A New Access Control Method Using Prime Factorisation

J.-J. HWANG[1], B.-M. SHAO[2] AND P. C. WANG[3]*

[1] *Department of Management Science and Institute of Information Management, National Chiao-Tung University, Hsinchu, Taiwan 30050, R.O.C.*

[2] *Institute of Information Management, National Chiao-Tung University.*

[3] *Institute of Statistics, National Central University, Chungli, Taiwan 32054, R.O.C.*

Based on prime factorisation in number theory, a new but simple and efficient protection method for access control is proposed. This method has several advantages on performance over previous methods in the literature. Especially, a file/user can be added to or removed from the system without much effort. Excellence of this method is more pronounced for those systems where files are accessible to only a few users.

Received May 1991; revised August 1991

1. INTRODUCTION

Access control protects privacy, integrity and availability of information in computer systems. It determines the accesses to information resources stored in a system by verifying the access rights of accessors. Accessors could be users, processors or processes. Resources could be files, proprietary programs, memory segments or devices. Access rights could be read, write, execute, own, or various privileges of changing the access rights. For discussion, the terms *user* and *file* are chosen here to represent accessor and resource respectively.

An access matrix as given in Fig. 1 is used to specify access rights of users to files. Particular implementations of the matrix can be impractical or inefficient depending on various factors.[1,2] Instead, Wu and Hwang[2] proposed an alternative scheme storing just one key for each user and one lock for each file. To figure out access rights (a_{ij})s of users to files, a function f of key K_i and lock L_j is used. Mathematically, $f(K_i, L_j) = a_{ij}$.

Several relevant methods appeared in the literature after Wu and Hwang's work. Chang proposed two of them based, respectively, on the Chinese remainder theorem and Euler's theorem in number theory.[3,4] Laih *et al.* used Newton's interpolating polynomial to design another method in 1989,[5] while Chang and Jiang presented a binary version of Wu and Hwang's method.[6] These were classified as single-key-lock (SKL) schemes.

User U_i \ File F_j	F_1	F_2	F_3	F_4	F_5	F_6
U_1	4	0	3	0	4	3
U_2	0	2	4	2	0	4
U_3	1	4	0	0	1	2
U_4	1	0	1	4	0	0

0: No access
1: Execute
2: Read
3: Write
4: Own

Figure 1. An access matrix $[a_{ij}]_{4\times6}$.

* To whom correspondence should be addressed.

In order to evaluate the effectiveness and efficiency of an SKL scheme, the following six criteria are considered.

(1) Effort for initialising keys and locks.

(2) Effort for computing an access right from a lock and key.

(3) Effort for revising keys and locks when an access right is modified.

(4) Effort for appending and updating keys and locks when a new user or file is added.

(5) Effort for removing and updating keys and locks when a user or file is deleted.

(6) Space for storing keys and locks.

A method will satisfy user/file appendability if the action for adding a new user/file is done by appending a new key/lock without affecting existing keys and locks. It is similarly said to satisfy user/file removability if the action for deleting an existing user/file is done by removing a key/lock without affecting other keys and locks.

In this paper we develop a new but simple and efficient SKL scheme using prime factorisation. Based on the criteria above, our method has good overall performance. In the next section, the idea is presented first and then three algorithms are developed for practical application. In section 3 our method is compared with previous ones. The conclusion follows.

2. THE NEW METHOD

2.1 Basic mechanism

The unique factorisation theorem states that the factoring of any integer $n > 1$ into primes as $n = p_1^{N_1} p_2^{N_2} \ldots p_r^{N_r}$ is unique, where p_is are distinct primes ($p_1 < p_2 < \ldots < p_r$) and N_i is the number of occurrences of p_i in factoring n. This representation of n as a product of prime powers is called the canonical factorisation of n. The uniqueness of this factorisation was proved by Gauss.[7] For example, the unique way to factor 180 is ($180 = 2^2 \cdot 3^2 \cdot 5^1$).

