a different authentication realm than expected, go to Step 6.

  * If you receive a 401-STALE message, go to Step 9.

  * If you receive a 401-INIT message, go to Step 13.

* If you receive a 200-VFY-S message, go to Step 14.

* If you receive a normal response, go to Step 11.

Step 4 (step_send_kex1_1):
Send a req-KEX-C1 request.

* If you receive a 401-INIT message with a different authentication realm than expected, go to Step 6.

* If you receive a 200-Optional-INIT message with a different authentication realm than expected, go to Step 6.

* If you receive a 401-KEX-S1 message, go to Step 10.

* If you receive a 401-INIT message with the same authentication realm, go to Step 13 (see Note 1).

* If you receive a normal response, go to Step 11.

Step 5 (step_send_normal_1):
Send a request without any Mutual authentication headers.

* If you receive a 401-INIT message, go to Step 6.

* If you receive a 200-Optional-INIT message, go to Step 6.

* If you receive a normal response, go to Step 11.

Step 6 (step_rcvd_init):
Check whether you know the user's password for the requested authentication realm. If yes, go to Step 7. Otherwise, go to Step 12.

Step 7:
Check whether there is an available sid for the authentication realm you expect. If there is one, go to Step 8. Otherwise, go to Step 9.

Step 8 (step_send_vfy):
Send a req-VFY-C request.

* If you receive a 401-STALE message, go to Step 9.

* If you receive a 401-INIT message, go to Step 13.

* If you receive a 200-VFY-S message, go to Step 14.

Step 9 (step_send_kex1):
Send a req-KEX-C1 request.

* If you receive a 401-KEX-S1 message, go to Step 10.

* If you receive a 401-INIT message, go to Step 13 (See Note 1).

Step 10 (step_rcvd_kex1):
Send a req-VFY-C request.

* If you receive a 401-INIT message, go to Step 13.

* If you receive a 200-VFY-S message, go to Step 14.

Step 11 (step_rcvd_normal):
The requested resource is out of the authenticated area. The client will be in the "UNAUTHENTICATED" status. If the response contains a request for authentications other than Mutual, it MAY be handled normally.

Step 12 (step_rcvd_init_unknown):
The requested resource requires a Mutual authentication, and the user is not yet authenticated. The client will be in the "AUTH-REQUESTED" status, and is RECOMMENDED to process the content sent from the server, and to ask user for a user name and a password. When those are supplied from the user, proceed to Step 9.

Step 13 (step_rcvd_init_failed):
For some reason the authentication failed: possibly the password or the username is invalid for the authenticated resource. Forget the password for the authentication realm and go to Step 12.

Step 14 (step_rcvd_vfy):
Check the validity of the received VK_s value. If it is equal to the expected value, it means that the mutual authentication has succeeded. The client will be in the "AUTH-SUCCEEDED" status.

If the value is unexpected, it is a fatal communication error.

If a user explicitly requests to log out (via user interfaces), the client MUST forget the user’s password, go to step 5 and reload the current resource without an authentication header.

Note 1: These transitions MAY be accepted by clients, but NOT RECOMMENDED for servers to initiate.

Any kind of response (including a normal response) other than those

shown in the above procedure SHOULD be interpreted as a fatal communication error, and in such cases the clients MUST NOT process any data (response body and other content-related headers) sent from the server. However, to handle exceptional error cases, clients MAY accept a message without an Authentication-Info header, if it is a Server-Error (5xx) status. The client will be in the "UNAUTHENTICATED" status in these cases.

The client software SHOULD display the three client status to the end-user. For an interactive client, however, if a request is a sub-request for a resource included in another page (e.g., embedded images, style sheets, frames etc.), its status MAY be omitted from being shown, and any "AUTH-REQUESTED" statuses MAY be treated in the same way as an "UNAUTHENTICATED" status.

Figure 5 shows a diagram of the client-side state.

