Network Working Group
Request for Comments: 3797
Obsoletes: 2777
Category: Informational

D. Eastlake 3rd
Motorola Laboratories
June 2004

Publicly Verifiable Nominations Committee (NomCom) Random Selection

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This document describes a method for making random selections in such a way that the unbiased nature of the choice is publicly verifiable. As an example, the selection of the voting members of the IETF Nominations Committee (NomCom) from the pool of eligible volunteers is used. Similar techniques would be applicable to other cases.

Table of Contents

1. Introduction

Under the IETF rules, each year ten people are randomly selected from among eligible volunteers to be the voting members of the IETF nominations committee (NomCom). The NomCom nominates members of the Internet Engineering Steering Group (IESG) and the Internet Architecture Board (IAB) as described in [RFC 3777]. The number of eligible volunteers in recent years has been around 100.

It is highly desirable that the random selection of the voting NomCom be done in an unimpeachable fashion so that no reasonable charges of bias or favoritism can be brought. This is as much for the protection of the selection administrator (currently, the appointed non-voting NomCom chair) from suspicion of bias as it is for the protection of the IETF.

A method such that public information will enable any person to verify the randomness of the selection meets this criterion. This document gives an example of such a method.

The method, in the form it appears in RFC 2777, was also used by IANA in February 2003 to determine the ACE prefix for Internationalized Domain Names [RFC 3490] so as to avoid claim jumping.

2. General Flow of a Publicly Verifiable Process

A selection of NomCom members publicly verifiable as unbiased or similar selection could follow the three steps given below.

2.1. Determination of the Pool

First, determine the pool from which the selection is to be made as provided in [RFC 3777] or its successor.

Volunteers are solicited by the selection administrator. Their names are then passed through the IETF Secretariat to check eligibility. (Current eligibility criteria relate to IETF meeting attendance, records of which are maintained by the Secretariat.) The full list of eligible volunteers is made public early enough that a reasonable time can be given to resolve any disputes as to who should be in the pool.

2.2. Publication of the Algorithm

The exact algorithm to be used, including the public future sources of randomness, is made public. For example, the members of the final list of eligible volunteers are ordered by publicly numbering them, some public future sources of randomness such as government run lotteries are specified, and an exact algorithm is specified whereby eligible volunteers are selected based on a strong hash function [RFC 1750] of these future sources of randomness.

2.3. Publication of Selection

When the pre-specified sources of randomness produce their output, those values plus a summary of the execution of the algorithm for selection should be announced so that anyone can verify that the correct randomness source values were used and the algorithm properly executed. The algorithm can be run to select, in an ordered fashion, a larger number than are actually necessary so that if any of those selected need to be passed over or replaced for any reason, an ordered set of additional alternate selections will be available. A cut off time for any complaint that the algorithm was run with the wrong inputs or not faithfully executed must be specified to finalize the output and provide a stable selection.

3. Randomness

The crux of the unbiased nature of the selection is that it is based in an exact, predetermined fashion on random information which will be revealed in the future and thus can not be known to the person specifying the algorithm. That random information will be used to control the selection. The random information must be such that it will be publicly and unambiguously revealed in a timely fashion.

The random sources must not include anything that any reasonable person would believe to be under the control or influence of the IETF or its components, such as IETF meeting attendance statistics, numbers of documents issued, or the like.

3.1. Sources of Randomness

Examples of good information to use are winning lottery numbers for specified runnings of specified public lotteries. Particularly for government run lotteries, great care is taken to see that they occur on time and produce random quantities. Even in the unlikely case one were to have been rigged, it would almost certainly be in connection with winning money in the lottery, not in connection with IETF use.

Other possibilities are such things as the daily balance in the US Treasury on a specified day, the volume of trading on the New York Stock exchange on a specified day, etc. (However, the reference code given below will not handle integers that are too large.) Sporting events can also be used. (Experience has indicated that stock prices and/or volumes are a poor source of unambiguous data due trading suspensions, company mergers, delistings, splits, multiple markets, etc.) In all cases, great care must be taken to specify exactly what quantities are being presumed random and what will be done if their issuance is cancelled, delayed, or advanced.

