Technical Report

UCAM-CL-TR-164
ISSN 1476-2986

Number 164

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

A matrix key distribution system

Li Gong, David J. Wheeler

October 1988

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN 1476-2986

A Matrix Key Distribution Scheme

Li Gong[1] and David J. Wheeler F.R.S.

University of Cambridge Computer Laboratory
Cambridge CB2 3QG, England

October 1988

Abstract. A new key distribution scheme is presented. It is based on the distinctive idea that lets each node have a set of keys of which it shares a distinct subset with every other node. This has the advantage that the numbers of keys that must be distributed and maintained are reduced by a square root factor; moreover, two nodes can start conversation with virtually no delay. Two versions of the scheme are given. Their performance and security analysis shows it is a practical solution to some key distribution problems.

Key words. Communication Security, Private-key Cipher, Conversation Key, Key Distribution.

[1]Supported by the Sino-British Friendship Scholarship Scheme.

1 Introduction

The effectiveness of any cryptographic system is highly dependent on the techniques used for selecting, handling, and protecting the keys. Key distribution is always a major problem in an environment where large numbers of nodes communicate with each other. In this paper, node-to-node encryption is assumed rather than link-to-link encryption which is considered unsuitable in an open system environment [8]. Although the concepts of host master key, secondary key, and key-encryption key can be introduced to protect the generation and distribution of the keys [6], we do not address this issue. So far, three major schemes have been proposed.

The first is based on private-key ciphers where each pair of nodes must share a secret key. As a result, when there are N nodes, $N(N-1)/2$ keys need to be generated and distributed by a secure key manager; moreover, each node has to maintain N keys for all the possible communications. This problem is called the N^2 problem and should always be avoided if possible. The second is based on public-key ciphers [4], where each node selects its own key pair (E, D) and publishes E. When node A wants to communicate with node B, A encrypts the message using key E_B and sends the cipher text to B. The text can only be decrypted using the secret key D_B. However, public-key encryption and decryption are expensive and slow. The usual variation is to establish a private session key between each pair of nodes using a public key when they start communication. The drawback of this scheme is that there is a considerable delay before nodes can start real conversation; furthermore, each node still has to cache one session key for each other node it wants to talk to. The third is to use an authentication server to set up session keys [7], where the nodes must first authenticate them to the server and request that certain session keys to be set up. The server then generates and distributes the session keys required. This can be based either on a private-key cipher where each node shares a private key with the server, or on a public-key cipher where nodes use the server's public key. This has virtually the same properties as the second scheme with even greater delay.

All these come from the concept of secret key which people assume is *unique* and kept *completely secret*. The fundamental idea of the scheme we propose here,

in contrast to the concept above, is to *let each node have a set of keys of which it shares a distinct subset with every other node.* A key server does the generating work, perhaps using a dedicated processor or doing it in background or overnight, and distributes them as often as required. Upon receiving the keys, nodes can *immediately* start communicating to any other node without delay. Using this method, the total keys to be generated could be N instead of $N(N-1)/2$; and each node needs to hold only about $\sqrt{N}$ keys. Moreover, as we shall see later, the keys can be as short as 8 bits or even less.

We give a simple example first and then describe the general principle and two versions of the matrix scheme. We discuss the performance in terms of time needed to distribute keys, to start conversation between nodes, and storage requirement at every node. We also discuss the security in terms of the minimum number of colluding nodes needed to compromise a pair of nodes' conversation key, and the average risk level of the nodes. Finally, we discuss some possible extensions to enhance performance and/or security, and some potential applications.

After we finished our main work, we came across two other key distribution schemes, the predistribution scheme [5] and the symmetric key generation scheme [2]. They both use algebraic code and thus are costly. They are also common in that the threshold of the number of colluding nodes required to compromise a single key equals the threshold to compromise all the keys. In our scheme, different thresholds can be tuned independently to different needs. The matrix scheme is also more cost-effective. For example, analysis of [5] shows that the number of possible nodes or entities is smaller or equal to the number of key bits or the length of the Secret-Algorithm sent to each node. In our schemes however, the latter can be significantly smaller than the former. This means that to maintain a network of the same size and of at least the same security level, our scheme needs much less transmission and storage.

2 A Basic Scheme

In this section, we describe a very simple scheme to give an intuitive idea of our matrix distribution scheme. Assume there are N nodes, where $N = m^2$. Each node is assigned a position i, j, and is denoted as n_{ij}. Similarly, there are N keys

denoted as k_{ij}.

