Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

A Scheme for Enhancing Transmission Rate on Asynchronous Secure Communication

Hyon-Cheol Chung
Electronics and Telecommunications Research Institute

Abstract

In asynchronous communication data of 00h ~ 1Fh are used as all kinds of control characters, and 7Fh also is used as control character in some protocol such as kermit. Therefore, data of this area must be converted to other form for transmission to prevent the misconception as control characters. This thesis presents several methods for character conversion that prevent the lengthening of data and enhance the overall efficiency of communication by transmitting with a certain conversion and without adding special characters on control-like characters occurring when data are transmitted with ciphering onto asynchronous communication path. For such conversion, the scope of transmitted data was supposed and efforts were made not to exceed that scope. Experiments showed that this method is better in communication speed than the existing ones and that ciphering has no problem by confirming the randomness of ciphered data.

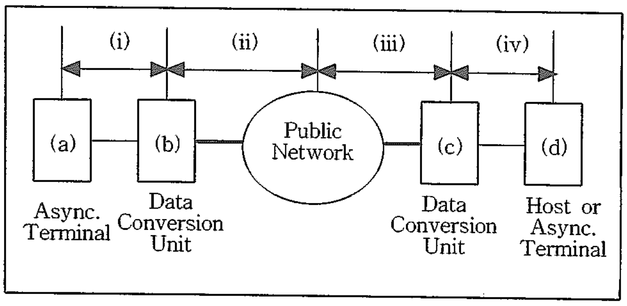
1. Introduction

The use of BBS(Bulletin Board System) by telephone line is sharply increasing recently due to the development of hardware/software and the propagation of personal computers. As most BBSs communicate using mainly asynchronous protocol, the use of asynchronous protocol increases continuously.

With the increase in the use of asynchronous protocol, the informations on it are in urgent need of protection. This paper refers to a scheme for processing control characters, one of the considerations in protecting informations on asynchronous data. Generally in asynchronous communication the data are transmitted in characters each of which consists of 7 or 8 bits. The characters those have the value of 00h ~ 1Fh or 7Fh in lower 7bits are control characters used for function or flow control in asynchronous protocol such as kermit. Characters with these code values can appear among binary data. Characters with the same value as control characters being not actually control characters [1][2][3], are defined as control-like characters. If control-like characters are transmitted as they are, they are recognized as control characters by communication nodes and let communication elements perform control operation resulting in the loss of data of one byte. Especially, in cases of cipher communication or the transmission of compressed data, many data can be lost in re-conversion of data due to data of one byte lost in communication. To solve this problem, these control-like characters must be converted to other form and the original data must be obtained by certain re-conversion on the receiver's side. Generally the method of executing exclusive- or(XOR) the data byte and 40h after the addition of '#' (23h) offered by kermit protocol is used a lot[1][2]. This method has the disadvantage that in the worst case too many of '#'s are added. Of course text data themselves do not have control-like characters but in case data are converted to cipher, control-like characters can not but occur in volume because text data become binary data, almost random numbers. Also in the case of binary data, although terminals already added '#' and processed control characters, if ciphering is performed at data conversion part they become binary data again so they must be processed in duplicate and the length of data gets longer.

Recently as the speed of processors gets faster, and pre-transmission processing such as ciphering causes no interference with the overall speed of communication or the rate of transmission. However, if one node on communication path receives data with a certain rate of transmission and sends them by attaching more data to them, many loads will hang on to a certain node and certain section as in (Fig. 1) regardless of the speed of the processor. And the frequency of flow controls occurring due to the overloads will decrease the speed of communication. This is because the lowering in the speed of communication can not be avoided for the rate of transmission on to the next node is constant no matter how fast the processor of that node processes the ciphering/deciphering of data. In order to solve this problem, this paper presents methods to enhance the transmission rate and to process without additional characters to the control-like characters occurring when data are ciphered and transmitted.