Like other SKL schemes, our method assigns a key to each user and a lock to each file. Distinct primes in a given list are used as keys, say $\{K_i\}$. Let $[a_{ij}]_{m\times n}$ be an access matrix. Locks (L_j) are then computed by $L_j = \prod_{i=1}^{m} K_i^{a_{ij}}$.

Let the maximum of access rights be a_{max}. Algorithm A below is used as $f(K_i, L_j)$ to figure out access rights (a_{ij}) from keys and locks.

Algorithm A
Input. Key K_i of user U_i and lock L_j of file F_j.
Output. Access right a_{ij} of U_i to F_j.
Step 1. Input K_i and L_j.
Step 2. Set $a_{ij} = 0$, $Temp = L_j$.
Step 3. Repeat
Compute $Q = Temp/K_i$.
If Q is an integer Then Set $a_{ij} = a_{ij} + 1$, $Temp = Q$.
Until Q is not an integer Or $a_{ij} = a_{\max}$.
Step 4. Output a_{ij}.

Performance of the proposed method will now be discussed based on six criteria set in Section 1. First, primes used as keys are available in many books on number theory (e.g. Ref. 7) when the number of users is moderate. As many as we need can be generated from a computer. Lock $L_j = \prod_{i=1}^{m} K_i^{a_{ij}}$ is not difficult to compute. Since $a_{ij} = 0$ represents 'no access', computations of L_j are relatively simple for the case in which most files are only accessible to a few users, resulting in a sparse access matrix whose entries are mostly zeros. More important, such cases are common in the general time-sharing and multi-user systems.

Second, Algorithm A uses several divisions to find a_{ij}. The number of divisions is clearly bounded by $a_{\max}$, which is usually a small positive integer. Computation of a_{ij} is efficient.

Third, when an access right is changed from a_{ij} into a'_{ij}, lock L_j is changed into $L'_j = L_j K_i^{(a'_{ij} - f(k_i, L_j))}$. All keys and other locks are retained. Such a revising effort is small.

Fourth, the case of adding a new file F_{n+1} is considered. Our method needs to compute $L_{n+1} = \prod_{i=1}^{m} K_i^{a_{i,n+1}}$ only. This is done without modifying existing keys and locks. File appendability is hence satisfied. To delete an existing file, it is enough to discard the corresponding lock. File removability is also satisfied.

Fifth, to add a new user U_{m+1} with access rights $a_{m+1,j}$ for $j = 1, 2, \ldots, n$, an unused prime K_{m+1} is assigned as his key. The locks of existing files accessible to U_{m+1} are then recomputed by $L'_j = L_j (K_{m+1})^{a_{m+1,j}}$. It is emphasised that only the locks of existing files which U_{m+1} can access need to be altered, since $L_j (K_{m+1})^{a_{m+1,j}} = L_j$ if $a_{m+1,j} = 0$. It is common that most users cannot access files of others. Thus in most cases appending a new user only requires changing a few locks. To delete user U_r, $L'_j = L_j K_r^{-f(K_r, L_j)}$ is recomputed for any F_j of the files accessible to U_r (i.e. $f(K_r, L_j) > 0$). K_r is then reserved for future users.

Finally, storage of keys should not be difficult. Locks here, however, are apt to overflow beyond the largest integer allowed in a system. The next subsection offers a solution to this problem.

Example 1
Application of the proposed method will now be illustrated using the access matrix $[a_{ij}]_{4\times 6}$ in Fig. 1.

(1) *Initialisation*. Let $K_1 = 2$, $K_2 = 3$, $K_3 = 5$ and $K_4 = 7$. Then $L_1 = 2^4 \cdot 5^1 \cdot 7^1 = 560$, $L_2 = 3^2 \cdot 5^4 = 5625$, $L_3 = 2^3 \cdot 3^4 \cdot 7^1 = 4536$, $L_4 = 3^2 \cdot 7^4 = 21609$, $L^5 = 2^4 \cdot 5^1 = 80$, and $L_6 = 2^3 \cdot 3^4 \cdot 5^2 = 16200$.