A key server generates the keys randomly and gives node n_{ij} a set of keys which consists of all the keys that are either on the same row or column as the node, $K_{ij} = \{k_{XY} | \ X = i, \ \ or, \ \ Y = j\}$. When node A (n_{ij}) wants to communicate to B (n_{uv}), it simply finds out B's position u, v and uses the keys k_{iv} and k_{uj} which are common between A and B to compose a conversation key, e.g., just concatenates the two keys. Then it could encrypt messages using this key and start the conversation. See the diagram below for an illustration.

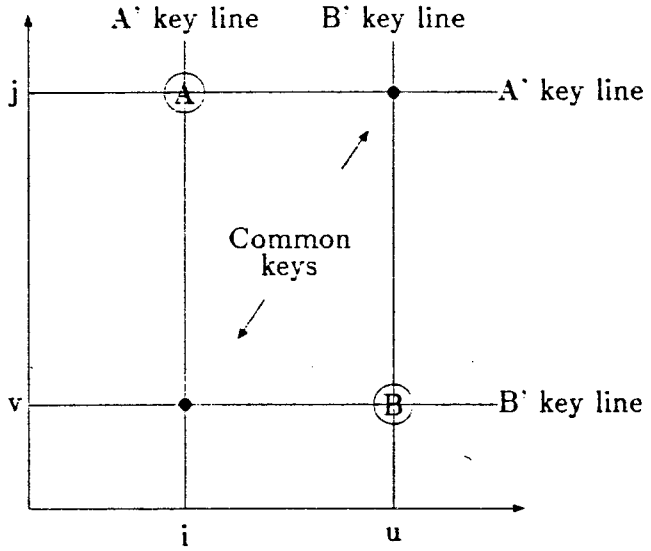

Fig. 1. The Key Map.

There are two major advantages of this scheme. First, there is virtually no delay when A wants to communicate to B. Those schemes where a session key has to be set up by a key server upon request may need several messages. Second, the storage requirement is reduced by a square root factor. The key server will generate N keys in total instead of $N(N-1)/2$ and each node receives and stores $2\sqrt{N}$ keys instead of N. However, there is a weak point in this scheme. If A and B are on the same line or column, any node on the same line or column could listen to their conversation because it shares the same common keys used between A and B. When A and B are not on the same line or column, the situation is better as two correctly positioned colluding nodes are needed to compromise the session key.

3 Principle

Assume there are N nodes, $n_i, i = 1, 2, \ldots, N$. Associated with each node n_i is a set of keys K_i and a published address P_i. Suppose there are two algorithms:

generate(N, R, C) : Given the number of nodes N, a random number R as salt, a set of constraints C, generates all K_i such that $\forall\, i, j, k, i \neq j \neq k : K_i \bigcap K_j \neq \emptyset$ and $K_i \bigcap K_j \neq K_i \bigcap K_k$. The keys are chosen at random.

compute(P_i, P_j, K_i) : Given two addresses and the key set of node n_i, derives a conversation key k_{ij}.

Note that by using a fixed order of the common keys independent of i or j, or a symmetric one-way function, we can make *compute* have the following property

$$compute(P_i, P_j, K_i) = compute(P_j, P_i, K_j) \quad i.e. \quad k_{ij} = k_{ji}$$

A key server generates and distributes all the keys. When A wants to communicate to B, the protocol is

A: Find out B's address and $k_{AB} = compute(P_A, P_B, K_A)$.
A: Encrypt message with k_{AB} and send it to B with a header specifying source A.
B: Calculate $k_{AB} = k_{BA} = compute(P_B, P_A, K_B)$.
B: Decrypt the message using k_{AB}, and continue conversation.

There might be an integrity check of the message sent to B; however, it does no harm to B if somebody else claims it is A as long as it has not cracked k_{AB}. Such an attack ought to be logged.

Based on this principle, we extend the basic scheme in two ways which results in the multi-line version and the multi-map version whose performances and security strengths are different. In the following sections, we describe, analyze, and compare them with existing major schemes.

4 Multi-Line Version

This version is achieved by allocating more key lines to each node instead of only two as in the basic scheme.

4.1 Definitions

Assume there are N nodes where $N = m^2$. A communication map is defined as a $m \times m$ matrix on which each point has an address P_{ij} and corresponds to a node n_{ij}, where $i, j = 1, 2, \ldots, m$. A key map is also defined as a $m \times m$ matrix where point i, j corresponds to a key k_{ij}. The key set sent to node n_{ij} is

$$K_{ij} = \{k_{XY} |\ Y - j + c_l(X - i) = 0 \mod (m)\}$$

$$l = 1, 2, \cdots, t \ \ and \ \ c_p \neq c_q \ \ when \ \ p \neq q$$

The key set is a set of t lines on the key map all passing through point i, j. These keys can be stored in t tables where each entry is indexed by the value $X - i$. This makes the calculation of common keys easier as we will see in *compute*.