It is important that the last source of randomness, chronologically, produce a substantial amount of the entropy needed. If most of the randomness has come from the earlier of the specified sources, and someone has even limited influence on the final source, they might do an exhaustive analysis and exert such influence so as to bias the selection in the direction they wanted. Thus it is best for the last source to be an especially strong and unbiased source of a large amount of randomness such as a government run lottery.

It is best not to use too many different sources. Every additional source increases the probability that one or more sources might be delayed, cancelled, or just plain screwed up somehow, calling into play contingency provisions or, worst of all, creating a situation that was not anticipated. This would either require arbitrary judgment by the selection administrator, defeating the randomness of the selection, or a re-run with a new set of sources, causing much delay. Three or four would be a good number of sources. Ten is too many.

3.2. Skew

Some of the sources of randomness produce data that is not uniformly distributed. This is certainly true of volumes, prices, and horse race results, for example. However, use of a strong mixing function [RFC 1750] will extract the available entropy and produce a hash value whose bits, remainder modulo a small divisor, etc., deviate from a uniform distribution only by an insignificant amount.

3.3. Entropy Needed

What we are doing is selecting N items without replacement from a population of P items. The number of different ways to do this is as follows, where "!" represents the factorial function:

$$\frac{P!}{N! * (P - N)!}$$

To do this in a completely random fashion requires as many random bits as the logarithm base 2 of that quantity. Some sample calculated approximate number of random bits for the completely random selection of 10 NomCom members from various pool sizes is given below:

Random Selection of Ten Items From Pool

Pool size	20	25	30	35	40	50	60	75	100	120
Bits needed	18	22	25	28	30	34	37	40	44	47

Using an inadequate number of bits means that not all of the possible sets of ten selected items would be available. For a substantially inadequate amount of entropy, there could be a significant correlation between the selection of two different members of the pool, for example. However, as a practical matter, for pool sizes likely to be encountered in IETF NomCom membership selection, 40 bits of entropy should always be adequate. Even if there is a large pool and more bits are needed for perfect randomness, 40 bits of entropy will assure only an insignificant deviation from completely random selection for the difference in probability of selection of different pool members, the correlation between the selection of any pair of pool members, etc.

An MD5 [RFC 1321] hash has 128 bits and therefore can produce no more than that number of bits of entropy. However, this is more than three times what is likely to ever be needed for IETF NomCom membership selection. A even stronger hash, such as SHA-1 [RFC 3174], can be used if desired.

4. A Suggested Precise Algorithm

It is important that a precise algorithm be given for mixing the random sources specified and making the selection based thereon. Sources suggested above produce either a single positive number (i.e., NY Stock Exchange volume in thousands of shares) or a small set of positive numbers (many lotteries provide 6 numbers in the range of 1 through 40 or the like, a sporting event could produce the scores of two teams, etc.). A suggested precise algorithm is as follows:

1. For each source producing multiple numeric values, represent each as a decimal number terminated by a period (or with a period separating the whole from the fractional part), without leading zeroes (except for a single leading zero if the integer part is zero), and without trailing zeroes after the period.

2. Order the values from each source from smallest to the largest and concatenate them and suffix the result with a "/". For each source producing a single number, simply represent it as above with a suffix "/". (This sorting is necessary because the same lottery results, for example, are sometimes reported in the order numbers were drawn and sometimes in numeric order and such things as the scores of two sports teams that play a game has no inherent order.)

3. At this point you have a string for each source, say s1/, s2/, ... Concatenate these strings in a pre-specified order, the order in which the sources were listed if not otherwise specified, and represent each character as its ASCII code [ASCII] producing "s1/s2/.../".