(2) *Verifying access request*. Suppose the system receives access request $(U_1, F_3, 3)$. The scheme fetches $K_1 = 2$ and $L_3 = 4536$ to compute $f(2, 4536) = 3$. Request is then accepted. Access request $(U_3, F_5, 2)$ would be rejected, since $f(K_3, L_5) = f(5, 80) = 1 \neq 2$.

(3) *Changing access right*. Assume a_{22} is changed into 3. Lock L_2 is recomputed as $L'_2 = L_2 K_2^{(3 - f(K_2, L_2))} = 5625 \cdot 3^{(3-2)} = 16875$.

(4) *Addition and deletion of files*. If U_2 creates a new file F_7 that U_1 can read, U_3 can execute and U_4 cannot access, then $L_7 = 2^2 \cdot 3^4 \cdot 5^1 = 1620$. Deleting F_7 is accomplished by simply dropping L_7.

(5) *Addition and deletion of users*. Suppose a new user U_5 with $a_{51} = 1$, $a_{52} = 0$, $a_{53} = 1$, $a_{54} = 0$, $a_{55} = 2$, and $a_{56} = 0$ is added to the system. A key $K_5 = 11$ is assigned to him. Locks L_1, L_3, and L_5 of the files accessible to U_5 need to be revised: $L'_1 = 560 \cdot 11^1 = 6160$, $L'_3 = 4536 \cdot 11^1 = 49896$, and $L'_5 = 80 \cdot 11^2 = 9680$. Deleting U_5 is done by recomputing $L''_j = L'_j \cdot K_5^{-f(K_5, L'_j)} = L_j$ for $j = 1, 3, 5$.

2.2 Decomposition of locks

In the new method, user keys (K_i) are prime numbers and file locks (L_j) are computed by $L_j = \prod_{i=1}^{m} K_i^{a_{ij}}$. Since a lock is the product of some prime powers, it may easily exceed the largest integer allowed in a computer with b-bit integer wordlength. To solve this overflow problem, lock decomposition is necessary. A base X is chosen to divide lock L_j recursively as follows until the last quotient $Q_{j,l}$ is zero.

$$\begin{aligned} L_j &= Q_{j,1} X + R_{j,1}, \\ Q_{j,1} &= Q_{j,2} X + R_{j,2}, \\ &\vdots \\ Q_{j,l-1} &= Q_{j,l} X + R_{j,l}, \end{aligned}$$

where remainders ($R_{j,k}$) satisfy $0 \leqslant R_{j,k} < X$ for $k = 1, \ldots, l$. To avoid overflow computations in the algorithms given later, X has to be chosen no larger than the square root of 2^b. Also, keys have to be set less than X (i.e., $X \leqslant 2^{\lfloor \frac{b}{2} \rfloor}$ and $K_i < X$ for $i = 1, 2, \ldots, m$).

In notation,

$$L_j \equiv (R_{j,l}, R_{j,l-1}, \ldots, R_{j,1}),$$

which is called L_j in X-based form. For example, if $L_j = 165$ and $X = 8$, $165 = 2 \cdot 8^2 + 4 \cdot 8^1 + 5 \equiv (2, 4, 5)$ and L_j is represented by $(2, 4, 5)$. Using this representation of locks, locks can be stored without problem. Algorithm B, however, needs to be repeated several times to generate each of the locks in X-based form.

Algorithm B
Input. Multiplicand *Mpd* in X-based form $(R_{j,l}, R_{j,l-1}, \ldots, R_{j,1})$ and multiplier K_i.
Output. Product P in X-based form $(P_p, \ldots, P_1)$.
Step 1. Input *Mpd* in X-based form $(R_{j,l}, R_{j,l-1}, \ldots, R_{j,1})$ and K_i.
Step 2. Set $carry = 0$.
Step 3. Begin For $k = 1$ To l Do
Compute
$N = R_{j,k} \cdot K_i + carry$,
$carry = \lfloor \frac{N}{X} \rfloor$,
$P_k = N - carry \cdot X$.
End.
Step 4. If $carry > 0$ Then Set $p = l + 1$, $P_p = carry$ Else Set $p = l$.
Step 5. Output $(P_p, \ldots, P_1)$.