An algorithm *Generate* generates m^2 random keys and puts one on each point on the key map. It calculates K_{ij} from the simple definition, sends it to the node at the address P_{ij}.

An algorithm *Compute* takes a pair of addresses on the communication map as input, say P_{ij} and P_{uv}, solves $t(t-1)$ linear equation groups each of which has the form

$$Y - j + c_p(X - i) = 0 \mod (m)$$

$$Y - v + c_q(X - u) = 0 \mod (m)$$

$$p, q = 1, 2, \cdots, t \ \ and \ \ p \neq q$$

The solution is in fact very simple:

$$X - i = (c_q(i - u) + v - j)/(c_p - c_q) \mod (m)$$

Using a table look up for the needed reciprocals will speed up the calculation. The solutions (X, Y) are positions on the communication map of keys that nodes n_{ij} and n_{uv} have in common. In fact to find out a common key, there is no need to calculate Y because p, q determines the key line table and the $X - i$ is the index of the key. *Compute* then composes a conversation key from these keys, possibly using a one-way function. Note when m is chosen to be a prime, each equation group must have exactly one solution. Other choices of m also work. For example, to let the machine implementation be easier and faster, it may be desirable to

make m a power of 2. In this case, when $(c_p - c_q)$ is odd, there is exactly one solution; so if we choose half c_i's to be even and half odd, there are at least $t^2/2$ common keys between each pair of nodes. When it is even and the numerator in the solution equation is odd, there is no solution and when both are even there might be none, one, or more than one solution. These extra keys can be used to enhance security, although it is unlikely to be worth the complications.

4.2 Performance

Property 1 *The key server has to generate N keys.*

Property 2 *The key server needs to send to each node $t\sqrt{N}$ keys.*

Property 3 *Assuming m is prime and $c_p \neq c_q$ when $p \neq q$, two different nodes have in common either $t(t-1)$ or $\sqrt{N} + (t-1)(t-2)$ distinct keys.*

Proof. Every two non-parallel lines always meet at exactly one point. This is ensured by the fact that m is a prime. Observe that two nodes cannot have more than one common line, and it is only at the two points where the nodes sit that more than two key lines can intersect, thus if they do not have a common line they have exactly $t(t-1)$ intersecting points; otherwise, they have

$$m + (t-1)(t-2) = \sqrt{N} + (t-1)(t-2)$$

such points. □

Property 4 *The time needed to set up a conversation key is the time complexity of compute and is $O(t^2)$.*

Property 5 *Each node stores $bt\sqrt{N}$ bits where b is the key length. The key server needs temporary storage of bN bits.*

If conventional schemes are used, we have the following results. First, if the common keys retain the same length as in this version, the key server has to generate $N(N-1)/2$ keys, a total of $bt(t-1)N(N-1)/2$ bits; each node has to hold $bt(t-1)(N-1)$ bits. Second, the delay in a session key scheme when using a typical protocol is a few messages, which might involve expensive public key encryption and decryption.

4.3 Security

Property 6 *Assuming m is prime and c_i's are distinct, then for a particular pair of nodes, there exists a group of $t-1$ other nodes who, when colluding together, will be able to compromise the conversation key between the pair of nodes. There are many groups of t colluding nodes that can compromise A's conversations with any node. More colluding nodes may be needed when empty points are allowed on the communication map.*

Proof. There exists a group of $t-1$ nodes each of which has a distinct key line in common with node A and covers another distinct key on the tth line used in the conversation. This group is able to compromise A's particular key for that conversation. A group of colluding nodes, one on each of the t lines through A, can compromise all A's communications. □

We have not derived a satisfactory lower bound of the minimum number of colluding nodes needed to compromise a conversation key. We only have the following result that if $\{c_i\}$ is a super-increasing sequence and m is sufficiently large, then to compromise the key between two nodes, at least $\lceil t/3 \rceil$ colluding nodes are needed. However, we strongly doubt that this lower bound could be reached in most cases or the constraints on c_i's are necessary. Therefore we skip the tedious proof here.

Property 7 *Assuming node location and key distribution are uniform, the average number of colluding nodes needed to compromise a particular conversation is*

$$s = \log(1/d)/\log(1 - t/\sqrt{N}) \simeq 2\log(d)\sqrt{N}/t$$

where d is the minimum number of intersecting keys between the pair of nodes which is $t(t-1)$ when m is prime.