You then produce a sequence of random values derived from a strong mixing of these sources by calculating the MD5 hash [RFC 1321] of this string prefixed and suffixed with an all zeros two byte sequence for the first value, the string prefixed and suffixed by 0x0001 for the second value, etc., treating the two bytes as a big endian counter. Treat each of these derived "random" MD5 output values as a positive 128-bit multiprecision big endian integer.

Then totally order the pool of listed volunteers as follows: If there are P volunteers, select the first by dividing the first derived random value by P and using the remainder plus one as the position of the selectee in the published list. Select the second by dividing the second derived random value by P-1 and using the remainder plus one as the position in the list with the first selected person eliminated. Etc.

It is STRONGLY recommended that alphanumeric random sources be avoided due to the much greater difficulty in canonicalizing them in an independently repeatable fashion; however, if you choose to ignore this advice and use an ASCII or similar Roman alphabet source or sources, all white space, punctuation, accents, and special characters should be removed and all letters set to upper case. This will leave only an unbroken sequence of letters A-Z and digits 0-9 which can be treated as a canonicalized number above and suffixed with a "./". If you choose to not just ignore but flagrantly flout this advice and try to use even more complex and harder to canonicalize internationalized text, such as UNICODE, you are on your own.

5. Handling Real World Problems

In the real world, problems can arise in following the steps and flow outlined in Sections 2 through 4 above. Some problems that have actually arisen are described below with recommendations for handling them.

5.1. Uncertainty as to the Pool

Every reasonable effort should be made to see that the published pool from which selection is made is of certain and eligible persons. However, especially with compressed schedules or perhaps someone whose claim that they volunteered and are eligible has not been resolved by the deadline, or a determination that someone is not eligible which occurs after the publication of the pool, it may be that there are still uncertainties.

The best way to handle this is to maintain the announced schedule, INCLUDE in the published pool all those whose eligibility is uncertain and to keep the published pool list numbering IMMUTABLE after its publication. If someone in the pool is later selected by the algorithm and random input but it has been determined they are ineligible, they can be skipped and the algorithm run further to make an additional selection. Thus the uncertainty only effects one selection and in general no more than a maximum of U selections where there are U uncertain pool members.

Other courses of action are far worse. Actual insertion or deletion of entries in the pool after its publication changes the length of the list and totally scrambles who is selected, possibly changing every selection. Insertion into the pool raises questions of where to insert: at the beginning, end, alphabetic order, ... Any such choices by the selection administrator after the random numbers are known destroys the public verifiability of fair choice. Even if done before the random numbers are known, such dinking with the list after its publication just smells bad. There should be clear fixed public deadlines and someone who challenges their absence from the pool after the published deadline should have their challenge automatically denied for tardiness.

5.2. Randomness Ambiguities

The best good faith efforts have been made to specify precise and unambiguous sources of randomness. These sources have been made public in advance and there has not been objection to them. However, it has happened that when the time comes to actually get and use this randomness, the real world has thrown a curve ball and it isn’t quite clear what data to use. Problems have particularly arisen in connection with stock prices, volumes, and financial exchange rates or indices. If volumes that were published in thousands are published in hundreds, you have a rounding problem. Prices that were quoted in fractions or decimals can change to the other. If you take care of every contingency that has come up in the past, you can be hit with a new one. When this sort of thing happens, it is generally too late to announce new sources, an action which could raise suspicions of its own. About the only course of action is to make a reasonable choice within the ambiguity and depend on confidence in the good faith of the selection administrator. With care, such cases should be extremely rare.

Based on these experiences, it is again recommended that public lottery numbers or the like be used as the random inputs and stock prices and volumes avoided.