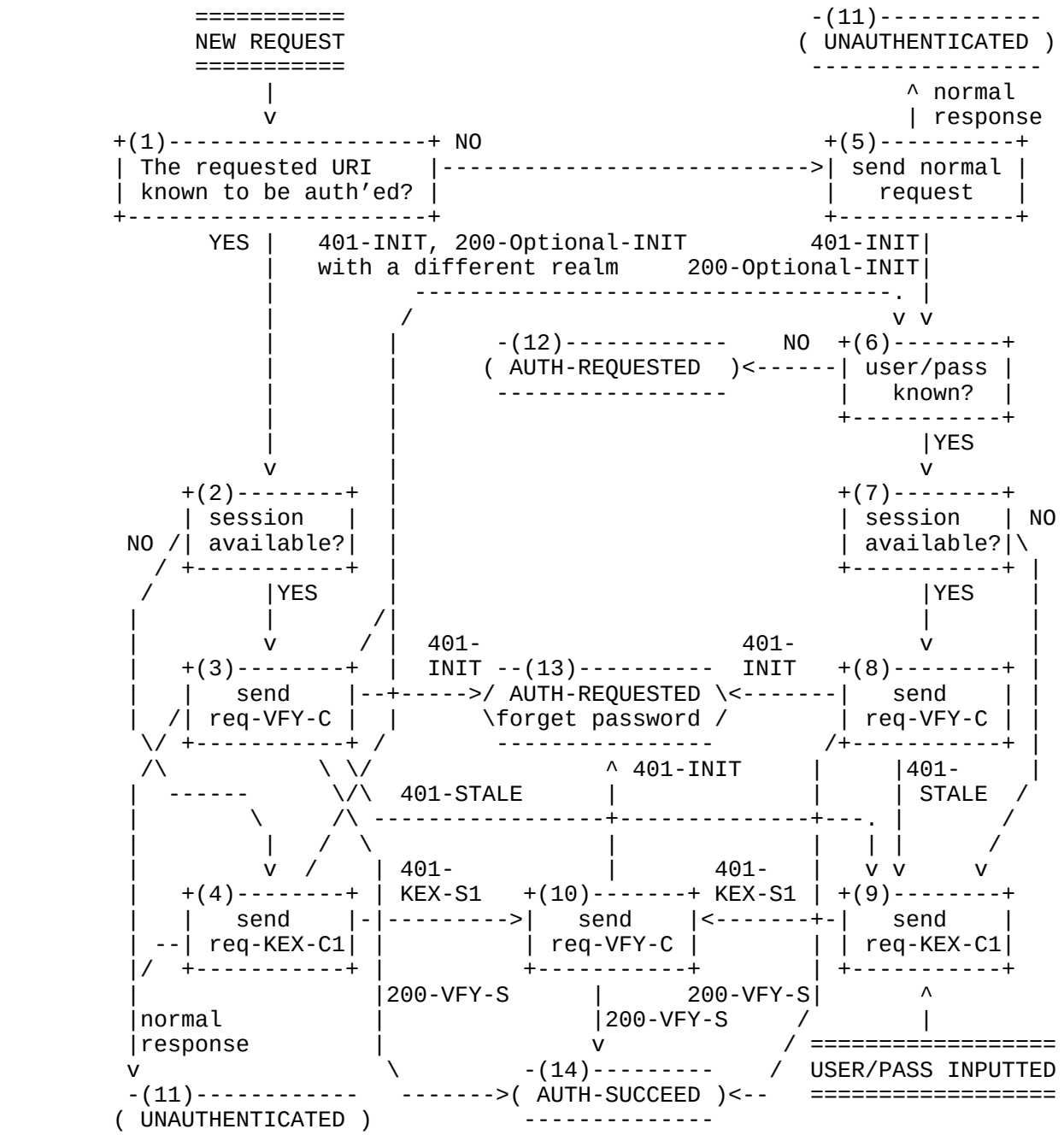


Figure 5: State diagram for clients

10. Decision Procedure for Servers

Each server SHOULD have a table of session states. This table need not be persistent over a long term; it MAY be cleared upon server restart, reboot, or others. Each entry in the table SHOULD contain at least the following information:

- The session identifier, the value of the sid parameter.
- The algorithm used.
- The authentication realm.
- The state of the protocol: one of "key exchanging", "authenticated", "rejected", or "inactive".
- The user name received from the client
- The boolean flag noting whether or not the session is fake.
- When the state is "key exchanging", the values of K_c1 and S_s1.
- When the state is "authenticated", the following information:
  - The value of the session secret z
  - The largest nc received from the client (largest-nc)
  - For each possible nc values between (largest-nc - nc-window + 1) and max_nc, a flag whether or not a request with the corresponding nc has been received.

The table MAY contain other information.

Servers SHOULD respond to the client requests according to the following procedure:

- When the server receives a normal request:
  - If the requested resource is not protected by the Mutual Authentication, send a normal response.
  - If the resource is protected by the Mutual Authentication, send a 401-INIT response.
  - If the resource is protected by the optional Mutual Authentication, send a 200-Optional-INIT response.

- When the server receives a req-KEX-C1 request:
  - If the requested resource is not protected by the Mutual Authentication, send a normal response.
  - If the authentication realm specified in the req-KEX-C1 request is not the expected one, send either a 401-INIT or a 200-Optional-INIT response.
  - If the server cannot validate the parameter kc1, send a 401-INIT response.
  - If the received user name is either invalid, unknown or unacceptable, create a new session, mark it a "fake" session, compute a random value as K_s1, and send a fake 401-KEX-S1 response. (Note: the server SHOULD NOT send a 401-INIT response in this case, because it will leak the information to the client that the specified user will not be accepted. Instead, postpone it to the response for the next req-VFY-C request.)
  - Otherwise, create a new session, compute K_s1 and send a 401-KEX-S1 response.

  The created session has the "key exchanging" state.

- When the server receives a req-VFY-C request:
  - If the requested resource is not protected by the Mutual Authentication, send a normal response.
  - If the authentication realm specified in the req-VFY-C request is not the expected one, send either a 401-INIT or a 200-Optional-INIT response.

  If none of above holds true, the server will lookup the session corresponding to the received sid and the authentication realm.

  - If the session corresponding to the received sid could not be found, or it is in the "inactive" state, send a 401-STALE response.
  - If the session is in the "rejected" state, send either a 401-INIT or a 401-STALE message.
  - If the session is in the "authenticated" state, and the request has an nc value that was previously received from the client, send a 401-STALE message. The session SHOULD be changed to the

"inactive" status.

* If the nc value in the request is larger than the nc-max parameter sent from the server, or if it is not larger then (largest-nc - nc-window) (when in "authenticated" status), the server MAY (but not REQUIRED to) send a 401-STALE message. The session SHOULD be changed to the "inactive" status if so.

* If the session is a "fake" session, or if the received vkc is incorrect, then send a 401-INIT response. If the session is in the "key exchanging" state, it SHOULD be changed to the "rejected" state; otherwise, it MAY either be changed to the "rejected" status or kept in the previous state.

* Otherwise, send a 200-VFY-S response. If the session was in the "key exchanging" state, the session SHOULD be changed to an "authenticated" state. The maximum nc and nc flags of the state SHOULD be updated properly.

At any time, the server MAY change any state entries with both the "rejected" and "authenticated" statuses to the "inactive" status, and MAY discard any "inactive" states from the table. The entries with the "key exchanging" status SHOULD be kept unless there is an emergency situation such as a server reboot or a table capacity overflow.

11. Authentication Algorithms

Cryptographic authentication algorithms which are used with this protocol will be defined separately. The algorithm definition MUST at least provide a definitions for the following functions:

- The server-side authentication credential J, derived from user-side authentication credential pi.
- Key exchange values K_c1, K_s1 (exchanged on wire) and S_c1, S_s1 (kept secret in each peer).
- Shared secret z, to be computed in both server-side and client side.
- A hash function H to be used with the protocol.