On the other hand, a special type of X-based division is also needed to compute a_{ij}. Algorithm C below is devised to perform the X-based division '$Q = Temp/K_i$' in Algorithm A when lock decomposition is incorporated.

Algorithm C
Input. Dividend *Dvd* in X-based form $(R_{j,l}, R_{j,l-1}, \ldots, R_{j,1})$ and divisor K_t.
Output. Quotient Q in X-based form $(Q_q, \ldots Q_1)$ with remainder r.
Step 1. Input *Dvd* in X-based form $(R_{j,l}, R_{j,l-1}, \ldots R_{j,1})$ and K_t.
Step 2. Set $r = 0$.
Step 3. Begin For $k = l$ Down To 1 Do
Compute
$N = R_{j,k} + r \cdot X$,
$Q_k = \lfloor \frac{N}{K_t} \rfloor$,
$r = N - Q_k \cdot K_t$.
end.
Step 4. If $Q_l > 0$ Then Set $q = l$ Else Set $q = l-1$.
Step 5. Output $(Q_q, \ldots Q_1)$ and r.

Note that computations of '$N = R_{j,k} \cdot K_t + carry$' in Algorithm B and '$N = R_{j,k} + r \cdot X$' in Algorithm C do not overflow due to our appropriate choice of $X \leqslant 2^{\lfloor \frac{b}{2} \rfloor}$ in a system with b-bit integer wordlength.

2.3 Exploration of storage requirement

We first count how many users the method can accommodate. If the integer wordlength has $b = 32$ bits, $X = 2^{\lfloor \frac{32}{2} \rfloor} = 2^{16} = 65536$. Primes less than 65536 are candidate keys for users and the number of such primes is $\pi(65536) = 6542$. The method can hence accommodate 6542 users. It would be enough for practical use, and key decomposition would be unnecessary. With the increase of b, user accommodation is exponentially enlarged. If $b = 64$ bits, $X = 2^{\lfloor \frac{64}{2} \rfloor} = 2^{32}$ and from the prime number theorem,[7] user accommodation would be increased to $\pi(2^{32}) \approx \frac{2^{32}}{\ln 2^{32}} \approx 193635000$. Therefore, the storage of keys is not a problem in the new scheme.

Next assume l is the greatest length of all locks in X-based form. $L_j = \Pi_{i=1}^{m} K_i^{a_{ij}} \leqslant \Pi_{i=1}^{m} K_i^{a_{\max}} < \Pi_{i=1}^{m} X^{a_{\max}} = X^{a_{\max} \cdot m}$ and then $l < a_{\max} \cdot m$. Storage for files with lock decomposition is hence $O(mn)$.

In order to further explore how lock decomposition works, a simulation study on a 32-bit Personal System/2 has been conducted (coded in C). Various numbers of users and files were carried out for explorations. Two deterministic parameters are picked in the study: *Non_Zero_Rate*, which is defined as the ratio of nonzero entries in the access matrix, ranges from 0.1 to 0.9 with increment 0.1, and $a_{\max}$ is set to $2, 3, 4, \ldots, 9$. Then two uniform distributions are utilised to determine who has the right to access a file and what access right one has when offered respectively. Available primes were picked for keys, and locks in X-based form were found using our procedure. The spaces needed for these locks were counted and divided by mn to give the *Storage_Index*.