6. Fully Worked Example

Assume the following ordered list of 25 eligible volunteers is published in advance of selection:

1. John
2. Mary
3. Bashful
4. Dopey
5. Sleepy
6. Grouchy
7. Doc
8. Sneazy
9. Handsome
10. Cassandra
11. Pollyanna
12. Pendragon
13. Pandora
14. Faith
15. Hope
16. Charity
17. Lee
18. Longsuffering
19. Chastity
20. Smith
21. Pride
22. Sloth
23. Envy
24. Anger
25. Kasczynski

Assume the following (fake example) ordered list of randomness sources:

1. The Kingdom of Alphaland State Lottery daily number for 1 November 2004 treated as a single four digit integer.
2. Numbers of the winning horses at Hialeia for all races for the first day on or after 13 October 2004 on which at least two races are run.
3. The People’s Democratic Republic of Betastani State Lottery six winning numbers (ignoring the seventh "extra" number) for 1 November 2004.

Hypothetical randomness publicly produced:

Source 1: 9319
Source 2: 2, 5, 12, 8, 10
Source 3: 9, 18, 26, 34, 41, 45

Resulting key string:

```
9319./2.5.8.10.12./9.18.26.34.41.45./
```

The table below gives the hex of the MD5 of the above key string bracketed with a two byte string that is successively 0x0000, 0x0001, 0x0002, through 0x0010 (16 decimal). The divisor for the number size of the remaining pool at each stage is given and the index of the selectee as per the original number of those in the pool.

index	hex value of MD5	div	selected
1	990DD0A5692A029A98B5E01AA28F3459	25	-> 17 <-
2	3691E55CB63FCC37914430B2F70B5EC6	24	-> 7 <-
3	FE814EDF564C190AC1D25753979990FA	23	-> 2 <-
4	1863CCACEB568C31D7DDBDF1D4E91387	22	-> 16 <-
5	F4AB33DF4889F0AF29C513905BE1D758	21	-> 25 <-
6	13EAEB529F61ACFB9A29D0BA3A60DE4A	20	-> 23 <-
7	992DB77C382CA2BDB9727001F3CDCCD9	19	-> 8 <-
8	63AB4258ECA922976811C7F55C383CE7	18	-> 24 <-
9	DFBC5AC97CED01B3A6E348E3CC63F40D	17	-> 19 <-
10	31CB111C4A4EBE9287CEAE16FE51B909	16	-> 13 <-
11	07FA46C122F164C215BBC72793B189A3	15	-> 22 <-
12	AC52F8D75CCBE2E61AFEB3387637D501	14	-> 5 <-
13	53306F73E14FC0B2FBF434218D25948E	13	-> 18 <-
14	B5D1403501A81F9A47318BE7893B347C	12	-> 9 <-
15	85B10B356AA06663EF1B1B407765100A	11	-> 1 <-
16	3269E6CE559ABD57E2BA6AAB495EB9BD	10	-> 4 <-

Resulting first ten selected, in order selected:

1. Lee (17)
2. Doc (7)
3. Mary (2)
4. Charity (16)
5. Kasczynski (25)
6. Envy (23)
7. Sneazy (8)
8. Anger (24)
9. Chastity (19)
10. Pandora (13)

Should one of the above turn out to be ineligible or decline to serve, the next would be Sloth, number 22.

7. Security Considerations

Careful choice of should be made of randomness inputs so that there is no reasonable suspicion that they are under the control of the administrator. Guidelines given above to use a small number of inputs with a substantial amount of entropy from the last should be followed. And equal care needs to be given that the algorithm selected is faithfully executed with the designated inputs values.

Publication of the results and a week or so window for the community of interest to duplicate the calculations should give a reasonable assurance against implementation tampering.

