J. Cryptology (1990) 2: 51–59

Journal of Cryptology

A Matrix Key-Distribution Scheme[1]

Li Gong and David J. Wheeler
Computer Laboratory, University of Cambridge,
Cambridge CB2 3QG, England

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 generated is proportional to the number of nodes. Moreover, two nodes can start a session with virtually no delay. The scheme suits an environment where there is a certain level of trust among the insiders. The security property to an outsider remains identical to that of other existing schemes. Two versions of the scheme are given. Analysis of security and performance shows it is a practical solution to some key-distribution problems.

Key words. Communication security, Private-key cipher, Session key, Key distribution.

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 a major problem in an environment where a large number of nodes communicate with each other. In this paper we assume node-to-node encryption rather than link-to-link encryption which is consider unsuitable in an open-system environment [6]. We do not address this issue of enhancing security by using host master keys, secondary keys, or key-encrypton keys [4].

Proposed so far are three major key-distribution schemes. The first is to use a private-key cipher system and arrange that each pair of nodes share a different secret key. Thus N nodes require that $N(N-1)/2$ keys be generated and distributed by a secure key manager. Each nodes has to maintain N keys for all possible communications. This is known as the N^2 problem.

The second is to use a public-key system [2]. 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 ciphertext to B. The ciphertext can only be decrypted using the secret key D_B. This avoids the N^2 problem but public-key encryptions and decryptions are expensive and slow. A usual variation

[1] Date received: March 31, 1988. Date revised: August 17, 1989.

is to establish a private session key between each pair of nodes using a public key when they start communication. A drawback of such schemes is that there is a considerable delay before nodes can start the session. Each node has to cache one session key for eah other node it wants to talk to.

The third scheme is to use an authentication server to set up session keys [5]. When a pair of nodes want to communicate, they first authenticate themselves to the server. The server then generates and distributes a session key. Authentication protocols can be based either on a private-key cipher or on a public-key cipher. The delay here is even greater than in the previous scheme.

All these come from the concept of a 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 generates the keys and distributes them as often as required. Upon receiving the keys, nodes can *immediately* start communicating to any other node. Our scheme requires a total of $O(N)$ instead of $N(N-1)/2$ keys. Each node needs to hold only $O(\sqrt{N})$ keys. Moreover, as we shall see later, the keys can be as short as 8 bits or even less. The scheme introduces an extra risk that it is possible for nodes to collude and compromise the session keys of other nodes. However, it can be arranged that the minimum number of colluding nodes required is acceptable.

There are two other key-distribution schemes, the predistribution scheme [3] and the symmetric key generation scheme [1]. They both use expensive algebraic codes. 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 the thresholds are different and it is possible to give higher security to particular groups of nodes. The matrix scheme is also more cost-effective. For example, analysis of [3] shows that the number of possible nodes cannot be greater than the number of key bits sent to each node. In our scheme the former can be significantly greater than the latter. 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.

We first illustrate the new idea with a simple example. Then we describe the general principle and two versions of the matrix scheme. We analyze the security and performance and discuss some possible extensions to enhance them.

2. A Basic 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 at random 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, $K_{ij} = \{k_{xy} | x = i \text{ or } y = j\}$. 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 session key, e.g., just concatenates the two keys (see Fig. 1).

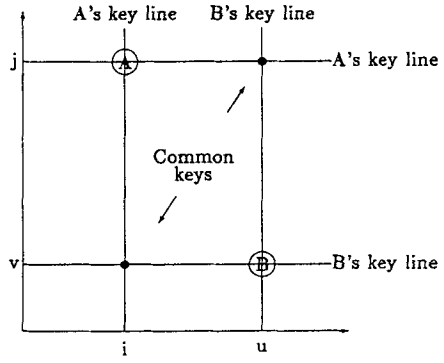

Fig. 1. The key map.

Two properties are interesting. First, two nodes can start a session virtually without delay. 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 nodes receives and stores $2\sqrt{N}$ keys instead of N. However, this basic scheme is weak in that if A and B are on the same line or column, any node on the same line or column could compromise the session 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 situtaion 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.

Given the number of nodes N, a random number R as a seed, a set of constraints C, an algorithm *generate*(N, R, C) generates the sets of random keys K_i such that

$$\forall i, j, k, \quad i \neq j \neq k: \qquad K_i \cap K_j \neq \varnothing \quad \text{and} \quad K_i \cap K_j \neq K_i \cap K_k.$$

Given two addresses and the key set of node n_i, an algorithm *compute*(P_i, P_j, K_i) derives a session key k_{ij}. Using a fixed order of the common keys independent of i or j, or a symmetric one-way function, we can arrange that

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

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

1. A find out B's address and $k_{AB} = \textit{compute}(P_A, P_B, K_A)$.
2. A encrypts with k_{AB} and sends to B specifying source A.
3. B calculates $k_{AB} = k_{BA} = \textit{compute}(P_B, P_A, K_B)$.
4. B decrypts using k_{AB} and continues if the decryption is successful.