For efficient unfolding of the thesis, section 2 introduces methods of processing control-like characters in kermit protocol and describes problems in cipher communication, section 3 presents methods of processing control-like characters without additional characters by executing exclusive-or in nibbles, and section 4 presents methods of using the operation with the modular addition, not exclusive-or. Section 5 presents and analyzes the experiment for each method and its result and section 6 concludes.



(Fig. 1) Typical asynchronous communication network

2. Encoding control-like characters in kermit protocol

Kermit protocol is an asynchronous protocol designed for file transmission. This protocol is operated in procedures similar to XMODEM[2]. This protocol can process not only ASCII but also binary files and this section refers to the method of processing control-like characters at the time of transmission/reception of binary files.

If control-like characters appear while files are transmitted, kermit uses the method of transmitting by adopting value 40h and exclusive-or on them and then inserting '#' into the front. Generally control characters are defined in between 00h and 1Fh and they are converted to values of between 40h and 5Fh. And the other control-like character, 7Fh is converted to 3Fh by performing exclusive-or with 40h. However, as these values are overlapped with data of values between 3Fh and 5Fh, which can not be distinguished from original values between 3Fh and 5Fh on the part of receiver and so '#' shall be transmitted first before the converted value and then the applicable characters. This way the receiver obtains the original data by adopting 40h and performing exclusive-or for next characters right after removing '#' met in receiving data. This '#' also must be distinguished from the original '#' so that data will not be lost even though the receiver removes '#'. For this the transmitter shall transmit one additional '#' followed by the original '#' and the receiver shall adopt only '#' of even numbers in case '#' appears sequentially[1][2]. (Fig. 2) shows the process of processing kermit's control-like characters on binary data..

```
while(1) {
    Getch(byte);
    ch = byte & 0x7f;
    if (ch<0x20 || ch==0x7f) {
     byte = byte ^ 0x40;
     Send_char('#');
    }
    else if (ch == '#') Send_char('#');
    Send_char(byte);
}
```

(a) Processing Method on Sending

```
while(1) {
    Getch(byte);
    if (byte == '#') {
     Getch(byte);
        ch = byte & 0x7f;
     if (ch != '#') byte = byte ^ 0x40;
    }
    Accept(byte);
}
```

(b) Processing Method on Receiving

(Fig. 2) A method of processing control-like characters on kermit protocol

However, if the file to be transmitted is data the almost random numbers like ciphered data, the data to be transmitted/received increase too much due to added '#'. In other words, in case of 8 bit data, '#' must be added one by one on data '#', 7Fh and between 00h and 1Fh in lower 7bit. The section of control characters among 256 characters triggers the increase of data by the probability of 68/256. Also in case of 7 bit data, '#' must be added one by one on data '#', 7F, between 00h and 1Fh. The section of control characters among 128 characters results the increase of data by the probability of 34/128. This value means burden of additional transmission/ reception of data more than 1/4 in addition to the original data. If this communication is plain one not ciphered/deciphered it's no big problem as control-like characters are processed and then transmitted at the transmission part but in case of cipher transmission/ reception things are different. In a structure like (Fig. 1) the communication terminal(a) at the transmission part will transmit data after control-like characters are processed to data conversion part(b). At data conversion part the data must be ciphered and then control-like characters must be processed on data becoming random numbers again in the same way. At this time communication terminal transmits at a constant rate and if data conversion part increases these data by 1/4 and transmits the bottle neck phenomenon occurs. If this is the case, data conversion part(b) on the transmitter may have to flow-control continuously not to lose data, which in turn lowers the overall speed of communication greatly. Comparison of real communication speeds is presented in section 5.

3. Processing control-like characters using exclusive-or in nibbles

One of the most popular method of ciphering data by stream cipher is just exclusive-or with key stream[6][7][8]. The method in this section is similar to popular method in the sense that it uses exclusive-or with key stream. But, it is a little bit different with popular method in the sense that it treats data in nibbles. This section presents a method of transmission in combination with the original data value at the time of ciphering, which takes advantage of the points that text data do not have the value 7Fh and between 00h and 1Fh. It means that even the binary data reaching the data conversion part have been converted to process control-like characters at terminal, so the data reaching the data conversion part do not have the value 7Fh and between 00h and 1Fh in lower 7bits.

