Logarithmic Keying of Communication Networks

Mohamed G. Gouda[1], Sandeep S. Kulkarni[2], and Ehab S. Elmallah[3]

[1] University of Texas at Austin
gouda@cs.utexas.edu
[2] Michigan State University
sandeep@cse.msu.edu
[3] University of Alberta
ehab@cs.ualberta.ca

Abstract. Consider a communication network where each process needs to securely exchange messages with its neighboring processes. In this network, each sent message is encrypted using one or more symmetric keys that are shared only between two processes: the process that sends the message and the neighboring process that receives the message. A straightforward scheme for assigning symmetric keys to the different processes in such a network is to assign each process $O(d)$ keys, where d is the maximum number of neighbors of any process in the network. In this paper, we present a more efficient scheme for assigning symmetric keys to the different processes in a communication network. This scheme, which is referred to as logarithmic keying, assigns $O(\log d)$ symmetric keys to each process in the network. We show that logarithmic keying can be used in rich classes of communication networks that include star networks, acyclic networks, limited- cycle networks, and planar networks.

Keywords: Secure communications, symmetric keys, keying scheme.

1 Introduction

A communication network consists of processes and connecting channels such that for each pair of processes p and q, either there are no connecting channels between p and q, or there is a single two-way channel between p and q. Two processes in a communication network are called neighbors iff there is a two-way channel between the two processes in the network. Two neighboring processes can exchange messages over the two-way channel between them. A communication network is said to be of degree d iff the network has a process that has exactly d neighbors, and each process in the network has at most d neighbors.

Let p and q be two neighboring processes in a communication network and assume that both p and q know a symmetric key s and that no other process in the network knows s. In this case, each exchanged message between p and q can be encrypted using s before it is sent (by p or q) and can be decrypted using s after it is received (by q or p, respectively) in order to guarantee the confidentiality of the communication between p and q. This simple arrangement suggests that if a process p in a communication network has d neighbors, then p needs to store and use d symmetric keys in order to guarantee the confidentiality of its d communications with each one of its neighbors.

We refer to any keying scheme, where $O(d)$ symmetric keys are assigned to each process in a communication network whose degree is d, as a *linear keying* scheme.

As it happened, recently published results [1], [2], and [3] have shown that linear keying is not the most efficient scheme, for assigning symmetric keys to the processes in a communication network, in the case where the network is fully connected (i. e. where each two distinct processes in the network are neighbors). In [1], Gong and Wheeler described a keying scheme where $O(\sqrt{d})$ symmetric keys are assigned to each process in a fully connected communication network. In [2], Kulkarni, Gouda, and Arora described a variation of the scheme in [1], and showed that this scheme is optimal if each pair of distinct processes share no more than two symmetric keys. In [3], Aiyer, Alvisi, and Gouda described a keying scheme where $O(\log^2 d)$ symmetric keys are assigned to each process in a fully connected network. They also showed, using a probabilistic but non-constructive argument, that there exists a keying scheme where $O(\log d)$ symmetric keys are assigned to each process in a fully connected communication network. Note that all these results apply only communication networks that are fully connected. So far, there are no corresponding results to arbitrary communication networks; hence, this paper.

We refer to any keying scheme, where $O(\log d)$ symmetric keys are assigned to each process in a communication network, whose degree is d, as a *logarithmic keying* scheme.

In this paper, we describe logarithmic keying schemes for assigning symmetric keys to the processes in rich classes of communication networks, which include star networks, acyclic networks, cycle-limited networks, and planar networks.

2 Logarithmic Keying of Star Networks

Consider a *star* network where a process p needs to communicate securely with each of its d neighboring processes, $q.0, q.1, \cdots, q.(d-1)$. This requirement can be easily fulfilled by assigning d symmetric keys $s.0, s.1, \cdots, s.(d-1)$ to the network processes as follows. Each symmetric key $s.i$ is assigned only to the two processes p and $q.i$. Thus, the messages exchanged between p and $q.i$ can be encrypted using the symmetric key $s.i$ before they are sent, and they can be decrypted using $s.i$ after they are received.

This straightforward assignment of symmetric keys to processes requires that process p stores d symmetric keys, namely $s.0, s.1, \cdots, s.(d-1)$, and each other process $q.i$ stores one symmetric key, namely $s.i$. Next, we describe a more balanced assignment of symmetric keys to the processes in this star network. According to this assignment, process p needs to store only $(2 * \log d)$ symmetric keys, and each other process $q.i$ needs to store $(\log d)$ symmetric keys. We refer to this scheme of assigning symmetric keys to the processes in a star network as a *logarithmic keying* of the star network. (Throughout this paper, we adopt the convention that $\log x$ denotes the smallest integer whose value is at least $\log x$.)

