Properties of the Classification Scheme of the Security Option for Internet Datagrams

Chyan Yang, Arthur Billingsley
Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, California 93943-5004

Abstract

The security option for datagrams that are transmitted through the internet will be essential as utilization grows. The newly issued DoD IPSO (Internet Protocol Security Option) provides one octet for the security classification of each datagram transmitted. The current scheme requires a Hamming distance of four between any two valid values. The determination of the values in this octet is a nontrivial problem and presents some interesting properties. This paper discusses the the complexity of the problem, its generalization and the properties of the octet value assignment.

1 Introduction

Distributed computing and computer networks are changing our culture and civilization. Computers and similar intelligent devices are continuing their prevalence in our society. These intelligent devices and computers, in general, operate more efficiently by using real-time data and information gathered from other sources and computers. This real-time gathering of information will increase the network traffic and will yield more heterogeneous messages existing in the network. Segregation of sensitive information is necessary. Sensitive information travelling from source to destination must provide some safeguard against unauthorized use or viewing. The Internet protocol supports such a safeguard by providing a security classification field. This security classification field is an eight-bit field in the header of the datagram. This eight-bit field is called an octet. We will examine some of the characteristics of this octet as well as properties of the octet value assignment.

The Internet Draft (AHWGIPSO-2 16Oct91) describes the entire security option. This option, which is a minimum of three octets, describes the length, classification and protection authority flags for the datagram. The security classification octet is specified for use to:

- Transmit datagrams from source to destination in a network which uses standard representation for security field.
- Allow addition security fields(optional) and the common security labels required by computer security models on the network.
- Validate the datagram as appropriate, for transmission from the source and delivery to the destination.
- Ensure that the route taken by the datagram is protected to the level required by all protection authorities indicated on the datagram. In order to provide this facility in a general Internet environment, interior and exterior gateway protocols must be augmented to include security label information in support of routing control.

The IP (internet protocol) declares that the datagram must be protected at the level of the security octet. Network use by the Department of Defense dictates that the security level of the datagram must not be ambiguous. How then are the security levels developed? The length of the field is specified by the draft as eight bits. This length conforms to many block transmission protocols as well as being a power of 2. It is therefore desirable to maintain the length as specified.

Each security code is developed to be Hamming distance d from all other valid codes. Given a code size of n, there are $O(2^{n-d})$ valid codes. Loosely stated, each code must differ from all other codes by d bit positions.

2 Properties

Table 1 lists the maximum number (upper bound) of valid codes that can be generated for a given Hamming distance d and code size n bits. The determination of valid codes using an exhaustive search is computationally expensive. For a given code(pattern), it must be examined against all other possible codes to determine usability. For example, given $n = 4$ and $d = 2$, there are 2^n possible codes. From these sixteen codes, a beginning(seed) pattern must first be selected. The seed is assumed to be in the solution set of valid codes. This seed code is exclusive-or(ed) with all remaining codes(15) to develop a set of possible valid codes that satisfy Hamming distance of 2. The exclusive-or operation will yield a 1 in bit positions where the seed code and possible valid code are different. The next code from the remaining set of possible valid codes is then selected as the seed code. This new seed code has already been verified to meet the solution set criteria and can now be used to disqualify other codes of the remaining set. The same procedure is followed as before, comparing the seed against the other codes left in the set of possible valid codes. The procedure is repeated until no codes remain in the set of possible valid codes. The output of an example of these intermediate steps for this procedure are given in the table below. In this example, $n = 4$ and $d = 2$. The remaining set of valid codes, that is, codes of size n that are Hamming distance d from one another, are presented in the last column of of the following table.