For different m and n, we obtain similar results. Only the results in the case of $m = 5000$ users and $n = 50$ files are summarised in Fig. 2 with *Non_Zero_Rate* in the X-axis and *Storage_Index* in the Y-axis. Each solid line represents a relation of *Storage_Index* with *Non_Zero_Rate* for different $a_{\max}$. The dashed line represents the constant *Storage_Index* one. Surely small *Storage_Index* is preferred. As in Fig. 2, most cases have *Storage_Index* smaller than one. This means most of time the storage for locks in our procedure is less than

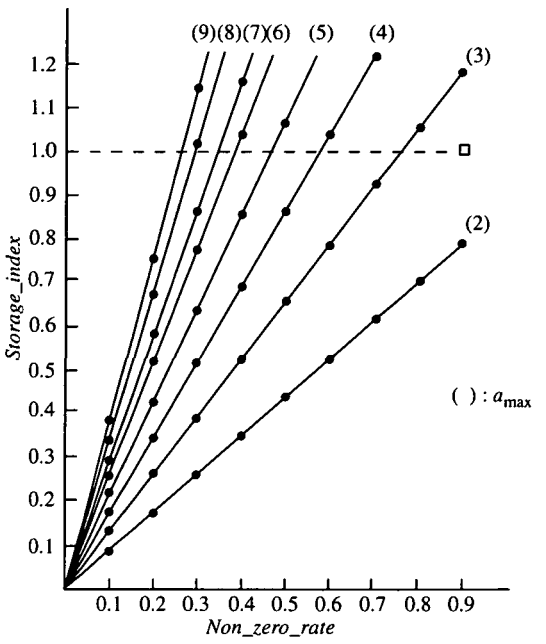

Figure 2. Simulation result of storages in the case of $m = 5000$ and $n = 50$.

mn. Some cases such as $a_{\max} = 9$ and *Non_Zero_Rate* = 0.1 save even more (60% of spaces). Some cases without sparse access matrix, say, *Non_Zero_Rate* > 0.3, would have *Storage_Index* greater than one. However, this does not need to be emphasised, since the access matrix is usually sparse in a time-sharing and multi-user computer system.

3. COMPARISONS

In this section, our method is compared with SKL schemes in the literature based on six criteria set in Section 1. Previous methods are briefly reviewed.

Wu and Hwang[2] used vectors K_i and L_j as keys and locks to find a_{ij} through $f(K_i, L_j) = K_i * L_j = a_{ij}$, where the operator * denotes the inner product in Galois field $GF(t)$ and t is the smallest prime larger than $a_{\max}$. Key vectors (K_i) must be linearly independent, and lock vectors (L_j) are constructed by solving sets of linear equations.

Chang's method,[3] referred to as Chang's 86, is based on the Chinese remainder theorem. Access right is computed by $f(K_i, L_j) = K_i \bmod L_j = a_{ij}$. File locks (L_j) are coprime numbers and user keys (K_i) are computed by

$$K_i = \sum_{j=1}^{n} (L/L_j)\, x_j\, a_{ij} \bmod L, \quad (i = 1, \ldots, m),$$

where $L = \Pi_{k=1}^{n} L_k$, and x_j satisfies $(L/L_j)\, x_j \bmod L_j = 1$. Note that the extended Euclid's algorithm is needed to find x_j.[8]

Chang's method,[4] referred to as Chang's 87, utilises Euler's theorem. Access right is obtained by $f(K_i, L_j) = \lfloor K_i/L_j \rfloor \bmod n = a_{ij}$. Locks (L_j) are coprime numbers and keys are computed by

$$K_i = \sum_{j=1}^{n} [a_{ij} \cdot L_j/n]\, nM_j, \quad (i = 1, \ldots, m),$$

where $M_j = (L/L_j)^{\phi(L_j)}$, $L = \Pi_{k=1}^{n} L_k$ and ϕ is Euler's totient function. *Note*: M_j need extra computations and are very large numbers for a general n.

Aiming at user/file appendability, Laih *et al.*[20] designed an SKL system using Newton's interpolating polynomial. Keys (K_i) are arbitrarily selected. Prime P is then chosen to satisfy $P > a_{\max}$ and $K_i \not\equiv K_j \bmod P$ for $K_i \neq K_j$. Lock vectors (L_j) are computed based on the coefficients of Newton's interpolating polynomial. There are m coefficients G_i^j in the polynomial corresponding to each lock, $L_j(x) = G_m^j(x-K_{m-1})(x-K_{m-2})\cdots(x-K_1)+\cdots+G_2^j(x-K_1)+G_1^j (j = 1,\ldots,n)$, where

$$G_i^j = \frac{\left[a_{ij} - \sum_{t=1}^{i-1} G_t^j \prod_{s=1}^{t-1}(K_i - K_s)\right]}{\prod_{s=1}^{i-1}(K_i - K_s)} \bmod P, \quad (i = 2,\ldots,m)$$

is computed recursively with starting value $G_1^j = a_{1j}$. Access right is obtained by computing

$$f(K_i, L_j) = L_j(k_i) \bmod P = a_{ij}.$$

Note: G_i^j involve complex computations.