The main idea of our logarithmic keying scheme is as follows. First, process p is assigned a set S of $(2 * \log d)$ symmetric keys. Second, each process $q.i$ is assigned a distinct subset $B.i$, that has $(\log d)$ symmetric keys, of set S. Later, if process p needs to send a secure message to process $q.i$, then p applies the bit-wise, exclusive-or operator to the keys in subset $B.i$ in order to compute a single symmetric key that is used to encrypt the message before p sends it to $q.i$. When process $q.i$ receives the encrypted message from process p, then $q.i$ applies the bit-wise, exclusive-or operator to the keys in subset $B.i$ in order to compute a single symmetric key that is used to decrypt the message after $q.i$ receives it from p. Similar procedure can be used to send an encrypted message from any process $q.i$ to process p.

The $(2 * \log d)$ symmetric keys in set S, assigned to process p, are named:

$$\begin{array}{ll} s.(0,0), & s.(0,1), \\ s.(1,0), & s.(1,1), \\ \cdots & \cdots \\ s.(\log d - 1, 0), & s.(\log d - 1, 1) \end{array}$$

In other words, these symmetric keys can be viewed as forming a two-dimensional matrix that has $(\log d)$ rows and two columns. We refer to this matrix as the S-matrix.

Next, we describe how to compute from set S a distinct subset $B.i$ of $(\log d)$ keys to be assigned to process $q.i$. Subset $B.i$ has exactly one key from each row in the S-matrix. Which of the two keys in the j-th row of the S-matrix is in subset $B.i$ depends on the j-th bit, $b.j$, in the bit representation of index i of process $q.i$ as follows.

```
**if**   $b.j = 0$
**then** key $s.(j, 0)$ is in $B.i$
**else** key $s.(j, 1)$ is in $B.i$
```

Therefore, each process $q.i$ is assigned a subset $B.i$ that is defined as follows:

$$B.i \;=\; \{\; s.(j, b.j) \;\mid\; 0 \le j < \log d \;\}$$

where $b.0, b.1, \cdots, b.(\log d - 1)$ is the bit representation of index i.

As an example, we describe a logarithmic keying of a star network that has five processes $p, q.0, q.1, q.2, q.3$. In this case, $d = 4$ and the logarithmic keying assigns $(2 * \log 4) = 4$ symmetric keys to process p. These four keys are named as follows.

$$\begin{array}{ll} s.(0,0), & s.(0,1), \\ s.(1,0), & s.(1,1) \end{array}$$

The index, 0, of process $p.0$ can be represented by the two bits $b.0 = 0$ and $b.1 = 0$. Thus, $q.0$ is assigned the two keys $s.(0, 0)$ and $s.(1, 0)$. The index of process $q.1$ can be represented by the two bits $b.0 = 1$ and $b.1 = 0$. Thus, $q.1$ is assigned the two keys $s.(0, 1)$ and $s.(1, 0)$. The index of process $q.2$ can be represented by the two bits $b.0 = 0$ and $b.1 = 1$, and so $q.2$ is assigned the two keys $s.(0, 0)$ and $s.(1, 1)$. Finally, the index of process $q.3$ can be represented by the two bits $b.0 = 1$ and $b.1 = 1$, and so $q.3$ is assigned the two keys $s.(0, 1)$ and $s.(1, 1)$. Note that no two of the four processes $q.0$ through $q.3$ are assigned the same subset of symmetric keys.

If each process $q.i$ uses the symmetric keys in its subset $B.i$ merely to encrypt messages before sending them to p and to decrypt messages after receiving them from p, then $q.i$ does not need to keep the keys in $B.i$ as separate keys. Instead, process $q.i$

can apply the bit-wise exclusive-or operator to the keys in $B.i$ and end up with a single key. Process $q.i$ needs to store only this one key (instead of storing the log d keys in subset $B.i$) and uses it to encrypt messages before sending them to p and to decrypt messages after receiving them from p. However, as discussed in the next section, there are other uses for the keys in subset $B.i$ that require these keys to remain separate and not be combined into a single key. Henceforth, we assume that the keys in each subset are stored as separate keys.

We end this section by showing that the logarithmic keying of a star network (described above) is asymptotically optimal. Assume that there is another keying scheme of the star network where process p is assigned a set T that has $|T|$ symmetric keys. To achieve security, it is necessary (but not sufficient) that process p shares with each process $q.i$ a distinct nonempty subset of set T. Because set T has $2^{|T|} - 1$ distinct nonempty subsets, and there are d of the $q.i$ processes, we have

$$|T| \;\; \geq \;\; \log(d + 1)$$

This implies that $|T|$ is of $O(\log d)$ which is the same size as that of set S in our logarithmic keying scheme.

3 Authenticated Broadcast in Star Networks

Consider the star network described in the previous section, and assume that symmetric keys are assigned to the processes in this network according to the logarithmic keying scheme discussed in the previous section. Thus process p is assigned $(2 * \log d)$ symmetric keys named

$$\begin{array}{ll} s.(0,0), & s.(0,1), \\ s.(1,0), & s.(1,1), \\ \cdots & \cdots \\ s.(\log d - 1, 0), & s.(\log d - 1, 1) \end{array}$$

Also each process $q.i$ is assigned the $(\log d)$ symmetric keys $s.(0, b.0), \cdots, s.(logd - 1, b.(logd - 1))$ where the bit string $b.0, b.1, \cdots, b.(logd - 1)$ is the bit representation of index i of process $q.i$.