3.1 Method of executing exclusive-or in nibbles once

The data conversion part of (Fig. 1) performs exclusive-or to cipher data from the transmitter in byte with key stream created at the ciphering part. And if the value of lower 7bits is between 00h and 1Fh then lower 3bits of upper nibble are adopted as they are and perform exclusive-or with key stream for the rest bits. For the control characters in the range of 00h ~ 1Fh, the lower 3bits of upper nibble is 0h or 1h and that of lower nibble can be anything between 0h ~ Fh. In other words, the values not to become control-like characters, can only be larger than '1' in lower 3bits of upper nibble and be smaller than 7Fh in lower 7bits. As in pre-conversion characters the lower 3bits of upper nibble is 2h ~ 7h, the converted data are not control-like characters even if lower 3bits of upper nibble is adopted as it is at the time of data conversion. And as the lower nibble of converted data is a value converted by exclusive-or this byte itself is different from the pre-conversion value. For 7Fh of ciphered value, only first bit have to be ciphered. Same methods must be executed in ciphering as in deciphering. First, operation of exclusive-or on key and data is executed and then if the upper nibble of deciphered data is '0' or '1' the upper nibble of data prior to deciphering must be adopted. Because the source data can not be '0' or '1' in lower 3bits of upper nibble. For 7Fh of deciphered value, only first bit have to be deciphered.

In case this method is used, the length of data did not increase when text data were converted to binary and transmitted and so the flow-control at the transmitting part has sharply decreased and the overall rate of transmission has greatly increased. (Fig. 3) represents a brief algorithm of the method of processing control-like characters using exclusive-or in nibbles. Same algorithms can be applied in both transmission and reception.

```
while(1) {
  Receive_char(byte);
  Getkey(key);
  ch = byte ^ key;
  temp = ch & 0x7f;
  if (temp < 0x20) {
    ch = byte ^ (key & 0x8f);
    if ((ch&0x7f)==0x7f) ch = byte^(key&0x80);
  }
  else if (temp==0x7f) ch = byte^(key&0x80);
  Send_char(ch);
}
```

(Fig. 3) Processing control-like characters with single XORing

The above-mentioned method causes no problem in the speed of communication but there is a high probability that the upper nibble will concur with the original value of data compared to normally ciphered data. While the probability is 1/16 that the upper nibble of normally ciphered data concur with that of the original (in case the upper nibble of key is '0') using this method includes the case in which the upper nibble of ciphered data is '0' or '1' and so the probability becomes about 1/16 + 1/8. This makes the invasion from a third party look likely but it is hard to verify that it actually is. That is because the upper nibble of data appearing as control-like characters at the time of ciphering is not always mapped as a constant value but appears different depending on the value of source data. However, if the source data are binary which processed control-like characters it is no problem as the source data themselves have randomness, but in case of text data they can lose randomness as a certain range of data appear often. This problem can be solved by method presented in the next section.

3.2 Method of executing exclusive-or in nibbles twice

If the exclusive-or in nibbles is executed once by the method presented in the fore section, the probability that the upper nibble concur with source data is about 1/16 + 1/8. As the frequently appearing data are invariable in case of text data, if this method is used, ciphered data in which upper nibble and source data concur can appear in a certain area (e.g., 41h ~ 7Ah) in volume. If this happens, the series of ciphered data can lose randomness.

To solve this problem the method of adopting the same conversion method once again can be used. Namely, after exclusive-or is executed on ciphered data with another key stream. If the lower 7bits of this value is smaller than 20h, the upper nibble is adopted as it is and the lower nibble adopts ciphered value. And if the lower 7bits of this value is 7Fh, just only first bit have to be ciphered. In this method, the points that key stream must be created in double and that the operation of exclusive-or must also be in double work as overhead but in asynchronous communication of about 19,200 bps do not affect greatly as the speed of recent processors is fast. Deciphering in clause 3.1 must be processed twice in deciphering too. In case this method is used the probability that the upper nibble concur with that of source data is decreased to about $(1/16 + 1/8)^2$ = 9/256, which is lower than 1/16 of the method using normal exclusive-or and so the problem appearing in the method mentioned in clause 3.1 is solved.