Proof. A third node's keys could cover a proportion of $t/\sqrt{N}$ of the whole key map and the same proportion of the common keys between another pair of nodes. To cover all the keys in order to compromise a conversation by a group of s colluding nodes, it is required that $d(1 - t/\sqrt{N})^s \leq 1$. □

Property 8 *Assuming node location and key distribution are uniform and the probability that not more than s colluding nodes can compromise any single conversation is p, then $s = (e^d p d!)^{1/d}$.*

Proof. A group of s colluding nodes have to cover d keys. Assuming a Poisson distribution and $s \ll d$, we have $p = \sum_{n=d}^{\infty} e^{-n} s^n / n! \simeq e^{-d} s^d / d!$. □

Property 9 *Assuming node location and key distribution are uniform, the average maximum number of common keys of a particular conversation that any group of n nodes can cover is approximately $nt(1 + t^2/\sqrt{N})$.*

Proof. Suppose it is up to the colluders to choose the c_i's and their own positions so that they can make the best choice. The best they can do is to choose the positions such that each of the group could cover at least t common keys, e.g., a whole distinct line which contains $t - 1$ common keys and a point on another line, and hope they can hit other common keys as many as possible. Suppose they choose n points such that each is on one line through A and covers one distinct common key on the remaining $t - n$ lines through A. Each node on such a point has $(t-n)(t-1) - 1$ extra intersections on $\sqrt{N}(t-n) - n$ remaining points of the remaining lines. Thus the number of total common keys the n nodes could hit is roughly

$$nt + n((t-n)(t-1) - 1)((t-n)(t-1) - n)/(\sqrt{N}(t-n) - n)$$

and is approximately $nt + nt^2(t-n)/\sqrt{N} \simeq nt(1 + t^2/\sqrt{N})$. □

The above result is dominated by nt if $t^2 \leq \sqrt{N}$ or $t \leq \sqrt[4]{N}$. Since the security requirement $nt(1+t^2/\sqrt{N}) \leq t^2$ *or* $n \leq t/(1+t^2/\sqrt{N})$ is normally to be assumed, the above result to some extent gives a relation between the choice of t and the security level n that could be expected. For example, if $t \leq \sqrt[4]{N}$ then we can expect that $n \leq t \leq \sqrt[4]{N}$.

5 Multi-Map Version

This version is achieved by generating more key maps but still allocating two key lines on each map to every node.

5.1 Definitions

Assume there are N nodes where $N = m^2$. A communication map is defined as a $m \times m$ matrix in which each point i, j has an address P_{ij} and corresponds to a node n_{ij} where $i, j = 1, 2, \ldots, m$. The lth key map is also defined as a $m \times m$ matrix in which each point $(i, j)^l$ corresponds to a key k^l_{ij}. The key set given to n_{ij} by the key server is

$$K_{ij} = \{k^l_{XY} \mid X = i + a_l j, \ \ or, \ \ Y = i + b_l j, \ \ \mathrm{mod}(m)\}$$

$$l = 1, 2, \ldots, t$$

where a's and b's are all distinct. K_{ij} consists of key rows or columns on the key maps, called key lines.

An algorithm *Generate* generates tm^2 random keys and puts one on each point on every key map. Then it trivially selects K_{ij} by the definition and sends it to the node at the address P_{ij}.

An algorithm *Compute* takes a pair of addresses on the communication map as input, say P_{ij} and P_{uv}, and finds out the common keys k^l_{pq} where

$$p = i + a_l j \ and \ q = u + b_l v, \ \ or, \ \ p = u + a_l v \ and \ q = i + b_l j \ \ , \ \ l = 1, 2, \ldots, t$$

It then uses the common keys to compose a conversation key, possibly using a one-way function. The above is easier to understand if m is a prime; but in fact other values of m work. For example, to make implementation easier and faster, it may be desirable to make m a power of 2; however, Property 12 in the next section will not hold in this case.