SET OF VALID CODES(upper triangle)

0000	0000	0000	0000	0000	0000	0000	0000	0000
0001		0011	0011	0011	0011	0011	0011	0011
0010	0011		0101	0101	0101	0101	0101	0101
0011	0101	0101		0110	0110	0110	0110	0110
0100	0110	0110	0110		1001	1001	1001	1001
0101	0111	1001	1001	1001		1010	1010	1010
0110	1001	1010	1010	1010	1010		1100	1100
0111	1010	1100	1100	1100	1100	1100		1111
1000	1011	1101	1110	1111	1111	1111	1111	
1001	1100	1110	1111					
1010	1101	1111						
1011	1110							
1100	1111							
1101								
1110								
1111								

SET OF POSSIBLE VALID CODES(lower triangle)

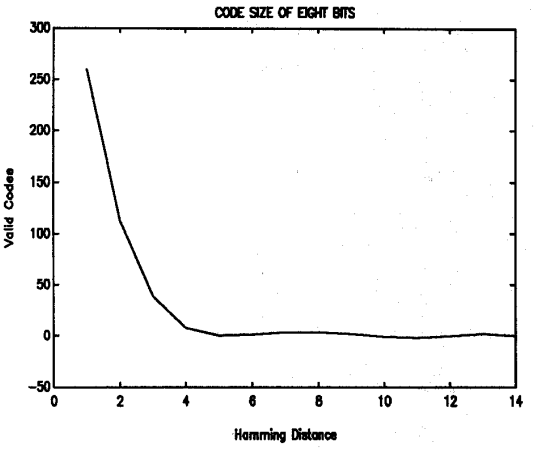

Figure 1: Number of Valid Codes vs Hamming Distance

To automate the generation of codes for examination, a computer program was developed. The algorithm for generating this set of valid codes is written in C. This algorithm was written to be portable and is ANSI C compliant. This C program will prompt the user for the starting code(seed), the desired Hamming distance d and required code size n. The program will display the valid codes and give the total number of valid codes found. Since Table 1 gives the maximum number of valid codes and all entries are powers of two, it is intuitive to investigate the determination of the maximum number as a power of two function. Subsequently, equations (1) through (3) are developed to model the maximum number of valid codes as a function of code size n and Hamming distance d.

$$g(d,n) = 2^{n-d+1} : d \in (1,2) \quad (1)$$

$$g(d,n) = 2^{n-d-1} : d \in (3,4) \quad (2)$$

$$g(d,n) = 2^{n-d-2} : d \in (5,6) \quad (3)$$

If $g(d,n) < 0$ then $g(d,n) = 0$. It is difficult to describe an accurate model for all code sizes. Even for the case where $d = 4$ and $n = 8$, a 5^{th} order polynomial (4) loosely approximates the number of valid codes.

$$g(d) = -0.01d^5 + \frac{1}{2}d^4 - 10d^3 + 82d^2 - 334d + 521 \quad (4)$$

The polynomial of equation (4) is represented graphically in Figure 1. The graph is for a code size n of 8 bits and is read by reading up from a given Hamming distance then across to the total number of

$d\backslash n$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
2	0	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
3	0	0	2	2	4	8	16	16	32	64	128	256	512	1024
4	0	0	0	2	2	4	8	16	16	32	64	128	256	512
5	0	0	0	0	2	2	2	4	4	8	16	16	32	64
6	0	0	0	0	0	2	2	2	4	4	8	16	16	32
7	0	0	0	0	0	0	2	2	2	2	4	4	8	16
8	0	0	0	0	0	0	0	2	2	2	2	4	4	8
9	0	0	0	0	0	0	0	0	2	2	2	2	2	4
10	0	0	0	0	0	0	0	0	0	2	2	2	2	2
11	0	0	0	0	0	0	0	0	0	0	2	2	2	2
12	0	0	0	0	0	0	0	0	0	0	0	2	2	2
13	0	0	0	0	0	0	0	0	0	0	0	0	2	2
14	0	0	0	0	0	0	0	0	0	0	0	0	0	2

Table 1: Maximum Number of Valid Codes

valid codes available. How then are the operations of determining the valid codes defined?