All algorithm used with this protocol SHOULD provide secure mutual authentication between client and servers, and generate a cryptographically strong shared secret value z, equivalently strong to or stronger than the hash function H. If any passwords (or pass-

phrases or any equivalents, i.e. weak secrets) are involved, these SHOULD NOT be guessable from any data transmitted in the protocol, even if an attacker (either an eavesdropper or an active server) knows the possible thoroughly-searchable candidate list of the passwords. Furthermore, if possible, the function for deriving server-side authentication credential J is RECOMMENDED to be one-way so that pi should not be easily computed from J(pi).

11.1. Support Functions and Notations

In this section we define several support functions and notations to be shared by several algorithm definitions:

The integers in the specification are in decimal, or in hexadecimal when prefixed with "0x".

The function octet(c) generates a single octet string whose code value is equal to c. The operator |, when applied to octet strings, denotes the concatenation of two operands.

The function VI encodes natural numbers into octet strings in the following manner: numbers are represented in big-endian radix-128 string, where each digit is represented by a octet within 0x80-0xff except the last digit represented by a octet within 0x00-0x7f. The first octet MUST NOT be 0x80. For example, VI(i) = octet(i) for i < 128, and VI(i) = octet(0x80 + (i >> 7)) | octet(i & 127) for 128 <= i < 16384. This encoding is the same as the one used for the subcomponents of object identifiers in the ASN.1 encoding [ITU.X690.1994], and available as a "w" conversion in the pack function of several scripting languages.

The function VS encodes a variable-length octet string into a uniquely-decoded, self-delimited octet string, as in the following manner:

VS(s) = VI(length(s)) | s

where length(s) is a number of octets (not characters) in s.

Some examples:

VI(0) = "\000" (in C string notation)

VI(100) = "d"

VI(10000) = "\316\020"

VI(1000000) = "\275\204@"

VS("") = "\000"

VS("Tea") = "\003Tea"

VS("Caf<e acute>" [in UTF-8]) = "\005Caf\303\251"

VS([10000 "a"s]) = "\316\020aaaaa..." (10002 octets)

[Editorial note: Unlike the colon-separated notion used in the Basic/Digest HTTP authentication scheme, the string generated by a concatenation of the VS-encoded strings will be unique, regardless of the characters included in the strings to be encoded.]

The function OCTETS converts an integer into the corresponding radix-256 big-endian octet string having its natural length: See Section 3.1.3 for the definition of "natural length".

11.2. Default Functions for Algorithms

The functions defined in this section are common default functions among authentication algorithms.

The client-side password-based string pi used by this authentication is derived in the following manner:

pi = H(VS(algorithm) | VS(auth-domain) | VS(realm) | VS(username) | VS(ph(password))).

The values of algorithm, realm, and auth-domain are taken from the values contained in the 401-INIT (or 200-Optional-INIT, hereafter implied) message. When pi is used in the context of an octet string, it SHALL have the natural length derived from the size of the output of function H (e.g. 32 octets for SHA-256). The function ph is determined by the value of the pwd-hash parameter given in a 401-INIT message. If the password comes from a user input, it SHOULD first be prepared using SASLprep [RFC4013]. Then, the password SHALL be encoded as a UTF-8 string before passed to ph.

The values VK_c and VK_s are derived by the following equation.

VK_c = H(octet(4) | OCTETS(K_c1) | OCTETS(K_s1) | OCTETS(z) | VI(nc) | VS(v))

VK_s = H(octet(3) | OCTETS(K_c1) | OCTETS(K_s1) | OCTETS(z) | VI(nc) | VS(v))

Specifications for cryptographic algorithms used with this framework MAY override the functions pi, VK_c, and VK_s defined above. In such cases implementations MUST use the ones defined with such algorithm specifications.

12. Application Channel Binding

Applications and upper-layer communication protocols may need authentication binding to the HTTP-layer authenticated user. Such applications MAY use the following values as a standard shared secret.

