Managing Secure Communications With Multilevel Security and Restricted Character Set Translation

Chyan Yang and Chien-Chao Tsai, *Senior Member, IEEE*

Abstract—**Naval message traffic is transmitted with restricted character set, the files of which are optionally compressed. Often in this type of transmission, both character set translation and data compression can be used as add-on data encryption. In supporting the multilevel security management of access control in communication, this paper presents a possible implementation of master keys. The underlying basis of this key management scheme allows a set of keys to be maintained in either a floating or connected fashion, thus making the system tolerant to expansion. In doing this, it is discovered that a direct implementation for master keys is not possible as modulo arithmetic is required whereby only the arithmetic operations of addition and multiplication follow the commutative property—not division, where the results of modulo division are irreversible. As a solution to this problem, a recursive procedure of modulo exponentiation is employed via software which utilizes indexes. Along with this research, an algorithm has been implemented with restricted character set translation scheme and incorporated into a data compression program for military applications. The restricted character set translation algorithms investigated for the U.S. Navy are discussed in this paper. The application of character set translation will defer hardware changes, and the software implementation alleviates the need for expensive hardware.**

I. Introduction

MULTILEVEL security is a familiar scheme of classification in the national security hierarchy. It may partition subjects in levels of clearance, and divide objects into levels of classification. Examples of these levels are top secret, secret, confidential, and unclassified. This paper reports one implementation that supports multilevel security with master keys, and is suitable for naval message traffic by character set translation and data compression [1]. An obvious application of this scheme is file transmission or storage. When users require shared access to secure data and files, it is convenient to partition the files into several classes and encrypt each class individually enforcing the security classification. A key management problem can be avoided by providing a master key to permit access to the required classes.

A shore-based system includes a large database which consists of relational tables of ASCII data in a commercial RDBMS (Relational Database Management System) as well as associated ASCII text and binary (graphic) files. Packages of data are prepared from the database for subsequent delivery to remote systems via floppy disks, electronic network, or via standard naval message traffic. Each prepared data package consists of a combination of ASCII and binary files grouped together in the standard hierarchical file storage structure of the host system. After either physical or electronic delivery of the data package, it resides within the file storage of the remote system.

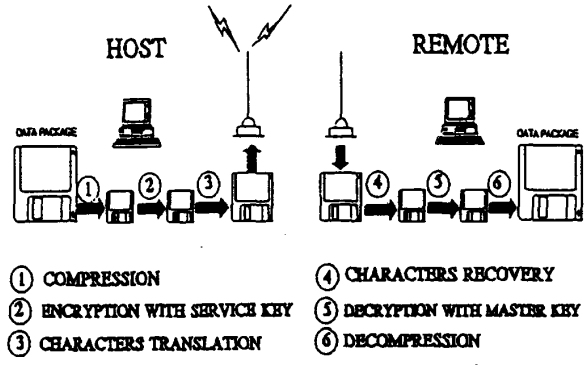


Fig. 1. Scenario of information transmission on naval message traffic.

The process of data compression is followed by data encryption prior to passing the processed file on to the character translation process. The entire scenario is then repeated in the reverse direction at the remote site as shown in Fig. 1. Data encryption is an optional requirement that allows different encryption schemes to be chosen. Without a hardware encryption chip available to us, DES (Data Encryption Standard) [2] has been implemented by software to incorporate the master key routines. As an unclassified research, this paper uses the encryption routine in UNIX.

Although designed for a different purpose, all three methods used in this research contribute to data security: data compression, data encryption, and character translation. When all methods are employed, the overall data security is greatly enhanced, since the probability of decoding by adversaries is the product of probabilities in breaking each process individually.

Based on number theory, the master key ideas proposed by Chick and Tavares [3] are straightforward. However, the implementation on a finite word length computer presents problems since most commercially available computers do not support extremely long digits, say, 70 digits. Additionally, to compound the implementation problems, if the local service keys and master keys are provided after taking the modulus, the proposed master key scheme [3] cannot work! This is

Manuscript received May 1992.

C. Yang is with the Institute of Management Science & Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan, Republic of China.

C. Tsai is with the Military Integrated Communications Agency, Ministry of Defense, Taipei, Taiwan, Republic of China.

IEEE Log Number 9206687.

because a result from modulus operations is irreversible. The restricted character set translation algorithms investigated for the U.S. Navy are discussed in this paper. The application of character set translation will defer hardware changes, and the software implementation alleviates the need for expensive hardware.

Section II discusses the incorporation of master keys for supporting multilevel security systems and data encryption. Section III shows some examples. Section IV explains the software implementation of master keys. Section V introduces the basic algorithms for character set translation. Section VI gives the improved version of the translation algorithm known to achieve less than 50% expansion ratio. The concluding remarks are given in Section VI.

II. The Master Key Scheme

A. Master Key System

Whether compressed or in original form, when a data package is transmitted to remote systems, and if encryption is additionally requested by a user, the access control of the encrypted package poses some reasonable concerns. In this section, a master key scheme is introduced to address the concerns.

A group of objects such as data packages may form a hierarchy or levels of security. These levels may relate to the objects as classification or to subject as clearance. The brief overview of master keys in this section is based on the work of Chick and Tavares [3] which is an improvement of [4].

In the following discussion, *service* is a file or a directory. For each service, there is a key (*service key*) controlling the access to the service. The $\leq$ indicates a partial order subordinating relation. It is assumed that within a master key system, we have a total number of N services reserved; N is the largest number of services allowed in the system. In our implementation, N is set to a number larger than the current system size so that the system is allowed to grow up to N without changing the keys. Each service S_i is assigned a service key SK_i. If $S_i \leq S_j$, then service (an object) S_i is subordinated to S_j and access to S_j guarantees access to S_i. Furthermore, each service is assigned a small prime p_i, but no primes are assigned to the Central Authority (CA) or master keys user. Let

$$T = \prod_{n=1}^{N} p_n.$$

For each service, a number u_i is defined as

$$u_i = \prod_{S_n \leq S_i} p_n$$

and the service key is defined as

$$SK_i = K_0^{T/u_i}$$

where K_0 is a random key number chosen by the Central Authority.

The master keys can be made by the following mechanism. First, v_j is computed as

$$v_j = \prod_{SK_n \leq MK_i} p_n$$

where the set $\{SK_i \leq MK_j\}$ consists of all the keys for services accessible with master key MK_j. The master key is defined as $MK_j = K_0^{T/v_j}$.

The computation of a service key from a master key is then

$$SK_i = K_0^{T/u_i} = (K_0^{t/v_j})^{v_j/u_i} = MK_j^{v_j/u_i} \quad (1)$$

which is valid if and only if $SK_i \leq MK_j$.

B. Flexibility of Master Key Scheme

1) The Notation of Master Key: A master key is a compact representation for a subset of the service keys; it can be assigned to authoritative users are needed. For any master key $MK, SK_i \leq MK$ for one or more SK_i. A trivial case is $MK = SK_i$. To provide a master key for each of N services, the master key space may contain up to $2N - 1$ members.

