Network Working Group
Request for Comments: 2777
Category: Informational

D. Eastlake 3rd
Motorola
February 2000

# Publicly Verifiable Nomcom Random Selection

## Status of this Memo

## 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 from the pool of eligible volunteers is used. Similar techniques would be applicable to other cases.

## Acknowledgement

Matt Crawford made major contributions to this document.

## Table of Contents

1. Introduction..............................................2
2. General Flow of a Publicly Verifiable Process...........2
2.1 Determination of the Pool..............................2
2.2 Publication of the Algorithm...........................2
2.3 Publication of Selection...............................3
3. Randomness..............................................3
3.1 Sources of Randomness..................................3
3.2 Skew...................................................4
3.3 Entropy Needed.........................................4
4. A Suggested Precise Algorithm...........................5
5. Fully Worked Example....................................6
6. Security Considerations.................................7
7.  Reference Code.........................................8
Appendix: History of NomCom Member Selection..............14
References................................................15
Author's Address..........................................15
Full Copyright Statement..................................16

## 1. Introduction

Under the IETF rules, each year 10 persons are randomly selected from among the eligible persons who volunteer to be the voting members of the nominations committee (NomCom) to nominate members of the Internet Engineering Steering Group (IESG) and the Internet Architecture Board (IAB) [RFC 2727]. The number of eligible volunteers in recent years has varied in the approximate range of 40 to 60.

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

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.

## 2. General Flow of a Publicly Verifiable Process

In general, 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, you need to determine the pool from which the selection is to be made.

Volunteers are solicited by the appointed (non-voting) NomCom chair. 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 there is a reasonable time to resolve any disputes as to who should be in the pool, probably a week to ten days before the selection.

### 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, several 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 prespecified 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.  A cut off time for any complaint that the algorithm was run with the wrong inputs or not faithfully executed should be specified to finalize the output and provide a stable NomCom.

## 3. Randomness

The crux of the unbiased nature of the selection is that it is based exactly on random information which will be revealed in the future and thus can not be known to the person specifying the algorithm by which that random information will be used to select the NomCom members.   The random information must be such that it will be publicly revealed in a timely fashion.

The random sources should 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 lottery winning numbers for specified runnings of specified lotteries.  Particularly for government run lotteries, great care is usually taken to see that they 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 closing price of a stock on a particular day, 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 be used but only with care to specify exactly what quantities are being presumed random and what will be done if they are cancelled or delayed.

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 it might be delayed or cancelled calling into play contingency plans or, worst of all, possibly creating a situation that was not anticipated. This would either require arbitrary judgement by the Nomcom chair, defeating the randomness of the selection, or a re-run with a new set of sources, causing much delay. Probably a good number of sources is three.

## 3.2 Skew

Many of the sources of randomness suggested above produce data which is not uniformly distributed. This is certainly true of stock 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., are uniformly distributed.

## 3.3 Entropy Needed

What we are doing is selection 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:

```
                         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 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 |
|---|---|---|---|---|---|---|---|---|---|
| Bits needed | 18 | 22 | 25 | 28 | 30 | 34 | 37 | 40 | 44 |

Using an inadequate number of bits means that not all of the possible selections would be available. For a substantially inadequate amount of entropy, there would be substantial correlations between the selection of two 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 theoretically more bits are needed for complete randomness, 40 bits of entropy will assure that the probability of selection of each pool member differs from that of other pool members, the correlation between the selection of any pair of pool members, etc., differs only insignificantly from that for completely random selection.

An MD5 [RFC 1321] hash has 128 bits and therefore can produce no more than that number of bits of entropy. However, this is three times what is likely to ever been needed for IETF nomcom membership selection.

## 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 each produce either a single positive number (i.e., closing price for a stock) 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 sample precise algorithm is as follows:

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) and without leading zeroes (except for a single leading zero if the integer part is zero) or trailing zeroes after the period. Order them from smallest to the largest and concatenate them and follow the results by a "/". For each source producing a single number, simply represent it as above with a trailing "/". At this point you have a string for each source, say s1/, s2/, ... Concatenate these strings in a pre-specified order and represent each character as its ASCII code producing s1/s2/.../.

You can 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 a zero byte for the first value, the string prefixed and suffixed by a 0x01 byte for the second value, etc. Treat each of these derived random values as a positive multiprecision integer. If there are P eligible volunteers, select the first voting member by dividing the first derived random value by P and using the remainder plus one as the position of the selectee in the ordered list or volunteers. Select the second voting member by dividing the second derived random value by P-1 and using the remainder plus one as the position of the selectee in the list with the first selectee eliminated. Etc.

It is recommended that alphanumeric random sources be avoided due to the greater difficulty in canonicalizing them in an independently repeatable fashion; however, if any are used, all white space, punctuation, 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 "/".

## 5. Fully Worked Example

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

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

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

1. The People’s Democracy of Betastani State Lottery six winning numbers (ignoring the seventh "extra" number) for 1 October 1998.
2. Numbers of the winning horses at Hialeia for all races for the first day on or after x September 1998 on which at least two races are run.
3. The Republic of Alphaland State Lottery daily number for 1 October 1998 treated as a single four digit integer.
4. Closing price of Example Corporation stock on the Lunar Stock Exchange for the first business day after x September 1998 when it trades.

Randomness publicly produced:

```
    Source 1:  9, 18, 26, 34, 41, 45
    Source 2:  2, 5, 12, 8, 10
    Source 3:  9319
    Source 4:  13 11/16
```

Resulting key string:

```
    9.18.26.34.41.45./2.5.8.10.12./9319./13.6875/
```

The table below gives the hex of the MD5 of the above key string bracketed with a byte whose value is successively 0x00, 0x01, 0x02, through 0x09. 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 | 746612D0A75D2A2A39C0A957CF825F8D | 25 | -> 12 <- |
| 2 | 95E31A4429ED5AAF7377A15A8E10CD9D | 24 | -> 6 <- |
| 3 | AFB2B3FD30E82AD6DC35B4D2F1CFC77A | 23 | -> 8 <- |
| 4 | 06821016C2A2EA14A6452F4A769ED1CC | 22 | -> 3 <- |
| 5 | 94DA30E11CA7F9D05C66D0FD3C75D6F7 | 21 | -> 2 <- |
| 6 | 2FAE3964D5B1DEDD33FDA80F4B8EF45E | 20 | -> 24 <- |
| 7 | F1E7AB6753A773EFE46393515FDA8AF8 | 19 | -> 11 <- |
| 8 | 700B81738E07DECB4470879BEC6E0286 | 18 | -> 19 <- |
| 9 | 1F23F8F8F8E5638A29D332BC418E0689 | 17 | -> 15 <- |
| 10 | 61A789BA86BF412B550A5A05E821E0ED | 16 | -> 22 <- |

Resulting selection, in order selected:

```
      1. Pendragon (12)     6. Anger (24)
      2. Grouchy (6)        7. Pollyanna (11)
      3. Sneazy (8)         8. Chastity (19)
      4. Bashful (3)        9. Hope (15)
      5. Mary (2)          10. Sloth (22)
```

## 6. 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 shoud 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.

To maintain the unpredictable character of selections, should a member of the nomcom need to be replaced due to death, resignation, expulsion, etc., new publicly announced future random sources should be used for the selection of their replacement.

## 7.  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.

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

   #include "global.h"
   #include "MD5.h"

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


   /* limited to 16 inputs of up to sixteen integers each */
   /****************************************************************/

   main ()
   {
   int         i, j,  k, k2, err, keysize, pool, selection;
   unsigned char   unch, uc16[16], remaining, *selected;
   long int    temp, array[16];
   MD5_CTX ctx;
   char        buffer[257], key [800], sarray[16][256];

   pool = getinteger ( "Type size of pool:\n" );
   if ( pool > 255 )

       {
       printf ( "Pool too big.\n" );
       exit ( 1 );
       }
   selected = (unsigned char *) malloc ( 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" );
       exit ( 0 );
       }
   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] = i + 1;
   printf ( "index        hex value of MD5        div  selected\n" );
   for (   unch = 0, remaining = pool;
           unch < selection;
           ++unch, --remaining )
       {
       MD5Init ( &ctx );
       MD5Update ( &ctx, &unch, 1 );
       MD5Update ( &ctx, (unsigned char *)key, keysize );
       MD5Update ( &ctx, &unch, 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",
   unch+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 an integer input */
   /*****************************************************************/
   int getinteger ( char *string )
   {
   int     i, j;
   char    tin[257];

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

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

   /* get remainder of dividing a 16 byte unsigned int
      by a small positive number */
   /****************************************************************/
   int longremainder ( unsigned char 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 1.0;     /* 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: 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 | This Algorithm |
| 1999/2000 | Avri Doria | This Alogrithm |

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 the same concepts as documented herein.

This Algorithm = Algorithmic selection using the algorithm and reference code (but not the fake example sources of randomness) described herein.

## References

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

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

RFC 2727 Galvin, J., "IAB and IESG Selection, Confirmation, and Recall Process: Operation of the Nominating and Recall Committees", BCP 10, RFC 2727, February 2000.

## Author's Address

Donald E. Eastlake, 3rd
Motorola
65 Shindegan Hill Road, RR #1
Carmel, NY 10512 USA

Phone: +1-914-276-2668 (h)
+1-508-261-5434 (w)
Fax: +1-508-261-4447 (w)
EMail: Donald.Eastlake@motorola.com

## Full Copyright Statement

## Acknowledgement

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