Chang and Jiang[6] offered a binary version of Wu and Hwang's SKL scheme. Keys are linearly independent vectors with 0 and 1 components. Lock L_j is expanded to a binary $d \times m$ matrix, where $2^d \geqslant a_{\max}$. The sth bit of $a_{ij} = K_i * L_{sj}$, where $*$ is the inner product in $GF(2)$ and L_{sj} means the sth row of lock L_j ($1 \leqslant s \leqslant d, 1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$).

Each method introduced above has its own ways of dealing with the six criteria for evaluating an SKL scheme. They are summarised in the following six tables.

Table 1 deals with initialisation of m keys and n locks. *Note*: the underlined items x_j, M_j, and G_i^j are barriers to computations of keys or locks in corresponding methods. Solving sets of linear equations is also time-consuming. This may suggest that our method demands the least effort for initialisation among all six methods. Chang's two methods have to resolve the same overflow issue as ours and may apply the same decomposition technique.

Numbers of operations to find access rights are given in Table 2. Chang's 86, 87, and our method need only a constant number of operations to compute access rights, while others require a number of operations proportional to m (i.e. the number of users).

Table 1. Initialisation of keys and locks

SKL schemes	Effort for initialising m keys and n locks
Wu and Hwang's	Give m keys, solve n sets of m linear equations for n lock vectors
Chang's 86	Give n locks, compute $K_i = \sum_{j=1}^{n} (L/L_j)\underline{x_j} a_{ij} \bmod L$ for m keys
Chang's 87	Give n locks, compute $K_i = \sum_{j=1}^{n} [a_{ij} L_j/n] n\underline{M_j}$ for m keys
Laih *et al.*'s	Give m keys, obtain $L_j(x) = \sum_{t=1}^{m} \underline{G_t^j} \prod_{s=1}^{t-1}(x-K_s)$ for n lock vectors
Chang and Jiang's	Give m keys, solve n sets of dm 0-1 linear equations for n lock matrices
Our method	Give m keys, compute $L_j = \prod_{i=1}^{m} K_i^{a_{ij}}$ for n locks in X-based form

Table 2. Computation of access rights

SKL schemes	Operations needed to compute access right a_{ij}
Wu and Hwang's	m multiplications, $(m-1)$ additions and one division
Chang's 86	One division
Chang's 87	Two divisions and one subtraction
Laih *et al.*'s	$(i-1)$ multiplications, $(i-1)$ additions and one division
Chang and Jiang's	dm ANDs and $d(m-1)$ XORs (very fast in $GF(2)$)
Our method	$\leqslant a_{\max}$ (X-based) divisions

Table 3. Modification for changing access rights

SKL schemes	Effort for changing access right a_{ij} into a'_{ij}
Wu and Hwang's	Solve a new set of m linear equations for L'_j
Chang's 86	Recompute $K'_i = K_i + (L/L_j)\, x_j(a'_{ij} - f(K_i, L_j)) \bmod L$
Chang's 87	Recompute $K'_i = K_i + ([a'_{ij} L_j/n] - [f(K_i, L_j) L_j/n])\, nM_j$
Laih *et al.*'s	Recompute the $(m-i+1)$ coefficients $G_t^j (t = i,\ldots,m)$ for L'_j
Chang and Jiang's	Solve a new set of dm 0-1 linear equations for L'_j
Our method	Recompute $L'_j = L_j K_i^{[a'_{ij} - f(K_{ij}, L_j)]}$