2) Prohibition of Nonmaster Key Intrusion: Computation of a service key is feasible if and only if $SK_i \leq MK_j$. If $SK_i \leq MK_j$, then, by definition, all primes in u_i must be included in v_j ; thus, u_i divides v_j. $(MK_j)^{v_j/u_i}$ is easily computed as in (1) since v_j/u_i results an integer. When $SK_i \leq MK_j$, the access is denied as follows. Let $u_i = \alpha p_i$.

$$MK^{v_j/u_i} = [(MK_j)^{v_j/p_i}]^{1/\alpha}. \quad (2)$$

Where p_i does not divide v_j, and the p_i^{th} root of MK_j must be computed, the computation of the r^{th} roots mod M for $r > 1$ is believed to be as difficult as factoring M [3]. So when p_i does not divide $v_j, MK_j^{v_j/p_i}$ cannot be computed if the factors of the modulus are unknown. This prohibits the unauthorized access by making M as large as possible.

3) Prohibition of Grouped Intrusion: The master key is also secure against illicit cooperation where a group of lesser privileged users may have sufficient information to accomplish tasks that none is capable of individually. A sufficient condition is that no group of master keys can be used to gain access to additional services. That is, from a group of master keys, one cannot create a key MK such that $SK_i \leq MK$ if none of the keys in the group has access of service S_i. This has been proven in [3] since p_i is not a factor of v_j in any master key of the group.

4) Expansion Capability: Chick and Tavares also proved that it is possible to add services to the system without affecting existing keys, provided that a new addition is not subordinate to any existing service. This addition chooses a new prime p_{N+1} which is relative to M and not previously assigned to a service. Hence, a newly added service will introduce an equation to compute SK_i as

$$SK_{N+1} = (K_0')^{T'/u_{N+1}} \quad (3)$$

where

$$K_0' = (K_0)^{1/p_{N+1}}, T' = T \times p_{N+1}, u_{N+1} = \prod_{S_n \leq S_{N+1}} p_n$$

and SK_{N+1} is the new service added.

Although the keys are unchanged by the substitution of T' and K_0', a new problem arises in that the number of services to be added is constrained by the primes relative to M. For instance, if $M = p_1 \times p_2$, then the maximum number of service keys that can be expanded is 2. In addition, since the new prime p_{N+1} is not a factor of v_j for any of the existing master keys, new master keys have to be redistributed by the CA to accommodate the $\leq$ relationships.

Notice that the T value is the product of all primes, thereby making system expansion inflexible. Either T must be fixed or all key numbers have to change accordingly since T/u_i or T/v_j provides the power of SK_i, MK_j. In our experiment, we introduce another method for master key system expansion. Numbers of individual services are assigned in the beginning when a system was built, assuming the future expansion is reserved. These services can originally be either $\leq$ or not be $\leq$ to any service, but primes are still assigned to them as usual. Thus, the T value will not be affected when anyone on these services is assigned to be a new service key, and the new service key can be inserted (or activated) in between two existing service keys or under any master key as required. The only values which must be modified are the u_i and v_j that were affected when insertion occurred. In this respect, no master key has to be redistributed when the system is expanded. Examples of system expansion are provided in the next section.

III. Examples

A. System Setup

To implement the concept of the master key scheme on compressed and/or encrypted data packages, we constructed a multilevel hierarchy of 20 services as illustrated in Fig. 2. These 20 assigned services are indicated as numbered circles in Fig. 2. Examples in breadth-first order are services 52, 48, 49, 51, 34, $\cdots$. A node with a darker boundary in Fig. 2 is an empty or reserved service that is preconnected for system expansion needs. Examples of preconnected services are 53, 50, 42–47, 24–33, 00–11. Additionally, there are ten floating or independent services (S_{54}–S_{63}) reserved for file system expansion. The floating services can be added freely, whereas the preconnected services are confined by their neighboring services. Notice that since modulus operation is performed during arithmetic computation, more empty services are considered tolerable. The number of empty services preconnected or independently prepared can be chosen according to future system expansion needs. The multilevel hierarchical in Fig. 2 shows a simplified example.

The assignment of primes in [3] uses the following rules.

1. If $S_j \not\leq S_i$ and $S_i \not\leq S_j$, then $p_i \neq p_j$.
2. If $S_i \leq S_j$, then $p_i = p_j$ is allowable and does not conflict with rule (1).
3. p_i is the smallest allowable prime.

The purpose of the above rules is to constrain the value of v_j or u_i as small as possible. This will not be necessary if modulus operation can limit all product value within M. Besides, a product of small primes may generate a value less than M. For this case, the modulus operation becomes useless and service can be easily invaded. Because of this, we generated 64 different prime numbers beginning from 59, and randomly distributed them to each service as shown in Table I. Notice that p_{ij} is the prime in column i and row j. For example, $p_{36} = 181$.

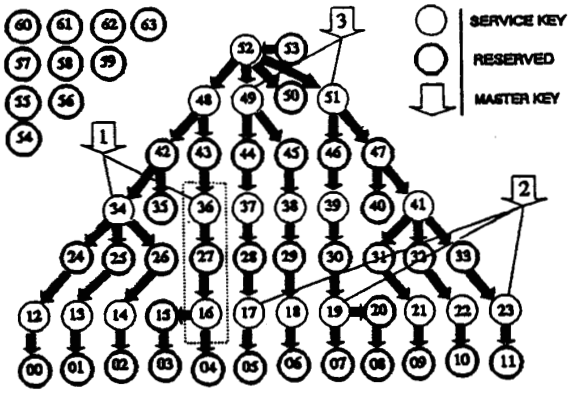


Fig. 2. A multilevel services hierarchical model.

TABLE I
Assignments of Prime Numbers

	0	1	2	3	4	5	6
0	307	241	223	251	89	227	353
1	73	277	83	79	281	359	229
2	293	383	131	97	127	367	137
3	173	211	167	179	373	193	191
4	163	283	379	409	197	311	
5	349	239	199	257	233	67	
6	263	157	103	181	101	347	
7	389	107	269	313	139	109	
8	397	151	271	61	149	401	
9	331	71	59	317	113	337	

Because T contains all primes in Table I, whenever services are added, the value of T remains unchanged. With a table of prime numbers, one can use the indexes to represent the primes instead of using actual primes. By doing so, one can avoid the finite word size of a computer in many operations (see Section IV). The expansion example will be discussed in Section III-D. As mentioned in Section II, the maximum master key numbers can be assigned up to $2^{19} - 1 = 524,287$. The three master keys MK_1, MK_2, and MK_3 shown as white arrows in Fig. 2 are simply examples to accommodate the implementation. Each master key superior to services can be indicated by a line. For example, MK_2 has lines going through S_{17}, S_{19}, S_{23} and, therefore, $SK_{17} \leq MK_2, SK_{19} \leq MK_2, SK_{23} \leq MK_2$. Each service can be treated as an encrypted data package, and its u_i is encrypted in its file header. To access (decrypt) an encrypted package S_i from S_j or by MK_j, the subordinating relationship of either $S_i \leq S_j$ or $S_i \leq MK_j$ must be satisfied, respectively.