5.2 Performance

Property 10 *The key server has to generate tN keys.*

Property 11 *The key server needs to send to each node $t\sqrt{N}$ keys.*

Property 12 *Assume m is prime. Then if two nodes say A and B have a common key line on a key map, they do not have any common key lines on any other key maps.*

Proof. All the calculations are done mod(m). A's and B's key lines on the pth key map are indicated respectively by the row and column indices

$$X_A = i - a_p j, \quad Y_A = i - b_p j \quad and \quad X_B = u - a_p v, \quad Y_B = u + b_p v$$

Indices of key lines on the qth key map are

$$X'_A = i + a_q j, \quad Y'_A = i + b_q j \quad and \quad X'_B = u + a_q v, \quad Y'_B = u + b_q v$$

Assume A and B have two common key lines, one on the pth key map and one on the qth. There are four cases to consider, i.e., whether the common key lines are rows or columns on the pth and the qth. The first case is that they are columns on the pth and the qth. This means $X_A = X_B$ and $X'_A = X'_B$. Solving these equations we get $(a_p - a_q)(j - v) = 0$. Because a's are distinct, thus $j = v$ which further results in $i = u$. This says that A and B are the same node, a contradiction. The other three cases are similar to this one. □

Property 13 *Two different nodes have in common either* $2t$ *or* $2t + \sqrt{N} - 2$ *distinct keys.*

Proof. If A and B have a common key line on a key map, they have $\sqrt{N}$ common keys on this map. According to Property 12, they do not have common key lines on any other key maps, so they have 2 common keys on each other map. Thus the total is $\sqrt{N} + 2(t - 1) = 2t + \sqrt{N} - 2$. If they never have a common key line, they have in total $2t$ common keys. □

Property 14 *The time needed to set up a conversation key is the time complexity of compute and is* $O(t)$. *It is independent of the size of the network.*

Property 15 *Each node has to store* $bt\sqrt{N}$ *bits where* b *is the key length. The key server needs temporary storage of* btN *bits.*

If conventional schemes are used, there are the following results. First, if the common keys retain the same length as in this version, the key server has to generate $N(N - 1)/2$ keys, a total of $btN(N - 1)$ bits; each node has to hold $2bt(N - 1)$ bits. Second, when a server is required to set up a session key using a typical protocol, the delay could be a few messages, which might involve expensive public key encryption and decryption. It will also depend on some environmental factors as the size of the network, the location of the key server.

5.3 Security

To simplify the results and the proof, we assume that when A and B have a common key line, they only use two of the $\sqrt{N}$ common keys.

Property 16 *When a particular pair of nodes communicate, any other node can have at most two of the common keys of the pair of nodes.*

Proof. Assume node C wants to listen to the conversation between A and B. According to Property 12, C could at most have one common key line with A or B once on all the key maps, so C could at most have two keys which are used between A and B. □

Property 17 *To compromise the key between two nodes, at least t other colluding nodes are needed.*

Proof. Considering that two nodes have at least $2t$ common keys, this property is a straightforward corollary of the last one. □

Note that in general case, this lower bound proven is also an upper bound because there is always a group of t nodes who when colluding together can compromise the conversation key. However, as stated before, empty points could be specially allocated on the communication map to enhance security.

Property 18 *Assuming that node location and key distribution are uniform, the average number of colluding nodes needed to compromise a particular conversation is*

$$s = \log(2t)/\log(1 - 2/\sqrt{N}) \simeq \log(2t)\sqrt{N}.$$

Proof. A third node C's probability to have a common key used between A and B is $2/\sqrt{N}$. To cover all the keys in order to compromise the conversation by s nodes, it is required that $(1 - 2/\sqrt{N})^s 2t \leq 1$ □

Property 19 *Assuming that node location and key distribution are uniform and the probability that no more than s colluding nodes can compromise any single conversation is p, then* $s = (e^{2t}p(2t)!)^{1/(2t)}$

Proof. s colluding nodes have to know all the $2t$ keys. Assuming a Poisson distribution, we have $p = e^{-2t}s^{2t}/(2t)!$ □

6 Discussion

A hybrid version is to choose pairs of lines from the multi-line version but with a separate key map for each pair, and also choose c_l's to make the difference between the c_l pair to be 1 to eliminate division. Proofs of security bounds similar to that in the multi-map version exist. The only difference is that now two nodes can have in total at most two common key lines. An important feature of this hybrid version is that the security bounds for the multi-map version still hold even when m is a power of 2. This yields a very fast and secure scheme. Note that if the difference is a constant other than 1, an initial multiplication is sufficient and the rest of the key calculation can be done by adding a constant difference. See the appendix for a sample program of this hybrid version.