Based on this principle, we extend the basic scheme in two direcions. This results in the multiline version and the multimap version.

4. Multiline Version

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

Assume there are N nodes where $N = m^2$. A communication map is defined as an $m \times m$ 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 an $m \times m$ matrix where point i, j corresponds to a key k_{ij}. The key set sent to node n_{ij} is

$$K_{ij} = \{k_{xy} | y - j + c_l(x - i) = 0 \bmod(m)\},$$

$$l = 1, 2, \ldots, t \quad \text{and} \quad c_p \neq c_q \quad \text{when} \quad p \neq q.$$

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 later.

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}.

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

$$y - j + c_p(x - i) = 0 \quad \bmod(m),$$

$$y - v + c_q(x - u) = 0 \quad \bmod(m),$$

$$p, q = 1, 2, \ldots, t \quad \text{and} \quad p \neq q.$$

The solution is in fact very simple:

$$x - i = (c_q(i - u) + j - v)/(c_p - c_q) \quad \bmod(m).$$

The solutions (x, y) are positions on the communication map of keys that nodes n_{ij} and n_{uv} have in common. 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 session key from these keys, possibly using a one-way function. Using a table look up for the needed reciprocals will speed up the calculation.

When m is chosen to be a prime, each equation group has exactly one solution. Other choices of m also work. For example, to let 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_l'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 complication.

Property 1. *The key server has to generate N keys and send to each node* $t\sqrt{N}$ *keys.*

The time needed to set up a session key is the time complexity of algorithm compute which is $O(t^2)$.

Property 2. *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 nonparallel lines always meet at exactly one point because 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 3. *Assume m is prime and the* c_l*'s are distinct, then for a particular pair of nodes, there exist a number of groups of* $t-1$ *other nodes who, when colluding together, will be able to compromise the session key between the pair of nodes. There are a number of groups of t colluding nodes that can compromise A's session keys with any node.*

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 session. This group is able to compromise A's particular key for that session. 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 session key. The only result is that if $\{c_l\}$ is a superincreasing 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 case or that the constraints on the c_l's are necessary. Therefore we skip the tedious proof here. Note more colluding nodes may be needed when empty points are allowed on the communication map.

5. Multimap Version

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

Assume there are N nodes where $N = m^2$. A communication map is defined as an $m \times m$ 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 an $m \times m$ matrix in which each point $(i, j)^l$ corresponds to a key k_{ij}^l. The key set given to n_{ij} is

$$K_{ij} = \{k_{xy}^l | x = i + a_l j \text{ or } y = i + b_l j, \bmod(m)\}, \qquad l = 1, 2, \ldots, t,$$

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

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

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

$$p = i + a_l j \quad \text{and} \quad q = u + b_l v \qquad \text{or} \qquad p = u + a_l v \quad \text{and} \quad q = i + b_l j,$$
$$l = 1, 2, \ldots, t.$$

It then uses the common keys to compose a session key, possibly using a one-way function.

Note that if $a_l = 0, 2, \ldots, 2t - 2$ and $b_l = a_l + 1$, then the above computation reduces to selecting keys from consecutive key lines by an index starting with value u and being stepped by v modulo m.

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, the following analysis may not hold in this case.

Property 4. *The key server has to generate tN keys and send to each node $t\sqrt{N}$ keys. The time needed to set up a session key is the time complexity of algorithm compute and is $O(t)$.*

Property 5. *Assume m is prime. If two nodes 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). Assume A and B have two common key lines, one on the pth key map and one on the qth. A's and B's key lines on the pth key map are indicated respectively by the row and column indices

$$x_A = i + a_p j, \quad y_A = i + b_p j \qquad \text{and} \qquad x_B = u + a_p v, \quad y_B = u + b_p v.$$

Indices of key lines on the qth key map are

$$x'_A = i + a_q j, \quad y'_A = i + b_q j \qquad \text{and} \qquad x'_B = u + a_q v, \quad y'_B = u + b_q v.$$

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. Thus $x_A = x_B$ and $x'_A = x'_B$. Solving these equations we get $(a_p - a_q)(j - v) = 0$. Because the a's are distinct, $j = v$ 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 6. *Two 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 the previous property, they do not have common

key lines on any other key maps, so they have two 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. □

To simplify the following results and the proofs, we assume that when A and B have a common key line, they only use two of the $\sqrt{N}$ common keys selected by the solutions given above.