In Fig. 2, the $\leq$ relationships among services are shown by arrows or lines (master key). For example, $S_{12} < S_{34}, S_{52} < S_{53} \cdots$, etc. The partial ordering $\leq$ relationship is assumed

to have a transitive property. Therefore, a master key that can access a service S_i can also access all the services inferior to S_i. For instance, if the master key $MK_1 \geq SK_{34}$, then the MK_1 can access $SK_{12}, SK_{13}, \cdots$ as well, since $SK_{12}, SK_{13}, \cdots$ are connected and are $\leq SK_{34}$.

Note that, in this section, the arithmetic is performed in modulo M for some integer M. Values are operated in the ring of integers $(0, M-1)$ [3], and M is defined by $M = p_1 \times p_1 \times p_2$ for some large but not-assigned primes p_1 and p_2 to meet the expansion needs. Since we implement a different approach for system expansion, we are free to choose M. Let us arbitrarily choose a 6-digit $M = 524\,287$ and $K_0 = 1992$ to accomplish key number computation. All large key number computations in the following subsections were done by a password verification program to be discussed in Section IV-A-2. In the following discussion, we use $\text{mod}(x)$ as a shorthand for x mod M, so that $\text{mod}(\text{mod}(p \times q) \times r)$ actually means $(((p \times q) \text{mod} M) \times r) \text{mod} M$.

B. Example of Access Control

As an instructive example, let us arbitrarily pick S_{36} as an encrypted data package to access. Fig. 2 shows $SK_{36} \leq MK_1, SK_{36} < SK_{48} < SK_{52}$, and $SK_{36} > SK_{16}, \cdots$, etc. Hence, there are four master keys that can access S_{36}: $\{MK_1, MK_{52}, MK_{48}, MK_{36}\}$. Let this set be named S_{MK}.

The number u_{36} is computed as $u_{36} = \Pi_i p_i$, where $i = 03, 04, 15, 16, 27, 36$, or $u_{36} = 173 \times 163 \times 239 \times 157 \times 269 \times 181$.

Now the service key for S_{36} is

$$SK_{36} = K_0^{T/u_{36}} = \text{mod}(1992 \Pi_{n=0}^{63} p_n)$$

where $n \neq i$ or

$$SK_{36} = \text{mod}(1992^{307 \times 73 \times 293 \times 349 \times \cdots \times 137 \times 191}) = 50\,199.$$

We now show how the master key MK_1 can access S_{36} by making SK_{36} from MK_1. The value of v_1 has to be computed first. $v_1 = \Pi_j p_j$ for $j = 00$–04, 12–16, 24–27, 34, 36, or $v_1 = 307 \times 73 \times 293 \times 173 \times 383 \times 211 \times 283 \times 239 \times 157 \times 379 \times 199 \times 103 \times 269 \times 409 \times 181$. Therefore, $MK_1 = K_0^{T/v_1} = \text{mod}(1992 \Pi_{n=0}^{63} p_n)$ for $n \neq j$, or $MK_1 = \text{mod}(1992^{349 \times 263 \cdots 137 \times 191}) = 172\,955$. With MK_1 one can derive SK_{36}:

$$MK_1^{v_1/u_{36}} = \text{mod}(172\,955^{p_{00} p_{01} p_{02} p_{12} p_{13} p_{14} p_{24} p_{25} p_{26} p_{34}})$$

or

$$\begin{aligned} MK_1^{v_1/u_{36}} &= \text{mod}(172\,955^{307 \times 73 \times \cdots \times 103 \times 409}) \\ &= 50\,199 \\ &= SK_{36}. \end{aligned}$$

In other words, MK_1 can access service S_{36}.

C. Example of Intrusive Prevention

On the other hand, let us examine whether $MK3$ (it is not in set S_{MK}) can access S_{36}. Since $v_3 = \Pi_j p_j$ for $j = 05$–11, 17–23, 28–33, 37–41, 44–47, 49, 51 or $v_3 = 349 \times 263 \times 389 \times \cdots \times 113 \times 359$, and for $n \neq j$ we have

$$MK_3 = K_0^{T/v_3} = \text{mod}\left(1992 \prod_{n=0}^{63} p_n\right)$$

or

$$MK_3 = \text{mod}(1992^{307 \times 73 \times 293 \times 173 \cdots \times 227 \times 367 \times 191}) = 270\,495.$$

To make the service key SK_{36} from MK_3, one may try the following:

$$MK_3^{v_3/u_{36}} = \text{mod}(270\,495^{v_3/u_{36}})$$

or

$$\begin{aligned} &MK_3^{v_3/u_{36}} \\ &\quad = \text{mod}(270\,495^{p_{05} \cdots p_{11} p_{17} \cdots p_{49} p_{51} / p_{03} p_{04} p_{15} p_{16} p_{27} p_{36}}). \end{aligned}$$

Although MK_3 is a master key which can access more than ten services, none of the primes in v_3 includes p_{36}. Since p_{36} does not divide v_3, MK_3 cannot access S_{36}. Similarly, one can show that all master keys in set S_{MK} can access S_{36}, but none of the others can. Additionally, it is shown that a master key formed from any group of master keys not in set S_{MK} will also be unable to access S_{36}, since $p36$ does not divide into any v of them. This outcome prevents grouped intrusion.

D. Example of System Expansion

1) Adding Single Service: One possible example of adding single service is a new service S which is needed between S_{36} and S_{16}, such that $S_{16} < S < S_{36}$. Examining the dashed box in Fig. 2, we see that this task can be done by simply letting S_{27} be the new added service. By definition,

$$\begin{aligned} v_{27} &= p_{03} \times p_{04} \times p_{15} \times p_{16} \times p_{27} \\ &= 173 \times 163 \times 239 \times 157 \times 269. \end{aligned}$$

The service key SK_{27} can be formed as

$$SK_{27} = K_0^{T/v_{27}} = \text{mod}\left(1992 \prod_{n=0}^{63} p_n\right)$$

where $n \neq 03, 04, 15, 16, 27$.

$$SK_{27} = \text{mod}(1992^{307 \times 73 \times 293 \times 349 \cdots 137 \times 191}) = 347\,497.$$

All master keys and service keys in the system will not be affected by this expansion, and $MK_1, MK_{36}, \cdots$ can access S_{27} as desired.

2) Adding Multiple Services: This subsection explores the case when more than one service expansion is required. For example, two services are added between S_{36} and S_{16} such that $S_{16} < S_a < S_{36}$ and $S_b < S_a$. The first additional service $S_a(S_{27})$ is achieved in the same manner as the previous example. Since no empty remaining service is related to S_{27}, the second service S_b can be constructed from those independent empty services, say, S_{54}. The relationship of new services is shown in Fig. 3. In this example, for all services that $< S_{27}$, the service keys will remain the same. The only key numbers that have to be changed are MK_1, SK_{36}, SK_{48}, and

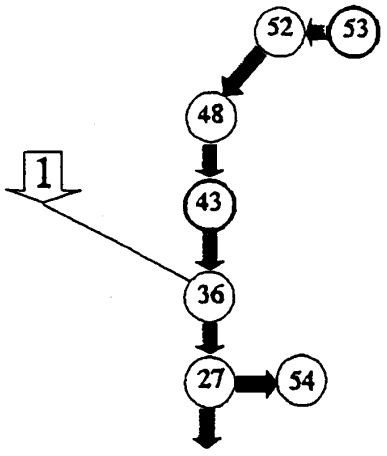


Fig. 3. Example of service key expansion.