As data once ciphered have randomness compared to source data, the ciphering method using exclusive-or twice has far better randomness than the one executing exclusive-or once. In the method executing exclusive-or in nibbles once the processes of transmission and reception are the same but in the method executing twice, during deciphering the second key must be used first among two keys. Because this method did not adopt simple exclusive-or but can go through different processes in nibbles.

For example, when 'key_1' and 'key_2' are 55h and 66h and 'ch' is 44h they are ciphered to 37h by this method. If 'key_1' is used first and deciphered it becomes 74h and so desired value can not be obtained. 'key_2' must be used first to obtain 44h.

4. Processing control-like characters using modular additions

The fore-mentioned methods use exclusive-or of key stream and data, generally used much in ciphering. The reason why exclusive-or is used much in ciphering is because it is fast in deciphering and convenient to program[6][7]. But in asynchronous communicationconsidering control characters the scopes of data must be compared each time and exclusive-or must be done twice on data of the probability 1/8 with this method. In addition to this, the method in which exclusive-or was done once on text data has some problem in randomness of ciphered data and the method in which it was done twice performs 6 exclusive-or's and 4 comparisons on one byte in the worst case and thus can have some overhead in performance speed. The ciphering by the use of operation with modular addition[6], which will be mentioned in this section, has little overhead in operation speed compared to these methods and no problem in randomness of data. Basic concept of methods in this section is that the source data can have the value in the range of 20h ~ 7Eh or A0h ~ FEh. So the number of elements of domain is 190(BEh). To cipher and decipher the data, refer to the following equation.

$$\text{Enc} = \text{remap}((\text{map}(\text{Src}) + \text{Key}) \bmod 190) \qquad (1)$$

$$\text{Src} = \text{remap}(\text{map}(\text{Enc}) + (\text{Count} * 190) - \text{Key}) \qquad (2)$$

If encrypted data are smaller than key in equation(2), increase count and compare before calculation. The encrypted data obtained in equation(1) are unique on constant source data and key and the source data having one of 190 values in equation(2) are unique on determined encrypted data and key. Therefore, ciphering/deciphering using this equation is normally performed. Function 'map' performs subtracting 20h or 41h from the argument, and function 'remap' performs adding 20h or 41h to the argument.

4.1 Method of organizing the domain of key stream with 256 elements

This method is to create the data of 00h ~ FFh with key stream for use to cipher data in byte and the brief description of the method of ciphering/deciphering is as follows.

First in ciphering as (a) of (Fig. 4) represents, deduct 20h or 41h from data and map the data to 00h ~ BDh. And add the mapped data into key value and then operate modulus with 190. Then this value has the value between 00h ~ BDh and if 20h or 41h is added to this value the data become ones without control-like characters.

In deciphering the process opposite to ciphering can be performed as (b) of (Fig. 4) represents. In deciphering, deduct 20h or 41h from ciphered value, and when this value and key value are compared, if the ciphered value is smaller, add 190. Deduct key value from this value and add 20h or 41h again to get source data. Here adding and deducting 20h or 41h is inserted for convenience to cipher and to decipher using modular addition.

```
while(1) {
     Receive_char(ch);
     if (ch < 0x80) ch = ch - 0x20;
     else ch = ch - 0x41;
     Getkey(key); /* 0x00 ≤ key ≤ 0xff */
     ch = (ch + key) mod 190 /* 0xbe */;
     if (ch < 0x5f) ch = ch + 0x20;
     else ch = ch + 0x41;
     Send_char(ch);
}
 (a) Conversion scheme on sending

while(1) {
     Receive_char(ch);
     if (ch < 0x80) ch = ch - 0x20;
     else ch = ch - 0x41;
     Getkey(key); /* 0x00 ≤ key ≤ 0xff */
     while(ch < key) ch = ch + 190;
     ch = ch - key;
     if (ch < 0x5f) ch = ch + 0x20;
     else ch = ch + 0x41;
     Send_char(ch);
}
 (b) Conversion scheme on receiving
```