These values are parameterized with an optional octet string (t) which may be arbitrarily chosen by each applications or protocols. If there is no appropriate value to be specified, use a null string for t.

For applications requiring binding to either an authenticated user or a shared-key session (to ensure that the requesting client is certainly authenticated), the following value b_1 MAY be used.

b_1 = OCTETS(H(OCTETS(H(octet(6) | OCTETS(K_c1) | OCTETS(K_s1) | OCTETS(z) | VI(0) | VS(v))) | VS(t))).

For applications requiring binding to a specific request (to ensure that the payload data is generated for the exact HTTP request), the following value b_2 MAY be used.

b_2 = OCTETS(H(OCTETS(H(octet(7) | OCTETS(K_c1) | OCTETS(K_s1) | OCTETS(z) | VI(nc) | VS(v))) | VS(t))).

Note: Channel bindings to lower-layer transports (TCP and TLS) are defined in Section 7.

13. Application for Proxy Authentication

The authentication scheme defined by the previous sections can be applied m.m. for proxy authentications. In such cases, the following alterations MUST be applied:

- The 407 status is to be sent and recognized for places where the 401 status is used,
- Proxy-Authenticate: header is to be used for places where WWW-Authenticate: is used,

- Proxy-Authorization: header is to be used for places where Authorization: is used,
- Proxy-Authentication-Info: header is to be used for places where Authentication-Info: is used,
- The auth-domain parameter is fixed to the host-name of the proxy, which means to cover all requests processed through the specific proxy,
- The limitation for the paths contained in the path parameter of 401-KEX-S1 messages is disregarded,
- The omission of the path parameter of 401-KEX-S1 messages means that the authentication realm will potentially cover all requests processed by the proxy,
- The scheme, host name and the port of the proxy is used for validation tokens, and
- Authentication extension in [I-D.oiwa-http-auth-extension] is not applicable.

The requirements for client software to display the authentication status to the end-user is also not applicable for proxy authentication. If the client software supports both end-to-end and proxy authentication using this protocol, it SHOULD be careful that the authentication status of the proxy communication will never be confused by users with authentication statuses of the end-to-end resource authentications.

14. Methods to Extend This Protocol

If a private extension to this protocol is implemented, it MUST use the extension-tokens defined in Section 3 to avoid conflicts with this protocol and other extensions. (standardized or being-standardizing extensions MAY use either bare-tokens or extension-tokens.)

Specifications defining authentication algorithms MAY use other representations for the parameters "kc1", "ks1", "vkc", and "vks", replace those parameter names, and/or add parameters to the messages containing those parameters in supplemental specifications, provided that syntactic and semantic requirements in Section 3, [I-D.ietf-httpbis-p1-messaging] and [I-D.ietf-httpbis-p7-auth] are satisfied. Any parameters starting with "kc", "ks", "vkc" or "vks" and followed by decimal natural numbers (e.g. kc2, ks0, vkc1, vks3

etc.) are reserved for this purpose. If those specifications use names other than those mentioned above, it is RECOMMENDED to use extension-tokens to avoid any parameter name conflict with the future extension of this protocol.

Extension-tokens MAY be freely used for any non-standard, private, and/or experimental uses for those parameters provided that the domain part in the token is appropriately used.

15. IANA Considerations

When bare-tokens are used for the authentication-algorithm, pwd-hash, and validation parameters MUST be allocated by IANA. To acquire registered tokens, a specification for the use of such tokens MUST be available as an RFC, as outlined in [RFC5226].

Note: More formal declarations will be added in the future drafts to meet the RFC 5226 requirements.

16. Security Considerations

16.1. Security Properties

- The protocol is secure against passive eavesdropping and replay attacks. However, the protocol relies on transport security including DNS integrity for data secrecy and integrity. HTTP/TLS SHOULD be used where transport security is not assured and/or data secrecy is important.

- When used with HTTP/TLS, if TLS server certificates are reliably verified, the protocol provides true protection against active man-in-the-middle attacks.