Formally, the $\oplus$ operation is associative and commutative. The exclusive-or operation is associative and commutative. Given a field of n-bits, it is Hamming distance d away if $c_1 \oplus c_2$ yields d number of ones. To assist in the examination of the octet values, a C program was developed. This program calculates all valid codes given a starting pattern of bits. The summation of this program is tabulated in table (1).

Lemma 1:
Given two distinct bit patterns P_1, P_2 and $P_1 \neq P_2$, and let $S_1 = \{P_{11}, P_{12}, P_{13}, \ldots, P_{1x}\}$ be the set of patterns that satisfy $distance(P_1, P_{1i}) \geq d$ where $i \in [1, x]$. Similarly, let $S_2 = \{P_{21}, P_{22}, P_{23}, \ldots, P_{2y}\}$ be the set of patterns that satisfy $distance(P_{21}, P_{2j}) \geq d$ where $j \in [1, y]$. Then $x = y$.

Proof
Since $P_1 \neq P_2$ let $P_1 \oplus P_2 = m$. $S_{1m} = \{m_{11}, m_{12}, \ldots, m_{1x}\}$ where $m_{1i} = m \oplus P_{1i}$. $\forall P_{1i} \in S_1$, there is a unique corresponding $m_{1i} \in S_{1m}$ such that $m_{1i} \geq d$. We can have a unique $P_2 \oplus m_{1i}$ and since $\mid m_{1i} \mid \geq d$ such that $d(P_i, P_i \oplus m_{1i}) \geq d$. The unique patterns of $P_2 \oplus m_{1i}$ should be in set S_2. Similarly, $\forall P_{2j} \in S_2$ we can produce a unique pattern $P_{1i} \in S_1$. The combination of (1) and (2) states that $x = y$. Since in the proof above, no assumption on the exact pattern of P_1 and P_2, therefore as long as we have two sets of solutions, both should be equal in size. Besides, we can use any pattern as the seed pattern to generate the solution set. □

Lemma 2:
Given a seed(starting pattern) P of $n - bits$, the size of the solution set S_p is bounded from above by $\sum_{D=d}^{n} C(n, D)$.

Proof

A masking pattern m that has k 1's can generate a new pattern p' from a given seed pattern p such that $d(p, p') = k$ by letting $p' = p \oplus m$. There are $C(n, d)$ possible masking patterns where each has d 1's in them. Therefore there may exist $C(n, d)$ patterns d away from the seed pattern p. In general, there may exist $C(n, w)$ patterns w away from the seed pattern p. In summary, the upper bound of the number of patterns that are at a distance greater than d away from p is the sum of all possible patterns p' that satisfy $d(p, p') \geq d$. The computational complexity of finding all patterns that satisfying $d(p, p') \geq d$ is bounded from above as $O(n^2)$ □

The field specification for the security octet states that the values used must achieve a minimum Hamming distance of 4 between and two valid codes. The maximum number of valid codes that can be generated for the given conditions, as specified by Internet Draft (AHWGIPSO-2 16Oct91), is 16. The starting code(initial valid code) has a great effect of the total number of valid codes. For instance, a starting code of 31 yields only 9 valid codes for a code size of 8 bits and a Hamming distance of 4 or greater. The valid code size drops dramatically if the Hamming distance is increased to a minimum of 5. The maximum number of valid codes is 4 and may be as low as 2 if the starting code is 127, 191, 223, 239, 247, 251, 253, 254

or 255.

3 Conclusion

Modeling the characteristics of the security option field of the IPSO is not trivial. The security option must account for the characteristics noted by this study. In particular, the specification must contain sufficient room for growth as network communications grow. In addition, ambiguity of classifications must not occur for proper handling of messages. Formal communications between computing devices is a rapidly growing field of engineering. Techniques and protocols to ensure proper interaction and reliability are continuing to development. The world, from an electronic perspective, is shrinking. Therefore, facilities must be available for private interaction between individuals and between organizations. The security classification option allows proper and private communications between such nodes. The protocols and techniques developed today must be forward looking to provide transition to the technology and social customs of the future. These protocols must support changes in information representation such as fuzzy logic. Our growing international society will demand accommodation for secure communications within groups and organizations such as the U.S. Department of Defense. The Internet Draft (AHWGIPSO-2 16Oct91) provides a suitable protocol for secure communications using the present technology.