SK_{52} (see Fig. 3). This modification is easily done by adding p_{27} and p_{54} to their respective v and u values; these values are in turn redistributed by CA. Consequently, to remove one or more existing services from the system, one can reassign them to be empty services without influencing the whole system. Overall, system expansion is simple and flexible.

IV. Software Implementation

Recall the scenario of the Naval message traffic described in Section I. If data encryption is required, it must be done at the host system, after data compression and before characters set translation. In software implementation, a user may provide the password (key) for encryption after data compression (at host system) or data recovery (at remote system). Given a correct key, the program will encrypt or decrypt the file; otherwise, it assumes no data encryption.

In the next two subsections, we will discuss how passwords are verified in the master key access control environment, and low data encryption is implemented.

A. Master Key Access Control

Access control is divided into two steps: password generation and password verification. Password generation is performed at the CA and distributed to individual services or master key users. When accessing a director or file, password verification is performed for authorization check.

1) Password Generation: To facilitate the friendly use of keys, passwords are used as keys. Printable characters are used as passwords instead of numerical values. Central Authority generates passwords by certain hashing methods or any translation scheme. From the examples in Section III, we need two parameters from the user for key number computation: v_j and MK_j. This signifies that a user's password is formed by these two values. For better security, the users of master keys receive ID numbers instead of MK_j's. The ID numbers can be any hashing function of MK_j. Theoretically, the computation $SK_i = MK_j^{v_j/u_i}$ is straightforward, but the finite length of a computer calls for modulo exponentiation whenever possible. In other words, when v_j and u_i are not of their true values but rather the values after modulo exponentiation, the result may include a noninteger v_j/u_i value (it should be an integer). For this reason, we have chosen the modulus operation to be in MK_j or SK_i (to be discussed in the next subsection). The value of MK_j will be in [0, 524 286], and the v_j's will be represented by a set of indexes to Table I. Table I is available in the executable program and can be coded in any encryption scheme.

Password generation can be done in many ways. We present here one method that can be easily implemented. Each password has 10 printable characters, and each character is formed by one row of Table I. First, the ID number is the decimal complement value of MK_j. For example, if $MK_j = 524\,287$, then the ID number is 586 823. Second, a password is formed by examining the inclusion of primes in v_j of a service key. For example, v_1 is the product of p_{00} —p_{04}, p_{12}–p_{16}, p_{24}–p_{27}, p_{34} and p_{36} (see Table I). For v_1, the primes in row 0, only p_{00} is used so that the first row can be thought of as 100000 if the last column is not used. The bit pattern 100000 can be arbitrarily assigned a printable charactero f. Similarly, the 10-character password of master key user number 1 can be constructed as 'ffvv8XbH//' ('/' means no primes related in that column). When accessing an encrypted package, a user will be asked to provide the ID number first and then the password.

At the host system, before encryption, a user's password (private key) is validated to avoid the data package being encrypted or scrambled by an invalid key which cannot be decrypted later. Verified passwords and an ID number of length 16 bytes are themselves encrypted and embedded in the file header starting at the fourth byte. At the remote system, the received file header will be examined to see if it is an encrypted file. If it is encrypted, the 16 bytes starting at the fourth byte in the file header will be used for service key number conversion. To access a service, a user must provide the ID number and password for key computation.

2) Password Verification: In using the master key, it is assumed that there is no password distributed electronically, and the access is done by key number computation. The master key numbers are not necessary to be the same as service key numbers before computation. Therefore, different master key numbers may result in the same service key number based on a unique v_j.

Recall the discussions in Section III—the modulus operation was taken in each arithmetic operation. While implementing in software, special care must be taken that a modulus operation will be performed as often as possible because if it is necessary to take a modulus on an overflowed number, i.e., larger than the largest mantissa in the floating number system, the operation is done too late and is therefore incorrect. Without concatenation of memory words, the product result will be truncated and become useless before the modulus process because the Floating Point Number System allows only limited digits (Mantissa, or significant) representation, (e.g., IEEE double precision only has 53 bits [5]). An example here is

the T value by definition:

$$T = \prod_{n=0}^{63} p_n = 307 \times 73 \times 293 \cdots \times 137 \times 191.$$

The result is a number that has more than 70 digits and thus cannot be easily represented. Moreover, since the product of primes will be used as the exponent of K_0, the existing Floating Point Number System cannot support such a large value. To solve this problem, a recursive procedure of modulo exponentiation is implemented in software. Modulus operation begins at first K_0^p, and it repeats in each multiplication until done. The operation restricts each result in the range [0, 524 286]. Now (1) becomes

$$\begin{aligned} SK_i &= \mathrm{mod}(K_0^{T/u_i}) \\ &= \mathrm{mod}(K_0^{\Pi_{n=0}^{63} p_n / \Pi_{S_n \le S_i} p_n}) \\ &= \mathrm{mod}(K_0^{p_a p_b \cdots p_n}) \end{aligned}$$

where $S_a, S_b, \cdots, S_n \not\leq S_i$. That is,

$$SK_i = \mathrm{mod}(\mathrm{mod}((\cdots \mathrm{mod}((\mathrm{mod}(K_0^{p_a}))^{p_b}) \cdots)^{p_n})).$$

Furthermore, to prevent a large prime number exponent, any inner factor above can be computed as

$$\mathrm{mod}(m^p) = \mathrm{mod}(\cdots \mathrm{mod}(\mathrm{mod}(m)_1 m)_2 \cdots m)_p$$

In other words, the computation of $\mathrm{mod}(K_0^{p_a})$ needs p_a times modulo exponentiation, and the total of modulo exponentiation in computing SK_i would be $p_a + p_b + \cdots p_n$.

Key number computation is implemented only at the remote system immediately after password conversion. If the service key computed from the master key equals the service key in the file header, the program will allow access using the header password. The algorithm for password verification and key number computation is shown below.

```
loop: request user to enter password;
 if  (user password == header password)
     do decryption; /* for encrypted user */
  else{ if (length of password != 10)
        {
        if(third entry) recover original file and exit;
        else display error message and go to loop;
        }
      }
  else
       covert user password to master key number;
  covert header password to service key number;
  key computation intialization;
  if  (Master Key user)
      {compute v using modular exponentiation;
       compute service key from master key;
      }
  else /* the case of superior service keys */
      {compute v using modular exponentiation;
       compute service key from superior service key;
      }
  if  (computed serivce key == service key number)
       do encryption;
  else
      { if(third try) receover original file and exit;
        else display error message and go to loop;
        }
```

Note that each encrypted file has a header which contains the encrypted password. The encrypted password is converted to a service key number and will be used to match the computed service key. If it matches, the access is allowed. The computed service key is based on the modulo exponentiation [6] such that $SK_i = K_0^{v/u_i}$, where u_i is encrypted in the file header.