Property 7. *When a pair of nodes communicate, any other node can have at most two of the common keys shared between the pair.*

Proof. Let C be a different node. From Property 5, C can have at most one common line with A and one with B on all maps. So C can have at most two keys which are used between A and B. □

Property 8. *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 previous one. □

In general this lower bound is also an upper bound because there is always a group of t nodes who when colluding together can compromise the session key. However, as stated before, more colluding nodes may be required if empty points could be specially allocated on the communication map to enhance security.

6. Enhancing Performance

We can simplify the computation in the multiline version by choosing c_l's as consecutive numbers so that the computation cycle can simply step through the tables and no more than $t-1$ reciprocals are needed. These reciprocals can be held in a table to speed up the calculation. Much of the above still holds when $m = 2^n$.

Since communicating nodes construct a session key from the common keys, it is sufficient if every key on the key map is very short, because there are enough bits in common from which to construct the session key. Thus the number of bits that the key server has to generate and distribute could be very small. Suppose b is the common key length and the session key is required to be 64 bits long, then, in the multiline version, let $t = 9$ and $b = 1$, every pair of nodes has at least 72 key bits in common while in the multimap version, let $t = 16$ and $b = 2$, the number of common key bits is at least 64. Of course making the total number of key bits larger by increasing t or b results in a higher level of security.

We summarize a comparison of the multiline and the multimap version (N, b, and t have the same meaning as before):

	Multiline version	Multimap 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	$[t/3]$*	t

* If $\{c_i\}$ forms a superincreasing sequence.

7. Security Considerations

When b is very small, we have to take into account the colluders' ability to perform exhaustive search. For example, in the multimap version, suppose m colluders can search up to s bits. To compromise a session key it is required that

$$2bm + s \geq 2bt, \qquad \text{i.e.,} \quad m \geq t - s/2b.$$

Thus the threshold is lowered from t to $t - s/2b$. When the amount of key storage or transmission is fixed, i.e., $b \times t$ is fixed, a smaller b increases the threshold. This also increases t and thus the time complexity to compute session keys. Another way to look at this is that, since security requires $2bt > 2bm + s$ and the amount of key storage or transmission is proportional to bt, a smaller b reduces the storage and transmission.

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 multimap 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 session key. These extra bits are already available in their common key set but not used in our previous versions. In this case, according to the security proof, an extra number of $\sqrt{N} - 2$ colluding nodes not on the common key line might be needed, and at *least* $\sqrt{N} - t$ extra such nodes are needed to attack their mutual communications. This is a big gain with little extra effort.

Note if a one-way function is used to compose a session key, it can ensure that attacking the session key itself does not help in attacking the common keys. 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.

The security results are derived on the assumption that there is one node at each point on the communication map. They may become stronger 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.

Or alternatively, it can be arranged that a node only receives the common keys it would use to talk to those nodes to which it is allowed to communicate. The little extra work by the key server, transparent to the concerned nodes, is to find out the

allowed keys for a node and replace the other keys by random bits, before sending them off. This little effort gives an extra security feature.

We can easily generalize to multimap multiline versions, which allow different compromises between security and node key set length. There is little loss in choosing $N = m_1 m_2$, a product of two different prime numbers, but the extra complications do not seem worthwhile.

A concept of logical positions of nodes could be introduced. Now the position of a node is not simply its physical address, but is assigned and can be changed by the key server. This has the advantage that to compromise a session key between a pair of nodes, a different group of colluding nodes is needed when logical relations change. This makes insider attacks more difficult.

The frequency of key change is based on the security level wanted. Normally, the whole or part of the key map is changed less frequently; the values of the a's, the 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.

Finally, in the multimap version it can be arranged that some key maps are issued by other servers so that no single server can eavesdrop. The total amount of key traffic remains unchanged.

References

[1] 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.

[2] W. Diffie and M. E. Hellman, New Directions in Cryptography, *IEEE Transactions on Information Theory*, vol. 22, no. 6, December 1976, pp. 644–654.

[3] T. Matsumto and H. Imai, On the Key Predistribution System: A Practical Solutions 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.

[4] S. M. Matyas and C. H. Meyer, Generation, Distribution, and Installation of Cryptography Keys, *IBM Systems Journal*, vol. 17, no. 2, 1978, pp. 126–137.

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

[6] 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.