(Fig. 4) Processing control-like characters with modular addition-(i)

4.2 Method of organizing the domain of key stream with 190 elements

The method mentioned in clause 4.1 is a good one in the safety of data and is better than the one introducedin section 3 in the complexity of calculation and is similar to that of clause 3.2 in randomness of ciphered data. However, the ciphered data do not have randomness as much as key stream because of redundancy of key stream. For example, two key values 00h and BEh produce the same ciphered data for one source data. This redundancy occurs on 66 key pairs(00h and BEh, 01h and BFh, ... , 41h and FFh) because of modulus 190. It means each key pair produce the same cipher data from same source data.

This section introduces the method of maximizing randomness from ciphered data by performing ciphering/deciphering using the procedure similar to clause 4.1 and the key created by making the domain of key stream 00h ~ BDh(189). It is similar to clause 4.1 in ciphering except that 'Count' of equation(2) is 0 or 1 in deciphering. As mapped source data and key of equations (1) and (2) is 0 ~ 189 each, the sum of the two data is 0 ~ 378. When this is operated with 190 by modulus two by two values fall between 0 ~ 189. As pair of value (0,190) is mapped to 0, (1,191) to 1 and (189,378) to 189, ciphered data have randomness as much as key. (Fig. 5) represents the method of ciphering/deciphering using key having the value between 0 ~ 189. The scope of key value can be set at the time of key creation but the case of using the algorithm of key creation between 0 ~ 255 is supposed and described.

5. Experiments and verifications

sections 3 and 4 reviewed the method of ciphering not to generate control-like characters whereas this section presents the measured speed of data transmission in cipher communication using the fore mentioned methods and represents the measured randomness of ciphered data to show using these methods that the ciphered data have not been weakened. The source data used in each experiment is a C program source of 1 Mbyte.

5.1 Communication speed for each method

In the environment of (Fig. 1) text data of 80Kbyte were transmitted in 9600 bps and 19200 bps for experiment. In this experiment (a) and (d) of both terminals in (Fig. 1) used 486DX and communication program used 'procomm'. Communication protocol used kermit and 8 bit data. Ciphering used stream ciphering and data conversion part used 8 buffers of 256 byte and when buffer was full or data forwarding character was input the ciphering transmission was used. The main processor of data conversion unit used MC68000.

```
while(1) {
     Receive_char(ch);
     if (ch < 0x80) ch = ch - 0x20;
     else ch = ch - 0x41;
     do {
          Getkey(key);
     } while(key >= 190)
     ch = (ch + key) mod 190;
     if (ch < 0x5f) ch = ch + 0x20;
     else ch = ch + 0x41;
     Send_char(ch);
}
```

(a) Conversion scheme on sending

```
while(1) {
     Receive_char(ch);
     if (ch < 0x80) ch = ch - 0x20;
     else ch = ch - 0x41;
     do {
          Getkey(key);
     } while(key >= 190)
     if (ch < key) ch = ch + 190;
     ch = ch - key;
     if (ch < 0x5f) ch = ch + 0x20;
     else ch = ch + 0x41;
     Send_char(ch);
}
```

(b) Conversion scheme on receiving

(Fig. 5) Processing control-like characters with modular addition-(ii)

As <Table 1> represents, kermit incurred tremendous lowering of speed. Theoretically, if data conversion part passed without ciphering in 45 seconds, transmission must be completed in 57 seconds, about 1/4 more when kermit is used but, many flow controls were ocurred due to bottle neck phenomenon in section (ii) of (Fig. 1), which dropped the communication speed further. Most of the methods presented in this paper did not drop the communication speed. Only Double XORed method dropped a little bit but if faster processor had been used the speed might not be dropped. This experiments have been performed 20 times. However, time measurement was manual and so the record was in seconds.