- Even if the server certificate is not used or is unreliable, the protocol provides protection against active man-in-the-middle attacks for each HTTP request/response pair. However, in such cases, JavaScript or similar scripting facilities can be used to affect the Mutually-authenticated contents from other contents not protected by this authentication mechanism. This is the reason why this protocol requires that valid TLS server certificates MUST be presented (Section 7).

16.2. Denial-of-service Attacks to Servers

The protocol requires a server-side table of active sessions, which may become a critical point of the server resource consumptions. For proper operation, the protocol requires that at least one key verification request is processed for each session identifier. After that, servers MAY discard sessions internally at any time, without causing any operational problems to clients. Clients will silently reestablishes a new session then.

However, if a malicious client sends too many requests of key exchanges (req-KEX-C1 messages) only, resource starvation might occur. In such critical situations, servers MAY discard any kind of existing sessions regardless of these statuses. One way to mitigate such attacks are that servers MAY have a number and a time limits for unverified pending key exchange requests (in the "wa received" status).

This is a common weakness of authentication protocols with almost any kind of negotiations or states, including Digest authentication method and most Cookie-based authentication implementations. However, regarding the resource consumption, a situation of the mutual authentication method is a slightly better than the Digest, because HTTP requests without any kind of authentication requests will not generate any kind of sessions. Session identifiers are only generated after a client starts a key negotiation. It means that simple clients such as web crawlers will not accidentally consume server-side resources for session managements.

16.3. Implementation Considerations

- To securely implement the protocol, the Authentication-Info headers in the 200-VFY-S messages MUST always be validated by the client. If the validation fails, the client MUST NOT process any content sent with the message, including other headers and the body part. Non-compliance to this requirement will allow phishing attacks.

- The authentication status on the client-side SHOULD be visible to the users of the client. In addition, the method for asking for the user’s name and passwords SHOULD be carefully designed so that (1) the user can easily distinguish the request from this authentication method from any other authentication methods such as Basic and Digest methods, and (2) the Web contents cannot imitate the user-interfaces for this protocol.

  An informational memo regarding user-interface considerations and recommendations for implementing this protocol will be separately

published.

- For HTTP/TLS communications, when a web form is submitted from Mutually-authenticated pages with the "tls-cert" validation method to a URI that is protected by the same realm (so indicated by the path parameter), if the server certificate has been changed since the pages were received, the peer is RECOMMENDED to be revalidated using a req-KEX-C1 message with an "Expect: 100-continue" header. The same applies when the page is received with the "tls-key" validation method, and when the TLS session has expired.

- Server-side storages of user passwords are advised to contain the values encrypted by one-way function J(pi), instead of the real passwords, those hashed by ph, or pi.

16.4. Usage Considerations

- The user-names inputted by a user may be sent automatically to any servers sharing the same auth-domain. This means that when host-type auth-domain is used for authentication on an HTTPS site, and when an HTTP server on the same host requests Mutual authentication within the same realm, the client will send the user-name in a clear text. If user-names have to be kept secret against eavesdropping, the server must use full-scheme-type auth-domain parameter. Contrarily, passwords are not exposed to eavesdroppers even on HTTP requests.

- The "pwd-hash" parameter is only provided for backward compatibility of password databases. The use of "none" function is the most secure choice and is RECOMMENDED. If values other than "none" are used, you MUST ensure that the hash values of the passwords were not exposed to the public. Note that hashed password databases for plain-text authentications are usually not considered secret.

- If the server provides several ways for storing server-side password secrets into the password database, it is advised to store the values encrypted by using the one-way function J(pi), instead of the real passwords, those hashed by ph, or pi.

17. Notice on Intellectual Properties

The National Institute of Advanced Industrial Science and Technology (AIST) and Yahoo! Japan, Inc. has jointly submitted a patent application on the protocol proposed in this documentation to the Patent Office of Japan. The patent is intended to be open to any implementors of this protocol and its variants under non-exclusive

royalty-free manner. For the details of the patent application and its status, please contact the author of this document.