Performance can be enhanced in many ways. For example, since communicating nodes construct a conversation key from the common keys, it is in theory sufficient if every key on the key map is very short, because there are enough bits in common from which to construct the conversation key. Suppose the conversation key is required to be 64 bits long, then in the multi-line version, let $t = 9$ and $b = 1$, every pair of nodes has at least 72 key bits in common while in the multi-map version, let $t = 16$ and $b = 2$, the number of common key bits is at least 64. Thus the number of bits that the key server has to generate and distribute could be very small. However, having longer keys results in a higher level of security. We can also simplify the computation in the multi-line version by choosing c_l's as consecutive numbers so that the computation cycle can simply step through the tables. Moreover, no more than $t(t-1)/2$ reciprocals are needed in the multi-line version and if the a's are consecutive integers, no more than t reciprocals are needed in the hybrid version. These reciprocals can be held in a table to speed up the calculation. Note much of the above still holds when $m = 2^n$.

Security can also be enhanced in many ways. First for instance, from the lower bounds stated in previous sections, we could infer that the larger t we have, the more colluding nodes are required to compromise a conversation. Therefore, if the amount of key storage or transmission is limited, we can always let b be as small as possible to achieve the maximum security. However, increasing b reduces the computation time. These are clearly reflected in the sample figures given in the

appendix. Second, keys generated and used are not necessarily equal in length. In fact, longer keys could be allocated to certain points to enhance the security of certain important nodes. As a more specific example, in the multi-map version, if *all* nodes who have a common key line on the communication map require higher *mutual* security, they can include up to $b(\sqrt{N}-2)$ extra bits in their conversation key. These extra bits are already available in their common key set but not used in previous schemes. In this case, according to the security proof, an extra number of $\sqrt{N}-2$ colluding nodes might be needed, and *at least* $\sqrt{N}-2t$ extra nodes are needed to attack their mutual communications. This is a big gain with little extra effort.

Third, two communicating nodes could use part of the common keys they have to compose the conversation key. The selection could be made according to some preset rule or a notice published by the key server. This virtually changes all the conversation keys without physically sending new ones. This makes attack more difficult. For example, in the multi-line version, if $t = 9$, then there are 72 common keys. If 64 of them are selected each time to compose the conversation key, there are at least $\binom{72}{64} \geq 2^{33}$ possible selections. One more parameter could be added to *compute* to choose an appropriate subset of common keys. The parameter could be agreed by the concerned nodes before communication or transmitted at the same time. Moreover, when a conversation key is to be composed, the common keys can be run through a one-way function. This separation by a secure one-way function is important because now knowing some common keys is only useful in exhaustively searching the common keys space. It does not help break the encryption algorithm or exhaustively search the conversation key space. When the one-way function is symmetric with respect to the dictionary order, it is not necessary to run the common keys through it in a fixed order, which speeds up the algorithm considerably. For instance, in the example 3 in the appendix for multi-line version, there are 2048 common key bits. Suppose we simply fold it 5 times to get a 64 bit key, each such bit is the exclusive-or of 32 common key bits and knowing less than 32 bits reveals nothing about the final key bit.

A concept of logical positions of nodes could be introduced. Now the position of node n_{ij}, P_{ij}, is not simply its physical address $address(n_{ij})$, but is assigned by

the key server using an algorithm as follows

$$P_{ij} = convert(address(n_{ij}), R)$$

where R is salt. When node A wants to talk to B, it calculates B's logical position P_B

$$P_B = convert(address(n_A), R)$$

and computes the conversation key from P_A, P_B, using algorithm *compute*. By doing this, the key server can change the address relations between nodes. It has the advantage that a group of colluding nodes cannot always compromise a conversation key between a particular pair of nodes. Every time the key server changes the logical relations, part or all the keys might be required to be changed at the same time; but not vice versa. *Convert* could be any one-one mapping from the set of physical addresses in the environment to the set of points on the communication map. R is taken to randomize the mapping. Note that changing some or all of the a's, b's, and c_l's at each key change will also change the logical relations between nodes and thus increase security.

The frequency of key change is based on the security level wanted. Normally, the selection of key bits is changed most frequently; the whole or part of the key map is changed less frequently; the values of a's, b's, and the communication map are changed the least frequently. As fewer bits are to be generated and changed than in conventional schemes they can be changed more frequently. And also note that an attack by outsiders meets no fewer difficulties than in other existing schemes. An insiders attack needs the collusion of at least a certain number of nodes which have to be located on the right points on the communication map to compromise a particular conversation key. When the communication map is changed as described above, another set of colluding nodes are needed, which is awkward for the colluders.

Finally, all the security results as lower bounds are derived on the assumption that there is one node at each point on the communication map. It may not remain true if empty points are allowed. In fact, empty points could be specially allocated to enhance security, for example, to protect some vital nodes, or to segregate notes which have unclean records. There is little loss in choosing $N = m_1 m_2$, a product of two different prime numbers, but the extra complications seem not worthwhile.