B. Data Encryption

Encryption on a compressed data package is an option to the user who may specify in the command line when executing the software at the host system. If requested, the program will process after the password is verified. At the remote system, the program will automatically get the first 3 bytes in the file header to check if it is an encrypted file and verify the key. Notice that encryption algorithms are varied from user to user and so are the password encryption of a file header and key number conversion. They can be implemented in different schemes to meet the data security requirements of, for example, the DES system. After all, regardless of the encryption method, a master key scheme must be able to properly drive the access control.

In this experiment, the encryption routine in UNIX is used (the key generation part from "Makekey" has been modified to master key password conversion). It is a one-rotor machine encryption algorithm designed along the lines of Enigma, but is considerably trivialized: encryption and decryption use the same keyword. Each encrypted package has an extra 19 bytes of encryption header, which includes 3 bytes for encryption and 16 bytes for password (it is the keyword to encrypt, too). The encrypted data package must be decrypted before it can be decompressed. To decrypt a package, a shift operation is used to decrypt the 16-byte keyword for the encryption key which in turn decrypts the compressed data stream if the password has been verified.

V. Restrict Character Set Translation

Since Naval message traffic uses the restricted character set listed in NTP3 Annex C [7], the compressed and/or encrypted package has to be translated using this restricted character set. Character set translation can be used as an add-on encryption scheme. Besides, many reliable telecommunication systems, e.g., those which use the Morse code restricted character set, can continue to be used when supported by a good translational scheme. Therefore, the character set translation is not limited to military applications.

A restricted character set of N symbols can be represented as

$$C = \{\alpha_1, \alpha_2, \cdots \alpha_N\} \quad (4)$$

TABLE II
ASSIGNMENT OF 45 RESTRICTED CHARACTERS

Sequence	Binary	Character	Sequence	Binary	Character
0	00000	.	23	10111	E
1	00001	/	24	11000	F
2	00010	0	25	11001	G
3	00011	1	26	11010	H
4	00100	2	27	11011	I
5	00101	3	28	11100	J
6	00110	4	29	11101	K
7	00111	5	30	11110	L
8	01000	6	31	11111	M
9	01001	7	32	100000	N
10	01010	8	33	100001	O
11	01011	9	34	100010	P
12	01100	:	35	100011	Q
13	01101	;	36	100100	R
14	01110	<	37	100101	S
15	01111	=	38	100110	T
16	10000	>	39	100111	U
17	10001	?	40	101000	V
18	10010	@	41	101001	W
19	10011	A	42	101010	X
20	10100	B	43	101011	Y
21	10101	C	44	101100	Z
22	10110	D			

where $N \leq 256$. The Naval message traffic allows only 45 restricted characters. Apparently, there must be some data expansion in the character translation process. However, the electrical update to the fleet via existing communications channels makes this a must. Note that if the character translation algorithm is unknown, the translation itself can serve as an additional encryption. In the following subsections, we will analyze the expansion ratios as well as the software implementation regarding the restricted character set translation.

Without loss of generality, it is assumed that the restricted 45 characters are in the contiguous decimal value range [46, 90] (ASCII "." to "Z"). Actually, the characters < and > are not used in NTP3; instead, the characters { and } are used. Table II shows the character assignment. In a source package, without character set translation, each input byte of 8 bits can assume $2^8 = 256$ various bit patterns, and all patterns are equally likely to occur. When mapping a byte of 8-bit to one of the 45 restricted characters, one may let 40 bit patterns uniquely map to the 40 corresponding characters; consequently, the mappings of the other 216 patterns have to use two characters each. In other words, five characters are reserved and used as leading characters for mapping bit patterns to two-character pairs; these five leading characters can accommodate $5 \times 44 = 220$ bit patterns. On the average, in using this method, the translated file is expanded to 185% of the original file since

$$\frac{40}{256} \times 1 + \frac{216}{256} \times 2 = 1.84375 \rightarrow 184.37\%.$$

The Navy requires expansion ratios not larger than 50% [9]. The expansion ratio of 85% is unacceptably high; therefore, a source file must be translated in blocks of bits smaller than 8 when data storage efficiency is concerned. Similarly, one can verify that if the translation is performed in 7-bit blocks by assigning 43 bit patterns to single characters and $85(2^7 - 43)$ bit patterns to pairs of two characters, one would get an expansion ratio of $90\% = (8-7)/7 \times (43/128) + (16-7)/7 \times 85/128$.

Note that $5 < \log_2(45) < 6$. Thus, the bit pattern to be translated could be blocks of either 5 or 6 bits, depending on the efficiency of expansion ratio to be discussed below. Namely, a translated character (8 bits) can represent a block of either 5 or 6 bits of input bit stream. This implies that an output byte (character) always starts with a 0 bit and the other 7 bits vary in 45 patterns. Therefore, a shift operation on input string is needed to output a byte in the desired range. Three basic translation methods are discussed as follows.

1. *Method A:* Scan an input stream in 6-bit blocks. Since a 6-bit block may form 64 different patterns, with only 45 characters to map, there are 44 lucky 6-bit patterns that can map to a single character in the restricted character set (say, $\alpha_1, \alpha_2 \cdots \alpha_{44}$), whereas the rest of the 20 patterns have to be translated in two combined characters $\alpha_{45}\alpha_i, i = 1, \cdots, 20$.
2. *Method B:* Scan an input stream in 5-bit blocks. Since a 5-bit block may span 32 different patterns, with 45 characters to map, there are 13 characters in the restricted character set unused.
3. *Method C:* This is an improvement to the second method and will translate 6-bit block whenever possible. However, it scans the input stream in 6-bit blocks before committing to a translation. We may assign 32 restricted characters to integer values [0, 31] for 5-bit blocks and the other unused 13 characters to [32, 44] for 6-bit blocks. If the value of the 6-bit block is in the range of [32, 44], then the block is translated to the corresponding character. When the value of a 6-bit block is not in the range of [32, 44], then it is either in [0, 31] or in [45, 63]. The algorithm shifts one bit backward (unget), leaving the value in [0, 31], and translates the 5-bit block. In the following discussion, let S denote the integer interval [32, 44].

A. Expansion Ratios

It can be shown that Methods A and B are not as efficient as Method C. Without prior knowledge of the source stream, it is reasonable to assume that all bit patterns are equally likely to occur in the following discussions. Let b_1, b_2 be the number of bits used to encode a translated character (byte), and let P_{b1}, P_{b2} be the corresponding probabilities of occurrences in translation. The expansion ratio η of Method A is then

$$\eta = (8 - b_1)P_{b1}/b_1 + (8 \times 2 - b_2)/b_2 = 0.75 \Rightarrow 75\%$$

where

$$b_1 = 6, P_{b1} = 44/64 = 0.688, b_2 = 6, P_{b2} = 1 - P_{b1} = 0.312.$$

In the above computation, the second term indicates that we expand from 6 bits to 2 bytes for the 20 unlucky 6-bit patterns. Similarly, we can compute the expansion ratio for Method B as $\eta = (8 - 5)/5 = 0.6$, or 60%.

For Method C,

$$b1 = 6, b2 = 5.$$

Thus,

$$\eta(8-6)/6 \times (13/64) + (8-5)/5 \times (51/64) = 0.546 \Rightarrow 54.6\%.$$

Method C provides the best of all three translation methods and is very close to the set Naval requirement of 50 percent.