8. Reference Code

This code makes use of the MD5 reference code from [RFC 1321] ("RSA Data Security, Inc. MD5 Message-Digest Algorithm"). The portion of the code dealing with multiple floating point numbers was written by Matt Crawford. The original code in RFC 2777 could only handle pools of up to 255 members and was extended to 2**16-1 by Erik Nordmark. This code has been extracted from this document, compiled, and tested. While no flaws have been found, it is possible that when used with some compiler on some system some flaw will manifest itself.

```
/****************************************************************
  *
  *  Reference code for
  *      "Publicly Verifiable Random Selection"
  *          Donald E. Eastlake 3rd
  *              February 2004
  *
  ****************************************************************/
 #include <limits.h>
 #include <math.h>
 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>

 /* From RFC 1321 */
 #include "global.h"
 #include "MD5.h"

 /* local prototypes */
 int longremainder ( unsigned short divisor,
                     unsigned char dividend[16] );
 long int getinteger ( char *string );
 double NPentropy ( int N, int P );


 /* limited to up to 16 inputs of up to sixteen integers each */
 /* pool limit of 2**8-1 extended to 2**16-1 by Erik Nordmark */
 /****************************************************************/

 main ()
 {
 int         i, j,  k, k2, err, keysize, selection, usel;
```

```
    unsigned short   remaining, *selected;
    long int    pool, temp, array[16];
    MD5_CTX     ctx;
    char        buffer[257], key [800], sarray[16][256];
    unsigned char    uc16[16], unch1, unch2;

    pool = getinteger ( "Type size of pool:\n" );
    if ( pool > 65535 )

        {
        printf ( "Pool too big.\n" );
        exit ( 1 );
        }
    selected = (unsigned short *) malloc ( (size_t)pool );
    if ( !selected )
        {
        printf ( "Out of memory.\n" );
        exit ( 1 );
        }
    selection = getinteger ( "Type number of items to be selected:\n" );
    if ( selection > pool )
        {
        printf ( "Pool too small.\n" );
        exit ( 1 );
        }
    if ( selection == pool )
        printf ( "All of the pool is selected.\n" );
    else
        {
        err = printf ( "Approximately %.1f bits of entropy needed.\n",
                       NPentropy ( selection, pool ) + 0.1 );
        if ( err <= 0 ) exit ( 1 );
        }
    for ( i = 0, keysize = 0; i < 16; ++i )
        {
        if ( keysize > 500 )
            {
            printf ( "Too much input.\n" );
            exit ( 1 );
            }
        /* get the "random" inputs. echo back to user so the user may
           be able to tell if truncation or other glitches occur.  */
        err = printf (
            "\nType #%d randomness or ’end’ followed by new line.\n"
            "Up to 16 integers or the word ’float’ followed by up\n"
            "to 16 x.y format reals.\n", i+1 );
        if ( err <= 0 ) exit ( 1 );
        gets ( buffer );
        j = sscanf ( buffer,
                "%ld%ld%ld%ld%ld%ld%ld%ld%ld%ld%ld%ld%ld%ld%ld%ld",
            &array[0], &array[1], &array[2], &array[3],
            &array[4], &array[5], &array[6], &array[7],
            &array[8], &array[9], &array[10], &array[11],
            &array[12], &array[13], &array[14], &array[15] );
        if ( j == EOF )
            exit ( j );
        if ( !j )
            if ( buffer[0] == ’e’ )
                break;

            else
                {   /* floating point code by Matt Crawford */
                j = sscanf ( buffer,
                    "float %ld.%[0-9]%ld.%[0-9]%ld.%[0-9]%ld.%[0-9]"
                    "%ld.%[0-9]%ld.%[0-9]%ld.%[0-9]%ld.%[0-9]"
                    "%ld.%[0-9]%ld.%[0-9]%ld.%[0-9]%ld.%[0-9]"
                    "%ld.%[0-9]%ld.%[0-9]%ld.%[0-9]%ld.%[0-9]",
                    &array[0], sarray[0], &array[1], sarray[1],
                    &array[2], sarray[2], &array[3], sarray[3],
                    &array[4], sarray[4], &array[5], sarray[5],
                    &array[6], sarray[6], &array[7], sarray[7],
                    &array[8], sarray[8], &array[9], sarray[9],
                    &array[10], sarray[10], &array[11], sarray[11],
                    &array[12], sarray[12], &array[13], sarray[13],
                    &array[14], sarray[14], &array[15], sarray[15] );
                if ( j == 0 || j & 1 )
                    printf ( "Bad format." );
                else {
                     for ( k = 0, j /= 2; k < j; k++ )
                     {
                           /* strip trailing zeros */
                     for ( k2=strlen(sarray[k]); sarray[k][--k2]==’0’;)
                           sarray[k][k2] = ’\0’;