References

1. Tanenbaum, A.S., *Computer Networks* -2nd ed, Prentice Hall, Englewood Clifs, New Jersey, 1988.
2. Chorafas, D.N., *Telephony: Today and Tomorrow*, Prentice Hall, Englewood Clifs, New Jersey, 1984.
3. Ermann, M. D. et al, *Computers, Ethics, and Society*, Oxford University Press, New York, New York, 1990.
4. Blahut, R.E., *Principles and practices of Information Theory*, Addison-Wesley, Menlo Park, California, 1987.
5. Stallings, W., *Data and Computer Communications* , Macmillan Publishing, New York, New York, 1985.
6. BYTE Magazine, *Wide Area Networking*, (vol 16, number 7), July 1991 pgs 158-190, McGraw-Hill, Peterborough, New Hampshire.
7. Comer, D., *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.
8. Hamming, R. W., *Coding and Information Theory*, Prentice Hall, Englewood Cliffs, New Jersey, 1986.
9. Clark, R. C., *Error-correction Coding for Digital Communications*, Plenum Press, New York, New York, 1981.
10. Schwartz, M., *Telecommunication Networks: Protocols, Modeling, and Analysis*, Addison-Wesley Company, Menlo Park, California, 1987.


```
#include "stdio.h"
#define TRUE 1
#define FALSE 0

unsigned int verifying, i, j, k, mask,
             codes[32767], d, stop;
unsigned int base, n, temp_code, d_check,
             valid_index, start_index;
unsigned int array_size, array_index,
             possible_good_codes[32767];

main()
{
 printf("\nEnter starting valid code (base 10) ==> ");
 scanf("%u",&k);
 possible_good_codes[0]=k; /* Load starting code */
 printf("Enter the Hamming distance (1-15)   ==> ");
 scanf("%u",&d);
 printf("Enter the number of bits (1-15)     ==> ");
 scanf("%u",&n);
 array_size = 1;
 array_size <<= n;
 base=array_size;
 valid_index = 0;
 start_index=0;  verifying = TRUE ;
 for(i=0; i<array_size; i++) codes[i]=i;

                  /* ALGORITHM FOR VALID CODES */
 while (verifying){
   for(array_index=start_index; array_index<array_size;
       array_index++){
     temp_code = ( possible_good_codes[start_index]
                 ^ codes[array_index] );
     mask = base;
     d_check = 0;  /* ### Reset code verifiers   ### */
     while(mask) { /* ### Verify code distance   ### */
       if (temp_code & mask)  d_check++;
       mask>>=1;
       }     /* ### Add valid codes if pass### */
     if (d_check >= d)
        possible_good_codes[++valid_index]
        = codes[array_index];
     }
```

```
   start_index++;
        /* ### Inc valid array limits ### */
   if (start_index>valid_index) verifying = FALSE ;
   valid_index++;
   for (i=0; i<valid_index; i++)
       codes[i] = possible_good_codes[i];
   array_size=valid_index;
             /* ### Prepare for next loop  ### */
   valid_index=start_index;
   }

/* PRINT CODES  */
  valid_index=0;
  printf("\n\n<<**** VALID CODES ****>>\n\n");
  for (array_index=0; array_index<array_size;
       array_index++){
   mask=base;
         /* ### Loop print valid codes ### */
   if ( possible_good_codes[array_index] < base){
    while (mask) { /* Print in binary format */
    printf("%c",
     (((mask & possible_good_codes[array_index]) && 1)+48));
   mask>>=1;
   }
   printf("\n");
   valid_index++; /* Count valid codes found */
     }
    }
printf("\nTOTAL NUMBER OF VALID CODES ==> %d\n",
       valid_index);

/*  PROGRAM END */
}
```