Variants of Method C can increase the probability of 6-bit block pattern translation, but they are not efficient. For

example, when $N = 45$, one may assign 16 characters for 4-bit and 29 characters for 6-bit. Two other variants are: 1) 8 characters for 3-bit and 37 characters for 6-bit, and 2) 4 characters for 2-bit and 41 characters for 6-bit. We calculate the corresponding expansion ratios for each case as follows.

1) 29 characters for 6-bit with 35 (=45–29) others for 4-bit:

$$\eta = (8-6)/6 \times (29/64) + (8-4)/4 \times (35/64) = 0.698 \Rightarrow 69.8\%$$

2) 37 characters for 6-bit with others for 3-bit:

$$\eta = (8-6)/6 \times (37/64) + (8-3)/3 \times (27/64) = 0.896 \Rightarrow 89.6\%$$

3) 41 characters for 6-bit with others for 2-bit:

$$\eta = (8-6)/6 \times (41/64) + (8-2)/2 \times (23/64) = 1.292 \Rightarrow 129.2\%$$

All three of these results are worse than Method C because the second term in each calculation grows faster than the reduction of the corresponding probabilities. Moreover, it is impossible to translate more than 6 bits in each decision when $N < 64$. To generalize the η computation of Method C for a restricted character set of size N in the range [2, 256], η can be calculated as follows:

$$\eta = (8 - b_1) \times (P_{b1}/b_1) + (8 - b_2) \times (P_{b2}/b_2)$$

where

$$b_1 = \lceil \log_2(N) \rceil, P_{b1} = \frac{N - 2^{\lfloor \log_2(N) \rfloor}}{2^{\lceil \log_2(N) \rceil}} = \frac{N - 2^{b_2}}{2^{b_1}}$$

and

$$b_2 = \lfloor \log_2(N) \rfloor, P_{b2} = 1 - P_{b1}.$$

The equation here is similar to the computations used in the previous paragraph except that it is now parameterized with N. Practical operation environment dictates the choice of N. For example, in order to accommodate a set of Morse code communication, the choice of $N = 45$ seems reasonable regardless of the expansion ratio. When $N < 16$, it is not practical to perform character translation since the expansion ratio can be as high as 700%. On the other hand, when N approaches 256, there is no need for character translation since the source character set and the target character set are equal in size. In software implementation, the bit-shift manipulation may improve the expansion ratio as will be explained in the next subsection.

B. Software Implementation

A C program based on Method C was implemented [1]. We now describe the algorithm that has been incorporated in a compressed/encrypted data package. The character set translation algorithm has two separate parts: translation (at host system) and recovery (at remote system).

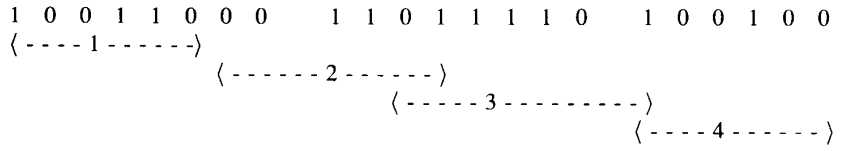
1) Translation Algorithm: The translation algorithm scans the input stream in 6-bit blocks before committing to a translation. We may assign 32 restricted characters (. through M) to decimal values interval [0, 31] for 5-bit blocks and the other 13 characters (N through Z) to the interval [32, 44] for 6-bit blocks. If the value of the 6-bit block is in the interval [32, 44], then the block is translated to the corresponding character. When the value is not in the range of [32, 44], then it is either in [0, 31] or in [45, 63]. The algorithm shifts one bit backward (unget), making the value reside in interval [0, 31], and translates the 5-bit block.

The translation algorithm scans the input stream from encoded and/or encrypted buffer into 6-bit blocks. It converts a 6-bit or 5-bit pattern into a unique character according to the integer value of the input block. Refer to the bit pattern dissection of a translation example below. When the input string is coming from right to left, we observe (see below).

- Bit pattern 1 (100110, the LSB is 0): Decimal value = 38 $\in S$ translates the 6-bit block and output 38 + 46 (T). Here, the 6-bit block of 100110 has a binary value of 38, which can also be seen in Table II.
- Bit pattern 2 (001101): Decimal value = 13 $\notin S$ translates a 5-bit block of 00110 (shift the pointer 1 bit backward, i.e., to the left), and output 6 + 46 (4). Note that 00110 has a binary value of 6.
- Bit pattern 3 (111101): Decimal value = 61 $\notin S$ translates a 5-bit block of 11110, and output 30 + 46 (L).
- Bit pattern 4 (100100): Decimal value = 36 $\in S$, output 36 + 46 (R).

The displacement of 46 above maps decimal values into the desired ASCII code range [., Z]. The output characters will be T4LR $\cdots$, and leave the last two bits to be the more significant bits of the next 6-bit block. When the EOF or last byte of the buffer is encountered, the remaining bits will be padded with 0's in LSB to form the last pattern. For the above example, if 10010001 is the last input byte, then the last 6-bit pattern will be 010000 and is translated to 01000 + 46 (6); the output is then T4LR6.

2) Recovery Algorithm: The displacement of 46 made in translation has to be reset for each input character in recovery at the receiving hosts. If the value after reset is in S, an original 6-bit translated pattern is assumed and a 6-



bit block is recovered; otherwise, it maps to a 5-bit block. The output characters T4LR6 in the previous example will be recovered to the original bit string. The process is explained now. First, the bit pattern of T4LR6 can be represented as hexadecimal string 54 34 4C 52 36. After a 46 offset is taken on each byte, the binary value corresponding to these five bytes becomes 38 6 30 36 8. In other words, the input to the inner loop of the recovery routine is now 00100110 00000110 00011110 00100100 00001000. This bit string will be recovered as 10011000 11011110 10010001. Because the file before translation is byte-oriented, the recovery of the last input character should complete the last byte of original compressed/encrypted package.

Translation and recovery algorithms are two separate functions in the implementation. In the host system, character set translation is the final step before transmission. The program takes each byte from the temporary file built by compression/encryption, and adds a 3-byte header in the output file. Each of the 3-byte headers, of course, is also within [46, 91] to guarantee the use of the restricted character set [., Z]. Moreover, the inner loop of translation function can be designed to incorporate the output buffer size of compression/encryption for various compression algorithms. For instance, the variable buffer size in an LZW algorithm requires a variable loop index [1]. At the receiving system, similar to the encryption operation, recovery operation is proceeded first by examining the file header.

VI. Improvement by Pattern Reassignment

In this section, each bit pattern corresponding to an output character will be examined. It will be shown that one can further reduce the translation expansion ratio by suitable reassignment of bit patterns.

A. Unused Patterns in Translation

The translation algorithm discussed in previous sections examines an input 6-bit block and translates either 5 or 6 bits of the input blocks. Theoretically, when each input pattern is assumed to be equally likely to occur, having been compressed and/or encrypted as discussed in Section V, Method C with expansion ratio 54.6% seems to be an optimized algorithm. Having enumerated all patterns, however, one can further reduce the expansion ratio to less than 50%—the Navy requirement [9].