Table 4. Appendability

SKL schemes	User appendability	File appendability
Wu and Hwang's	Recompute all lock vectors	Yes
Chang's 86	Yes	Recompute all keys
Chang's 87	Yes	Recompute all keys
Laih *et al.*'s	Yes (add a coefficient to each lock vector)	Yes
Chang and Jiang's	Recompute all lock matrices	Yes
Our method	Recompute the locks of accessible files only	Yes

Table 3 shows that only Chang's 86, 87, and our method can modify the original key or lock value to gain a new one, so the recomputation efforts are smaller than those of other methods. Modifications in Chang's two methods are, however, more complex due to computations of x_j, M_j and L.

Properties of appendability and removability listed in Tables 4 and 5 might be critical for practical application. For appendability, Laih *et al.*'s is best because it satisfies both user and file appendability. Our method satisfies

Table 5. Removability

SKL schemes	User removability	File removability
Wu and Hwang's	Recompute all lock vectors	Yes
Chang's 86	Yes	Recompute all keys
Chang's 87	Yes	Recompute all keys
Laih *et al.*'s	To delete U_t, recompute $(m-i)$ coefficients $G_t^j (i+1 \leqslant t \leqslant m)$ of all L_js for deleting G_t^j	Yes
Chang and Jiang's	Recompute all lock matrices	Yes
Our method	Recompute the locks of accessible files only	Yes

Table 6. Storage of keys and locks

SKL schemes	Required storage of m keys and n locks
Wu and Hwang's	$O(m^2+mn)$
Chang's 86	$O(m+n)$*
Chang's 87	$O(m+n)$*
Laih *et al.*'s	$O(mn)$
Chang and Jiang's	(m^2+dmn) bits
Our method	$O(mn)$

* Chang ignored the overflow issue and obtained $O(m+n)$.

file appendability, and when a new user is added it recomputes only the locks of accessible files instead of all locks. Our method would be the second and not far away from best. For removability, our method satisfies file removability, and when a user is deleted it recomputes only the locks of accessible files instead of all locks. Our method is hence best among all six methods for removability. In summary, the recomputation effort of our method is relatively small when a user is added to or removed from the system, especially for the case of a sparse access matrix.

Storages of keys and locks are as shown in Table 6. Note that the $O(m+n)$ for Chang's methods were obtained by ignoring the overflow issue. According to the formula for computations of keys, the key decomposition would require no less than $O(mn)$ storage for both Chang's methods. The storage requirement analysis in Section 2.3 indicates that our method would have more space saving for the cases with sparse access matrices.

4. CONCLUSION

Based on six criteria, our SKL scheme is a considerably better method for access control than most of the other comparable schemes. The merit lies in its simplicity in terms of both the underlying idea and the algorithm for computing access rights. The convenient way to modify keys and locks while adding or removing files/users is also impressive, especially for the case of a sparse access matrix. Before any better scheme appears, our method is a good alternative to the ones in the literature.

REFERENCES

1. D. E. R. Denning, *Cryptography and Data Security*. Addison-Wesley, Reading, MA (1982).
2. M. L. Wu and T. Y. Hwang, Access control with single-key-lock. *IEEE Transactions on Software Engineering* **10** (2), 185–191 (1984).
3. C. C. Chang, On the design of a key-lock-pair mechanism in information protection systems. *BIT* **26** (4), 410–417 (1986).
4. C. C. Chang, An information protection scheme based upon number theory. *The Computer Journal* **30** (3), 249–253 (1987).
5. C. S. Laih, L. Harn and J. Y. Lee, On the design of a single-key-lock mechanism based on Newton's interpolating polynomial. *IEEE Transactions on Software Engineering* **15** (9), 1135–1137 (1989).
6. C. K. Chang and T. M. Jiang, A binary single-key-lock system for access control. *IEEE Transactions on Computers* **38** (10), 1462–1466 (1989).
7. I. Niven and H. S. Zuckerman, *An Introduction to the Theory of Numbers*. Wiley, New York (1980).
8. D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Addison-Wesley, Reading, MA (1980).