                     err = printf ( "%ld.%s\n", array[k], sarray[k] );
                     if ( err <= 0 ) exit ( 1 );
                     keysize += sprintf ( &key[keysize], "%ld.%s",
                                          array[k], sarray[k] );
                     }
                     keysize += sprintf ( &key[keysize], "/" );
                     }
                }
        else
            {   /* sort values, not a very efficient algorithm */
            for ( k2 = 0; k2 < j - 1; ++k2 )
                for ( k = 0; k < j - 1; ++k )
                    if ( array[k] > array[k+1] )
```

```
                        {
                        temp = array[k];
                        array[k] = array[k+1];
                        array[k+1] = temp;
                        }
            for ( k = 0; k < j; ++k )
                { /* print for user check */
                err = printf ( "%ld ", array[k] );
                if ( err <= 0 ) exit ( 1 );
                keysize += sprintf ( &key[keysize], "%ld.", array[k] );
                }
            keysize += sprintf ( &key[keysize], "/" );
            }
        }   /* end for i */

    /* have obtained all the input, now produce the output */
    err = printf ( "Key is:\n %s\n", key );
    if ( err <= 0 ) exit ( 1 );
    for ( i = 0; i < pool; ++i )
        selected [i] = (unsigned short)(i + 1);
    printf ( "index        hex value of MD5        div  selected\n" );
    for (   usel = 0, remaining = (unsigned short)pool;
            usel < selection;
            ++usel, --remaining )
        {
        unch1 = (unsigned char)usel;
        unch2 = (unsigned char)(usel>>8);
        /* prefix/suffix extended to 2 bytes by Donald Eastlake */
        MD5Init ( &ctx );
        MD5Update ( &ctx, &unch2, 1 );
        MD5Update ( &ctx, &unch1, 1 );
        MD5Update ( &ctx, (unsigned char *)key, keysize );
        MD5Update ( &ctx, &unch2, 1 );
        MD5Update ( &ctx, &unch1, 1 );
        MD5Final ( uc16, &ctx );
        k = longremainder ( remaining, uc16 );
    /* printf ( "Remaining = %d, remainder = %d.\n", remaining, k ); */
        for ( j = 0; j < pool; ++j )
            if ( selected[j] )
                if ( --k < 0 )
                    {
                    printf ( "%2d  "
    "%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X%02X  "
    "%2d  -> %2d <-\n",
    usel+1, uc16[0],uc16[1],uc16[2],uc16[3],uc16[4],uc16[5],uc16[6],
    uc16[7],uc16[8],uc16[9],uc16[10],uc16[11],uc16[12],uc16[13],
    uc16[14],uc16[15], remaining, selected[j] );
                    selected[j] = 0;
```

```
                  break;
                  }
         }
      printf ( "\nDone, type any character to exit.\n" );
      getchar ();
      return 0;
      }

      /* prompt for a positive non-zero integer input */
      /****************************************************************/
      long int getinteger ( char *string )
      {
      long int     i;
      int          j;
      char    tin[257];

      while ( 1 )
      {
      printf ( string );
      printf ( "(or 'exit' to exit) " );
      gets ( tin );
      j = sscanf ( tin, "%ld", &i );
      if (    ( j == EOF )

          ||  ( !j && ( ( tin[0] == 'e' ) || ( tin[0] == 'E' ) ) )
              )
          exit ( j );
      if ( ( j == 1 ) &&
           ( i > 0 ) )
          return i;
      }   /* end while */
      }

      /* get remainder of dividing a 16 byte unsigned int
         by a small positive number */
      /****************************************************************/
      int longremainder ( unsigned short divisor,
                          unsigned char dividend[16] )
      {
      int i;
      long int kruft;

      if ( !divisor )
          return -1;
      for ( i = 0, kruft = 0; i < 16; ++i )
          {
          kruft = ( kruft << 8 ) + dividend[i];
          kruft %= divisor;
          }
      return kruft;
      }   /* end longremainder */

     /* calculate how many bits of entropy it takes to select N from P */
     /*****************************************************************/
     /*             P!
       log  ( ----------------- )
          2    N! * ( P - N )!
     */

     double NPentropy ( int N, int P )
     {
     int         i;
     double      result = 0.0;

     if (    ( N < 1 )   /* not selecting anything? */
        ||   ( N >= P )  /* selecting all of pool or more? */
        )
         return 0.0;     /* degenerate case */
     for ( i = P; i > ( P - N ); --i )
         result += log ( i );
     for ( i = N; i > 1; --i )
         result -= log ( i );
     /* divide by [ log (base e) of 2 ] to convert to bits */
     result /= 0.69315;

     return result;
     }   /* end NPentropy */
```