The clue is that certain 5-bit patterns do not appear in practical translation procedure due to the 1-bit shift operation (unget) when the 6-bit value is not in S or interval [32, 44]. Table III lists all 6-bit patterns with corresponding decimal values and output characters. Notice that, in Table III, the 5-bit blocks in sets S_L and S_U exhibit redundancies and six patterns do not occur (values in interval [16, 21]). For instance, the first two 5-bit blocks in S_L are both 00000. Note that, of the 6-bit patterns in S_L and S_U, only 5-bit values are actually translated while each of the rightmost bits is restored back to input stream. Let x denote a don't-care bit. The six missing patterns

TABLE III
List of 6-Bit Patterns and Their Output Characters ;* When Unget 1 Bit ,** Total 39 Characters,*** '$N' = 32 + 46$

Set	6-bit Patterns	5-bit Value*	6-bit Value	Output**
	000000	0		'.'
	000001	0		'.'
	000010	1		'/'
S_L	.....	..		...
	.....	..		...
	011101	14		'<'
	011110	15		'='
	011111	15		'='
	100000		32	'N'***
	100001		33	'O'
S	.....	..	..	...
	.....	..	..	...
	101011		43	'Y'
	101100		44	'Z'
	101101	22		'D'
	101110	23		'E'
	101111	23		'E'
	110000	24		'F'
	110001	24		'F'
	110010	25		'G'
	110011	25		'G'
	110100	26		'H'
	110101	26		'H'
	110110	27		'I'
	110111	27		'I'
	111000	28		'J'
	111001	28		'J'
S_U	.....	..		...
	.....	..		...
	111101	30		'L'
	111110	31		'M'
	111111	31		'M'

are $10000x, 10001x, 10010x, 10011x, 10100x$, and $10101x$, with corresponding characters >, ?, @, A, B, C, respectively (see Table II). In other words, when Method C in Section V is used to translate 64 6-bit patterns, only 39 $(45 - 6)$ characters are actually assigned with 25 of them appearing twice and leaving six characters unused. The effort now is to translate patterns in S_L or S_U in 6-bit blocks by assigning unused characters to them. Reexamining Method C in Section V, one can verify that these missing characters in fact did not appear! That is, what we have done in the previous section is restricted to 39 characters instead of 45. This observation could improve or reduce the expansion ratio.

B. Characters Reassignment

Using the six unused characters can improve the expansion ratio. These six unused characters may be assigned to the first six unique 6-bit blocks of S_U (from 101101 to 110010; see Table III). That is, we can assign these six characters to values in [45, 50]. By doing this, the characters originally assigned to interval [45, 50] (four characters: D, E, F, and G) become unused. These four characters can be used to substitute another four 6-bit patterns, say, [51, 54] (from 110011 to 110110). Furthermore, the characters H and I correspond to [51, 54] and are reassigned to [55, 56]. This recursive characters reassignment may continue until value 57 is assigned and there are no more unused characters. 6 + 4 + 2 + 1 = 13 characters have been reassigned. As shown in Table V through appropriate displacement of +46 (within S_A), +30 (within S'), or +59 (within S_B), we can rearrange all output characters to be contiguous similar to that in ASCII code. The 6-bit patterns 100000 through 111001 (interval [32, 57]) are now assigned to characters > through W. All other 6-bit blocks (in S_A or S_B) still have to be translated in 5-bit blocks.

TABLE IV
LIST OF 6-BIT PATTERNS WITH NEW ASSIGNMENTS

Set	6-bit Patterns	5-bit Value	6-bit Value	Output
S_A	.....	..	0	...
	.....	..		...
	011111	15	31	'='
	100000		32	' >'
	100001		33	'?'
S'	.....		..	..
	.....		..	..
	101011		43	'I'
	101100		44	'J'
	101101		45	'K'
	101110		46	'L'
	101111		47	'M'
	110000		48	'N'
	110001		49	'O'
	110010		50	'P'
	110011		51	'Q'
	110100		52	'R'
	110101		53	'S'
	110110		54	'T'
	110111		55	'U'
	111000		56	'V'
	111010		57	'W'
S_B	111010	29		'X'
	111011	29		'X'
	111100	30		'Y'
	111101	30		'Y'
	111110	31		'Z'
	111111	31		'Z'

C. Expansion Ratio Improvement

The expansion ratio is improved because we increase the probability of translating 6-bit blocks and reduce that of 5-bit blocks. For all 64 possible 6-bit patterns, we now have 26 that can be translated and 38 that have to be translated in 5-bit blocks. The overall expansion ratio is then

$$\eta = (8-6)/6 \times (26/64) + (8-5)/5 \times (38/64)$$
$$= 0.4917 \Rightarrow 49.17\%.$$

This expansion ratio shows a 5.43% (=54.6 − 49.17%) improvement over that of Method C and achieves the expectation set by the Navy Security Group. Modification of the translation program from that of Method C to accommodate the observation in this section is made straightforward by adjusting the partitions for S_U, S, S_L (Table III) to S_A, S', S_B (Table IV). For all 64 possible 6-bit patterns, we now have 26 that can be translated and 38 that have to be translated in 5-bit blocks.

D. Experiments

The text reproduced below shows a short ASCII file that is to be processed through data compression, data encryption, and character set translation (phone numbers are not real).

TABLE V
TESTING RESULTS OF IMPROVED TRANSLATION

File	SIZE	IMPROVED SIZE	TRANSLATION η	Compress& SIZE	Translate %ORG
Text	24969	37243	49.16%	13853	55.48%
WordPerfect	25195	37570	49.12%	15082	59.86%
C Source	17325	25864	49.29%	6703	38.69%
Binary	24630	36745	49.19%	19695	79.90%
PAK	76644	114625	49.56%	44626	58.23%
PAK1	39024	58327	49.46%	23327	59.78%
PAK2	104622	156516	49.60%	61662	58.94%
PAK3	174226	260362	49.44%	95402	54.76%

```
I was very pleased to receive your draft
data compression research proposal dated
February 27th. As requested, I am enclosing
a detailed list of our projected require-
ments to further assist you in smoothing
your proposal and formalizing thesis work
on this subject. Please feel free to contact
LT Frank at (Comm) (202) 452-6313 or (AV)
911-1313 if additional clarification will
be helpful. Thank you for your professional
interest in our data compression needs.
Sincerely,
```

The compressed and encrypted file after character set translation is shown below; all characters appeared within . and Z as desired. Note that there is neither CR (carriage return) nor LF (line feed). The file is displayed in multiple lines for readability. Because the original file is small and therefore resistant to compression, translated file size is larger than that of the original file (see below).