We summarize a comparison of the multi-line and the multi-map version. N, b, and t have the same meaning as before.

	Multi-line version	Multi-map version
Server generated key bits	bN	btN
Node stored key bits	$bt\sqrt{N}$	$2bt\sqrt{N}$
Common key bits	$bt(t-1)$	$2bt$
Time to find common keys	$O(t^2)$	$O(t)$
Minimum colluding nodes	$\lceil t/3 \rceil$	t
Average colluding nodes for a particular conversation	$2\log(d)\sqrt{N}/t$	$\log(2t)\sqrt{N}$
Average colluding nodes for any conversation	$(e^d pd!)^{1/d}$	$(e^{2t}p(2t)!)^{1-2t}$

Our scheme can be adopted in many applications. The general characteristic of suitable environments is that most nodes involved are trusted, although it may or may not be always clear as which nodes are trusted or not; particularly where an acceptable upper limit of the number of colluding nodes has been decided according to the particular application. A sample environment is a switch center where communications from all incoming ports to all outgoing ports are to be protected. Here the majority of the employees are trusted and activities are supervised. Furthermore, our protocol reduces hand shakes so that useful messages pass much faster than where exchanges are needed before the real message can be securely passed. This is good where starting conversations without delay is appreciated or essential.

As an example, authentication protocols which in future will be required extensively [1] can be built using the matrix principle. Another example is to support the unconditional sender and recipient untraceability [3]. In there, when there is a subset of participants who each participant believes are sufficiently unlikely to collude, such as of conflicting interests, each other participant can then share a key with every member of this subset, which guarantees that the participants outside the subset are untraceable. To implement this key sharing, our basic scheme is sufficient. We could simply put the participants of the subset on to a single line on the communication map. Of course the cheating problem has to be dealt with separately. Our scheme might also be applicable to other key sharing topology in [3] which remains to be investigated.

7 Summary

A new key distribution scheme is presented. It is based on the idea that lets each node have a set of keys of which it shares a distinct subset with every other node. Two versions of the scheme are given. The scheme suits an environment where there is a certain level of trust among the insiders i.e. the nodes. The security property to an outsider remains identical to that of other existing schemes. Moreover, there is virtually no delay when nodes want to start communication so it supports immediate conversation. Finally, compared with existing schemes, the number of keys to be generated and stored are reduced by a square root factor so it needs much less transmission and storage. The matrix scheme has many potential applications.

Appendix

A Sample Programs

To show the simplicity of our schemes, we give three C programs which are parts of the *compute* to find the common keys.

A.1 Multi-Line Version

Assume that node n_{ij} wants to find out the common keys between itself and node n_{uv}. Assume that its keys given by the key server are stored in array k[l,x-i], where l = 1,2, $\cdots$, t indicates the l*th* key line, and x-i = 1,2, $\cdots$, m is the index of the keys on the line. Let $m = 2^n$; t is even; c[1], $\cdots$, c[t/2] are odd, c[t/2 +1], $\cdots$, c[2t] are even; and a table contains all the reciprocals

$$r[i-j] = 1/(c[i] - c[j]) \bmod 2^n$$

The common keys found are to be stored in byte array c[s], s = 1,2, $\cdots$, t(t-1). The changes when $b \neq 8$ are obvious but tedious. The program does not include the part to order the common keys. One simple way to do this is to sort the keys according to the indices as. If $(i > u \mid (i = u)\&(j > v))$ then swap the two bytes. The algorithm follows

```
common(i,j,u,v,t,m,k,ck,r)
int i,j,x,y,u,v,t,m;
char k[], ck[],r[];  /* Byte arrays */
{
    int p,q,w, s = 0;

    for(p = 1; p <= t/2; ++p)
       for(q = t/2 + 1; q <= t; ++q)
       {
       w = r[p-q];
       x = ((i - u)*w) % m;
       y = ((v - j)*w) % m;
       ck[s] = k[p, (c[q]*x + y) % m];
       ck[s + 1] = k[q, (-c[p]*x - y) % m];
       s += 2;
       }
}
```

A.2 Multi-Map Version

Assume keys given to node n_{ij} are stored in array k[l,p,q], where l indicates the *lth* key map, p = 0 or 1 indicates the key row or column on the key map. q = 1,2, $\cdots$, $\sqrt{N}$ is the index of the keys on the line, and m = $\sqrt{N}$ is assumed a prime. Other assumptions are the same as in the multi-line version.

```
common(i,j,u,v,n,t,a,b,k,ck)
int i,j,u,v,m,t,a[],b[],k[],ck[];
{
    int l,s = 0;

    for(l = 1; l <= t; ++l)
    {
    ck[s] = k[l,0,(u + b[l]*v) % m];
    ck[s + 1] = k[l,1,(u + a[l]*v) % m];
```

```
    s += 2;
    }
}
```