The elliptic-curve based authentication algorithms might involve several existing third-party patents. The authors of the document take no position regarding the validity or scope of such patents, and other patents as well.

18. References

18.1. Normative References

[I-D.ietf-httpbis-p1-messaging]
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-19 (work in progress), March 2012.

[I-D.ietf-httpbis-p7-auth]
Fielding, R., Lafon, Y., and J. Reschke, "HTTP/1.1, part 7: Authentication", draft-ietf-httpbis-p7-auth-19 (work in progress), March 2012.

[I-D.oiwa-http-auth-extension]
Oiwa, Y., Watanabe, H., Takagi, H., Kihara, B., Hayashi, T., and Y. Ioku, "HTTP Authentication Extensions for Interactive Clients", draft-oiwa-http-auth-extension-02 (work in progress), June 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

18.2. Informative References

[I-D.ietf-oauth-v2]
Hammer-Lahav, E., Recordon, D., and D. Hardt, "The OAuth 2.0 Authorization Framework", draft-ietf-oauth-v2-26 (work in progress), May 2012.

[I-D.ietf-precis-framework]
Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation and Comparison of Internationalized Strings in Application Protocols", draft-ietf-precis-framework-03 (work in progress), May 2012.

[I-D.oiwa-http-mutualauth-algo]
Oiwa, Y., Watanabe, H., Takagi, H., Kihara, B., Hayashi, T., and Y. Ioku, "Mutual Authentication Protocol for HTTP: KAM3-based Cryptographic Algorithms", draft-oiwa-http-mutualauth-algo-02 (work in progress), May 2012.

[ISO.10646-1.1993]
International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane", ISO Standard 10646-1, May 1993.

[ITU.X690.1994]
International Telecommunications Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, 1994.

[OASIS.saml-core-2.0-os]
Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.

[OIDF.Connect.Standard]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, "OpenID Connect Standard 1.0 - draft 10", May 2012.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

Appendix A. (Informative) Draft Remarks from Authors

The following items are currently under consideration for future revisions by the authors.

- Whether to keep TLS-key validation or not.
- When keeping tls-key validation, whether to use "TLS channel binding" [RFC5929] for "tls-key" verification (Section 7). Note that existing TLS implementations should be considered to determine this.
- Adopt [I-D.ietf-precis-framework] for replacing SASLprep reference. Especially, use NFC canonicalization instead of NFKC.
- Adding test vectors for ensuring implementation correctness.
- Possibly adding a method for servers to detect availability of Mutual authentication on client-side.

- Possible support for optional key renewal and cross-site federated authentication.

Appendix B. (Informative) Draft Change Log

B.1. Changes in Revision 12

Note: the token for the header parameter "version" is NOT changed from "-draft11", because the protocol semantics has not been changed in this revision.

- Added a reason "authz-failed".

B.2. Changes in Revision 11

- Message syntax definition reverted to pre-07 style as httpbis-p1 and p7 now defines a precise rule for parameter value parsing.
- Replaced "stale" parameter with more infomative/extensive "reason" parameter in 401-INIT and 401-STALE.
- Reserved "rekey-sid" and "rekey-method" parameters for future extensions.
- Added descriptions for replacing/non-replacing existing technologies.

B.3. Changes in Revision 10

- The authentication extension parts (non-mandatory authentication and authentication controls) are separated to yet another draft.
- The default auth-domain parameter is changed to the full scheme-host-port syntax, which is consistent with usual HTTP authentication framework behavior.
- Provision for application channel binding is added.
- Provision for proxy access authentication is added.
- Bug fix: syntax specification of sid parameter was wrong: it was inconsistent with the type specified in the main text (the bug introduced in -07 draft).
- Terminologies for headers are changed to be in harmony with httpbis drafts (e.g. field to parameter).

- Syntax definitions are changed to use HTTP-extended ABNF syntax, and only the header values are shown for header syntax, in harmony with httpbis drafts.