<Table 1> Comparison of communication speeds depending on each methods

Cypher/Plain — baud rate	Cypher				Plain (By-pass)
	Single XOR	Double XOR	Modular Addition	Kermit	
19200bps	45s	48s	45s	62s	45s
9600bps	85s	86s	85s	114s	85s

5.2 Randomness of key stream and of data ciphered normally

To show that key stream used in this experiment is random enough the most common methods for evaluation were used such as frequency test, serial test and poker-8 test and the significance level in each experiment was 5%. The frequency test is an experiment on how evenly '0' and '1' appeared per bit, serial test on if possible values on all sequential bits are evenly distributed, and poker-8 test on the possible values from data cut in 8 bits are evenly distributed[8][9][10]. The latter is same as chi-square test whose degree of freedom is 255. First it was experimented that key stream and normally ciphered data were random enough, to compare the randomness of key stream with that of series created by the each methods of processing control-like character. However, control-like characters were not processed to experiment and compare randomness. That is because, if control-like characters are processed, data appear between 20h ~ 7Eh and between A0h ~ FEh an appropriate value can not be obtained with experiment on randomness in bit or byte. <Table 2> represents the result of experiment on randomness of key stream using each experimental method and that of ciphered data without the processing of control-like characters.

As this result suggests, key stream and ciphered data have enough randomness.

5.3 Randomness of data ciphered using exclusive-or in nibbles

This clause experiments the randomness of data created by the ciphering method mentioned in section 3 and presents the result. As the data created by the method mentioned in section 3 are values between 20h ~ 7Eh and between A0h ~ FEh, the experiment on randomness in bit or byte bears no meaning.

<Table 2> Randomness test for key and cipher data

test / data	Frequency Test			Serial Test			Poker-8 Test		
	100bytes X 10000times	1000bytes X 1000times	1Mbytes X once	100bytes X 10000times	1000bytes X 1000times	1Mbytes X once	100bytes X 10000times	1000bytes X 1000times	1Mbytes X once
key stream	95.4%	96.5%	succeed	95.2%	94.0%	succeed	94.4%	94.5%	succeed
cipher data	94.6%	95.1%	succeed	94.0%	91.7%	succeed	93.0%	90.6%	succeed

Instead of the experimental method in clause 5.2, chi-square experiment measured randomness with degree of freedom 189[9]. Chi-square value V is obtained by equation(3) in which k is the kind of sample, Ys the frequency that s th sample appeared and nps the likeliness that s th sample will appear. In this experiment k is 190. (In reality s = 0 to 189.)

$$V = \sum_{s=1}^{k} \frac{(Y_s - np_s)^2}{np_s} \qquad (3)$$

In what the value obtained in equation(3) would be reasonable can be known by standard value obtained using the degree of freedom in the distribution chart of chi-square and v the degree of freedom is determined by k-1. In this experiment the significance level is 5% and v = 189. As <Table 3> represents, the method executing the exclusive-or in nibbles once passed only when randomness was verified in 100 byte, and long series had not enough randomness. In methods executing exclusive-or in nibbles twice, randomness was enough in verification not only in 100 byte but also in 1000 byte.

<Table 3> Randomness test for data produced by XORing Method

test / scheme	100bytes X 10000times	1000bytes X 1000times	1Mbytes X once
Single XOR	94.6%	50.5%	fail
Double XOR	94.2%	92.0%	fail

5.4 Randomness of data ciphered using operation with modular additions

This clause experiments the randomness of data created with the ciphering method mentioned in section 4 and presents the result. As the data created with the method presented in section 4 are values between 20h ~ 7Eh and between A0h ~ FEh, the experiment on randomness in bit or byte would be meaningless. So randomness was measured by chi-square experiment with the degree of freedom 189 as in the method using exclusive-or in nibbles. As <Table 4> represents, in case the scope of key is 0 ~ 255 it has enough randomness in verification of 100 but it failed in verification on randomness on series of 1000 byte and 1 Mbyte. The method with keys in scope of 0 ~ 189 passed all verifications on randomness indicating that key stream and randomness are almost equal. It is reasonable that randomness of latter method is better because of redundancy of key as I mentioned in clause 4.2.