An extensive performance analysis on data compression software found that an average of 39.3% compression ratio is achievable [8]. The theoretical compressed and translated file size can be reduced to 39.3% × (1 + 49.17%) = 58.62%. This improved algorithm has been applied to a set of testing files. Let PAK represent the file types used by the Naval data package which consists of 90% ASCII text and 10% binary or image data. The results are listed in Table V and are consistent with the larger variance among testing results. However, this is shown in the last column–when both compression and

```
B2C8/3 > :0.F4./C6:B./55XW;CP.2/A > 2N63/24160.27043.H;A < R6D;;
JY.8/?22R7@Z;3M < > N74.IP;TO < 7.XJK3:T3P6L9:PPALJT6 > X4Z.88;MT
@P;4F4Z290MX=3.8BHQ < 90Q < B8 < X:9G?6455U;@T < YRY5:UXHU;NR8B.=
ZP91J9.UZ7;YO;U;LW:5:OB=ZWLK3;94W < D160XWBTX4? < N641I1XEASDO
MXM5Y3LV < TR < Z:LXAD82XZRXR= < E=56 < VPT7SD=YQ7F1MB1=A80P2M2/ < W
H > ;;:Y7EDD2LNUY8; < N=SY8UD==6K;98LZ=0Z6N6ZT.5:4J6554UXZ;1=H
YG:Y=Y/ < F < HN;Y0TLO;U3KC1LK1U < Y==ZKUZXQ8IGM749LXIZPOXWQX < 34
B9K2U0?.1=Y0W;BQZDX/ST92?OO8KY/T1M8/48CNIZ17.L < XT3=J6R7282
< ZKF8VPR7RR:VBTQGJP91:QVRWVSE > ; < 8F0JGY;1.Q:T9155D:4H6 < 7 < 94
O;1GJUIZS3ZZNHHBM5RU > X80B84/ > U62X=P < < < J/MH?::1.7W@I05M3TKE
C:BC@F2J. > 63 > .3N.:H12 > .7QYY9:L6. > : < .6/ > 0/20?24F2J0 > N62.100
7 > 2ZV
```

translation are performed. This is due to the different file types benefiting from different compression ratios. The PAK files in the last column are reduced to 58.23%, 59.78%, 58.94%, or 54.76% of their original file sizes; these are very close to the theoretical value 58.62! This meets the requirement set by the Naval Security Group Detachment (NSGD) [9]: a compression ratio of 50% and an expansion ratio of 50% with the combining effect of reducing file size to 75% of original size when the file is compressed and translated.

The discussion in this section assumed that the occurrences of all bit patterns are equally likely in the input to the translation algorithm. This is a reasonable assumption since the input bit patterns are dictated by the pointer values in unknown compression/encryption algorithms. When bit patterns are not equally likely, the translation algorithm may be more sophisticated but may achieve a better expansion ratio. For example, if a text file to be processed does not require compression or encryption, the expansion ratio of the character set translation should be smaller than the theoretical 49.17% because the first and/or second 0 bit of each input byte may be skipped in the translation process.

VII. Conclusion

It has been shown that a source file can be translated using a restricted character set. Naval message traffic is transmitted with a restricted character set, and the files are optionally compressed. Both character translation and data compression can be used as add-on data encryption. Various schemes of restricted character set translation have been investigated and their implementations on computers are discussed. The translation algorithms that use a restricted character set have been implemented with multilevel security access control using the master key scheme and have been incorporated in a data compression program for military applications.

Although the key management scheme discussed in this paper is perfectly feasible, it is by no means the single or best possibility. The method employed here allows only for the availability of different keys for different links and hosts, but does not differentiate the different functions or activities for which the keys are used. The functions stressed in this study are data compression and character translation; however, the host system is most likely more versatile. Moreover, the transmissions between hosts, remotes and hosts, remotes and remotes, if independent of each other in encryption, may provide much better protection. These important issues could be solved by a key management scheme based on the popular private-key algorithm, DES. This, however, is beyond the scope of this paper. Nevertheless, its possibilities introduce worthwhile followup research.

Finally, to make all algorithms and source code completely transportable among hardware/operating system environments, the work of error detection and correction becomes an absolute necessity. It is possible to use checksum or CRC techniques to detect transmission errors by attaching d characters to each block of b characters. These d checking characters (D) are computed from the b information characters (B). Traditional checksum is not capable of locating which byte in B is in error since characters in B may have $b!$ permutations, and some permutations may result in the same D. To facilitate the error correction, we may have to use some noncommutative operations in the construction of D from B. For instance, characters in B are arranged in two matrices and their product is used as D. Because matrix multiplication is noncommutative, it may be a starting point for character-oriented error detection and correction. Other powerful error correction codes such as R–S (Reed–Solomon codes) are also available for further study.

Acknowledgment

This research has benefited tremendously from the Naval Security Group Detachment, Pensacola, FL. The authors are grateful to Capt. Engel and his staff for the initiation of the research subject. The authors are indebted to Lt. P. Nguyen and anonymous referees for helpful comments on the working draft. The research was started while the authors were at the U.S. Naval Postgraduate School, and the final revisions were done while the authors were in Taiwan, Republic of China.

References

[1] C. C. Tsai, "Multilevel security with master key and restricted character set translation in data compression," Naval Postgrad. School, Masters thesis, Mar. 1992.
[2] FIPS, "Data encryption standard," Fed. Inform. Process. Stand. Pub. 46-1, NBS, Jan. 1988.
[3] G. C. Chick and S. E. Tavares, "Flexible access control with master keys," in *Crypto '89*. New York: Springer-Verlag, pp. 316–322.
[4] S. G. Akl and P. D. Taylor, "Cryptographic solution to a multilevel security problem," in *Crypto 82*. New York: Springer-Verlag, pp. 237–249.
[5] IEEE, "IEEE Standard for binary floating-point arithmetic," ANSI/IEEE Standard 754–1985.
[6] P. A. Findlay and B. A. Johnson, "Modular exponentiation using recursive sums of residues," in *Advances in Cryptology—Crypto '89*. New York: Springer-Verlag, pp. 371–386.
[7] Commander, Naval Computer Telecommun. Command NTP3, *Annex C*, 4401 Mass. Ave. NW, Washington, DC 20394-5000. (Unclassified but need-to-know basis.)
[8] Y. J. Jung, C. Yang, and G. A. Myers, "Performance analysis of data compression and archiving software for U.S. Navy," presented at the *9th Ann. Decision Aids Conf.*, June 1992.
[9] Commander, Naval Security Group Detachment, Corry Station, *Data Compression Project Specification*, Mar. 1991.

Chyan Yang (S'86–M'87–SM'90) received the M.S. degree in information and computer science from the Georgia Institute of Technology and the Ph.D. degree in computer science from the University of Washington.

He is currently an Associate Professor in the Institute of Management Science and the Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan, Republic of China. His current teaching and research interests are distributed operating systems, office automation, and network security. From 1987 to 1992, he was an Assistant Professor of Electrical and Computer Engineering at the U.S. Naval Postgraduate School where he carried out research in computer networks, parallel processing, information management, and microelectronic systems.

Dr. Yang is a member of ACM.

Chien-Chao Tsai received the B.S. degree from the Chinese Military Academy in 1978, and the M.S.E.E. degree from the U.S. Naval Postgraduate School in June 1992.

Since 1978, he has worked at MICA (Military Integrated Communications Agency), Taiwan, Republic of China. His research interests include computer systems, communications, and software engineering. He is now a Squadron Leader in MICA and is planning to continue his research toward a doctoral degree.