- Names of parameters and corresponding mathematical values are now renamed to more informative ones. The following list shows correspondence between the new and the old names.

new name	old name	description
S_c1, S_s1	s_a, s_b	client/server-side secret randoms
K_c1, K_s1	w_a, w_b	client/server-side exchanged key components
kc1, ks1	wa, wb	parameter names for those
VK_c, VK_s	o_a, o_b	client/server-side key verifiers
vkc, vks	oa, ob	parameter names for those
z	z	session secrets

B.4. Changes in Revision 09

- The (default) cryptographic algorithms are separated to another draft.

- Names of the messages are changed to more informative ones than before. The following is the correspondence table of those names:

new name	old name	description
401-INIT	401-B0	initial response
401-STALE	401-B0-stale	session key expired
req-KEX-C1	req-A1	client->server key exchange
401-KEX-S1	401-B1	server->client key exchange
req-VFY-C	req-A3	client->server auth. verification
200-VFY-S	200-B4	server->client auth. verification
200-Optional-INIT	200-Optional-B0	initial with non-mandatory authentication

B.5. Changes in Revision 08

- The English text has been revised.

B.6. Changes in Revision 07

- Adapt to httpbis HTTP/1.1 drafts:
  - Changed definition of extensive-token.
  - LWSP continuation-line (%0D.0A.20) deprecated.
- To simplify the whole spec, the type of nonce-counter related parameters are change from hex-integer to integer.
- Algorithm tokens are renamed to include names of hash algorithms.
- Clarified the session management, added details of server-side protocol decisions.
- The whole draft was reorganized; introduction and overview has been rewritten.

B.7. Changes in Revision 06

- Integrated Optional Mutual Authentication to the main part.
- Clarified the decision procedure for message recognitions.
- Clarified that a new authentication request for any sub-requests in interactive clients may be silently discarded.
- Typos and confusing phrases are fixed.
- Several "future considerations" are added.

B.8. Changes in Revision 05

- A new parameter called "version" is added for supporting future incompatible changes with a single implementation. In the (first) final specification its value will be changed to 1.
- A new header "Authentication-Control" is added for precise control of application-level authentication behavior.

B.9. Changes in Revision 04

- Changed text of patent licenses: the phrase "once the protocol is accepted as an Internet standard" is removed so that the sentence also covers the draft versions of this protocol.

- The "tls-key" verification is now OPTIONAL.
- Several description fixes and clarifications.

B.10. Changes in Revision 03

- Wildcard domain specifications (e.g. "*.example.com") are allowed for auth-domain parameters (Section 4.1).
- Specification of the "tls-cert" verification is updated (incompatible change).
- State transitions fixed.
- Requirements for servers concerning w_a values are clarified.
- RFC references are updated.

B.11. Changes in Revision 02

- Auth-realm is extended to allow full-scheme type.
- A decision diagram for clients and decision procedures for servers are added.
- 401-B1 and req-A3 messages are changed to contain authentication realm information.
- Bugs on equations for o_A and o_B are fixed.
- Detailed equations for the entire algorithm are included.
- Elliptic-curve algorithms are updated.
- Several clarifications and other minor updates.

B.12. Changes in Revision 01

- Several texts are rewritten for clarification.
- Added several security consideration clauses.

Authors' Addresses

Yutaka Oiwa
National Institute of Advanced Industrial Science and Technology
Research Institute for Secure Systems
Tsukuba Central 2
1-1-1 Umezono
Tsukuba-shi, Ibaraki
JP

Email: mutual-auth-contact-ml@aist.go.jp

Hajime Watanabe
National Institute of Advanced Industrial Science and Technology

Hiromitsu Takagi
National Institute of Advanced Industrial Science and Technology

Boku Kihara
Lepidum Co. Ltd.
#602, Village Sasazuka 3
1-30-3 Sasazuka
Shibuya-ku, Tokyo
JP

Tatsuya Hayashi
Lepidum Co. Ltd.

Yuichi Ioku
Yahoo! Japan, Inc.
Midtown Tower
9-7-1 Akasaka
Minato-ku, Tokyo
JP