<Table 4> Randomness test for data produced by Modular addition

test / key domain	100bytes X 10000times	1000bytes X 1000times	1Mbytes X once
0 to 255	90.5%	48.7%	fail
0 to 189	94.2%	95.5%	succeed

6. Conclusion

Binary data transmitted in asynchronous communication can have same code values as control characters and are called control-like characters. In order to perform protocol well these control-like characters must be converted to other characters to be transmitted. Generally, 40h and exclusive-or are executed on control-like characters and '#' and then they are transmitted with '#'. This method is defined in kermit protocol, in which many '#'s are added and transmission takes much time and thus the frequency of flow control becomes large dropping the efficiency of communication further, in case the data are ciphered and transmitted for they become almost random numbers. This paper introduces and analyzes the method of processing control-like characters provided by kermit and analyzes the problems occurring in cipher communication. As a solution to them, schemes that may enhance the efficiency of transmission in cipher communication by processing control-like characters without additional characters by presenting two methods of synthesizing key stream and data were presented. These took advantage of the characteristics that data sent from terminal to data conversion part do not have control-like characters.

The first one is for operation of key stream and exclusive-or in nibbles, the method in which upper nibble adopts the value of source data if the value from the operation of exclusive-or is control-like character. This method performs some more procedures than the one by kermit but that does not interfere with communication speed as the speed of processor is fast enough. Rather, experiment confirmed the fact that the overall speed of transmission is much faster in this method than in the one by kermit requiring additional data by 33/128 for it allows communication without additional data in a constant communication speed. As the data obtained by this method have weak randomness, it was taken care of by applying this method to data twice. It could be known that the case in which exclusive-or was executed in nibbles twice had enough randomness relatively compared to once and the loss of time following the increased operation was not big.

The second method is one by the use of operation with the modular addition, and took advantage of the characteristics that the source data from terminal do not have control-like characters. The source data can have values between 32(20h) ~ 126(7Eh) and between 160(A0h) ~ 254(FEh), and are ciphered by executing the addition with key and modulus with 190 after mapping source values to 0 ~ 189. This method does not yield to the first one in the aspect of time and is superior to it in the aspect of safety of ciphered data. It could also be known in particular, that when keys were in the scope of 0 ~ 189 they had randomness equal to that of key stream. As a result of each experiments the last method is the best in the repect of time and safety.

The methods presented as above did not interfere with safety of data but bettered the efficiency of communication. However, as it is hard to verify if ciphering by this method was more vulnerable to invasion than one by a normal method, and safety was demonstrated by measuring of randomness of ciphered data.

REFERENCES

[1] Frank da Cruz, *KERMIT - A File Transfer Protocol*, Digital Press, pp.206-239, 1987.

[2] Uyless Black, *Data Link Protocols*, PTR Prentice-Hall, pp.84-85, 1993.

[3] Young Il Chung, *Introduction to PC communication I*, Young Jin Publishing Co., pp. 43-85, Korea, 1988.8.

[4] In Tak Hwang, *PC Communication Protocol Handbook*, Kanamsa, pp.326-333, Korea, 1989.9.

[5] Hans-Georg Gohring, Erich Jasper, *PC-Host Communication-Strategies for Implementation*, Addison -Wesley, pp.1-35, 1993.

[6] Electronics and Telecommunications Research Institute(ETRI), *Modern Cryptology*, pp.1-102, 1991.

[7] R.A.Rueppel, *Contemporary Cryptology : The Science of Information Integrity*, IEEE Press, pp.65-134, 1992.

[8] Henry Beker, Fred Piper, *Cipher Systems - The Protection of Communications*, John Wiley & Sons, Inc. New York, 1982.

[9] Donald E. Knuth, *The Art of Computer Programming - Seminumerical Algorithms*, Addison-Wesley, vol.2, Second Edition, pp.38-113, 1981.

[10] I.J.Good, "On the Serial Test for Random Sequences", Ann.Math.Statist., vol.28, pp.262-264, 1957.