Appendix A: History of NomCom Member Selection

For reference purposes, here is a list of the IETF Nominations Committee member selection techniques and chairs so far:

YEAR	CHAIR	SELECTION METHOD
1993/1994	Jeff Case	Clergy
1994/1995	Fred Baker	Clergy
1995/1996	Guy Almes	Clergy
1996/1997	Geoff Huston	Spouse
1997/1998	Mike St.Johns	Algorithm
1998/1999	Donald Eastlake 3rd	RFC 2777
1999/2000	Avri Doria	RFC 2777
2000/2001	Bernard Aboba	RFC 2777
2001/2002	Theodore Ts’o	RFC 2777
2002/2003	Phil Roberts	RFC 2777
2003/2004	Rich Draves	RFC 2777

Clergy = Names were written on pieces of paper, placed in a receptacle, and a member of the clergy picked the NomCom members.

Spouse = Same as Clergy except chair’s spouse made the selection.

Algorithm = Algorithmic selection based on similar concepts to those documented in RFC 2777 and herein.

RFC 2777 = Algorithmic selection using the algorithm and reference code provided in RFC 2777 (but not the fake example sources of randomness).

Appendix B: Changes from RFC 2777

This document differs from [RFC 2777], the previous version, in three primary ways as follows:

(1) Section 5, on problems actually encountered with using these recommendations for selecting an IETF NomCom and on how to handle them, has been added.

(2) The selection algorithm code has been modified to handle pools of up to 2**16-1 elements and the counter based prefix and suffix concatenated with the key string before hashing has been extended to two bytes.

(3) Mention has been added that the algorithm documented herein was used by IANA to select the Internationalized Domain Name ACE prefix and some minor wording changes made.

(4) References have been divided into Informative and Normative.

(5) The list in Appendix A has been brought up to date.

Acknowledgements

Matt Crawford and Erik Nordmark made major contributions to this document. Comments by Bernard Aboba, Theodore Ts'o, Jim Galvin, Steve Bellovin, and others have been incorporated.

References

Normative References

[ASCII] "USA Standard Code for Information Interchange", X3.4, American National Standards Institute: New York, 1968.

[RFC 1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC 1750] Eastlake, 3rd, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

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

Informative References

[RFC 3777] Galvin, J., "IAB and IESG Selection, Confirmation, and Recall Process: Operation of the Nominating and Recall Committees", BCP 10, RFC 3777, April 2004.

[RFC 2777] Eastlake, 3rd, D., "Publicly Verifiable Nomcom Random Selection", RFC 2777, February 2000.

[RFC 3490] Falstrom, P., Hoffman, P. and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

Author’s Address

Donald E. Eastlake, 3rd
Motorola Laboratories
155 Beaver Street
Milford, MA 01757 USA

Phone: +1-508-786-7554(w)
       +1-508-634-2066(h)
EMail: Donald.Eastlake@motorola.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.