A.3 Hybrid Version

This simple hybrid version shows that the multiplications in the previous programs can be eliminated in a simple way to speed up the algorithm while retaining the same security strength. Let node x_0, y_0 have keys at points

$$(x, y_0 + s(x - x_0)) \ \ where \ \ x = 0, \cdots, m-1 \ \ , s = 0, \cdots, t$$

On the lth key map the node has the key lines indicated by $s = l, l+1$, $l = 0, \cdots, t-1$. A position of a common key on a map is the intersection of the two lines

$$y = y_0 + s(x - x_0)$$

$$y = y_1 + (s+1)(x - x_1)$$

The solution is

$$x = y_0 - y_1 + s(x_1 - x_0) + x_1$$

Because s steps from 0 to $t-1$, the multiplication in the above formula can be done by an addition. We also use addition as a symmetric function to compose the conversation key. This avoids ordering the keys so that there is no need to swap the key bytes. Assume keys are stored in array k and the common keys are stored in array ck. The inner cycle of the calculation is

```
{
x = (y[0] - y[1] + x[1]) & (m-1);
/* this masking is mod (m) since m is a power of 2 */
for (s = 0; s < t; s++)
  {
  ck[s] = k[s,x] + k[s, (x[1] + x[0] -x) & (m-1)]
  x = (x + x[1] - x[0]) & (m-1)
  }
}
```

The cycle as given has no multiplications and can be done rapidly even on a simple micro.

B Examples

Here we choose m a power of 2 thus there are $bt^2/2$ common key bits in the multi-line version.

Multi-line version example	1	2	3	4
Number of nodes	2^{20}	2^{16}	2^{16}	2^{10}
Number of key lines	8	8	64	4
Key length	4	4	1	8
Server generated key bits	2^{22}	2^{19}	2^{22}	2^{15}
Node stored key bits	2^{15}	2^{13}	2^{14}	2^{11}
Common key bits	128	128	2048	64
Time to find common keys	32	32	2048	8
Minimum colluding nodes	3	3	22	1

Multi-map version example	1	2	3	4
Number of nodes	2^{20}	2^{16}	2^{10}	2^{10}
Number of key maps	16	64	32	48
Key length	4	1	1	1
Server generated key bits	2^{26}	2^{22}	2^{15}	$3{\cdot}2^{14}$
Nodes stored key bits	2^{17}	2^{14}	2^{10}	$3 \cdot 2^{9}$
Common key bits	128	128	64	96
Time to find common keys	16	64	32	48
Minimum colluding nodes	16	64	32	48

References

[1] A.D. Birrell, B.W. Lampson, R.M. Needham, and M.D. Schroeder, A Global Authentication Service Without Global Trust, *Proceedings of the IEEE Symposium on Security and Privacy*, IEEE, New York, 1986, pp. 223-230.

[2] R. Blom, An Optimal Class of Symmetric Key Generation Systems, *Advances in Cryptology: Proceedings of Eurocrypt 84*, Lecture Notes in Computer Science, vol. 209, Springer-Verlag, Berlin, 1984, pp. 335-338.

[3] D. Chaum, The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability, *Journal of Cryptology*, vol.1, no.1 (1988), pp. 65-75.

[4] W. Diffie and M.E. Hellman, New Directions in Cryptography, *IEEE Transactions on Information Theory*, vol. IT-22, no.6 (1976), pp. 644-654.

[5] T. Matsumoto and H. Imai, On the Key Predistribution System : A Practical Solution to the Key Distribution Problem, *Advances in Cryptology: Proceedings of Crypto 87*, Lecture Notes in Computer Science, vol. 293, Springer-Verlag, Berlin, 1987, pp. 185-193.

[6] S.M. Matyas and C.H. Meyer, Generation, Distribution, and Installation of Cryptographic Keys, *IBM Systems Journal*, vol.17, no.2 (1978), pp. 126-137.

[7] R.M. Needham and M.D. Schroeder, Using Encryption for Authentication in Large Networks of Computers, *Communications of the ACM*, vol.21, no.12, (1978), pp. 993-999.

[8] V.L. Voydock and S.T. Kent, Security Mechanisms in High-Level Network Protocols, *ACM Computing Surveys*, vol.15, no.2, (1983), pp. 135-171.