Now assume that process p needs to broadcast a message m to all the processes $q.0, q.1, ..., q.(d - 1)$, and it needs to attach to message m an "authentication code" so that when a process $q.i$ receives the message, process $q.i$ can verify that only process p could have sent this message, and accept the message. But how to design this authentication code?

Thanks to the logarithmic keying scheme that we adopted for this star network, the authentication code for any broadcast message m can have a logarithmic length. Specifically, the authentication code for message m consists of the following $(2 * \log d)$ digests of m:

$$\begin{array}{ll} md.(0,0), & md.(0,1), \\ md.(1,0), & md.(1,1), \\ \cdots & \cdots \\ md.(\log d - 1, 0), & md.(\log d - 1, 1) \end{array}$$

Each digest $md.(x, y)$ is defined as $MD.(m|s.(x, y))$, where MD is a well known digest function, "|" is the concatenation operation, and $s.(x, y)$ is one of the symmetric keys assigned to process p by the logarithmic keying scheme.

Therefore, the format of the message that process p ends up broadcasting to each of the processes $q.0, q.1, ..., q.(d-1)$ is as follows.

$$(m, md.(0, 0), md.(0, 1), ..., md.(logd - 1, 1))$$

In other words, the broadcasted message consists of message m followed by $(2 * \log d)$ digests of m.

When a process $q.i$ receives a copy of the broadcasted message, $q.i$ computes $(\log d)$ digests of m using the symmetric keys in subset $B.i$. (Each digest $md.(x, y)$ is computed as $MD.(m|s.(x, y))$, where $s.(x, y)$ is a symmetric key in subset $B.i$.) If process $q.i$ detects that every one of its computed digests is present in the received message, $q.i$ concludes that the received message was sent by p and accepts m. Otherwise, $q.i$ concludes that the message was not sent by p and rejects it.

So far, we have presented a logarithmic keying scheme of star networks, and discussed how to take advantage of this scheme to encrypt and decrypt unicast messages, and to authenticate broadcast messages in any star network. In the next section, we extend this logarithmic keying scheme to a richer class of networks, called acyclic networks.

4 Logarithmic Keying of Acyclic Networks

The *topology* of a network is a connected undirected graph, where each node $p.j$ corresponds to a distinct process, also called $p.j$, in the network, and where each (undirected) edge connecting nodes $p.j$ and $p.k$ corresponds to a two-way channel that can be used in exchanging messages between the two corresponding processes $p.j$ and $p.k$ in the network.

(It follows from this definition that if a network topology has no edge between two nodes $p.j$ and $p.k$, then the two corresponding processes $p.j$ and $p.k$ cannot directly exchange messages in the network.)

A network is called a *star* iff the network topology consists of one center node and several peripheral nodes, and each peripheral node is connected only to the center node (by an edge).

A network is called *acyclic* iff the network topology is an acyclic undirected graph. Thus each star network is also acyclic, but not vice versa. In this section, we extend our logarithmic keying scheme for star networks to acyclic networks.

Consider an acyclic network that has n processes:

$$p.0, p.1, ..., p.(n-1)$$

Assume that the degree of this network is d. Therefore, we can use a straightforward edge coloring algorithm to assign an index in the range $0..d-1$ to each (two-way) channel in the network such that the indices of any two channels incident at the same process are distinct.

Each process $p.j$ in this network is assigned $(2 * \log d)$ symmetric keys named

$$\begin{array}{ll} s.j.(0,0), & s.j.(0,1), \\ s.j.(1,0), & s.j.(1,1), \\ \ldots & \ldots \\ s.j.(\log d - 1, 0), & s.j.(\log d - 1, 1) \end{array}$$

Before we can describe how to compute the symmetric keys assigned to each process, we need first to describe how can a process use its assigned keys to encrypt and decrypt messages that this process exchanges with its neighboring processes.

Assume that a process $p.j$ needs to securely send a message m to a neighboring process $p.k$ via a channel whose index has the bit representation $b.0, b.1, ..., b.(\log d - 1)$. In this case, $p.j$ applies the bit-wise exclusive-or operator to the symmetric keys

$$s.j.(0, b.0), s.j.(1, b.1), ..., s.j.(\log d - 1, b.(\log d - 1))$$

and ends up with a single key that $p.j$ uses to encrypt each message m before sending it to $p.k$ via the channel. When process $p.k$ receives the encrypted message via the channel whose binary representation is $b.0, b.1, ..., b.(\log d - 1)$, then $p.k$ applies the bit-wise exclusive-or operator to the symmetric keys

$$s.k.(0, b.0), s.k.(1, b.1), ..., s.k.(\log d - 1, b.(\log d - 1))$$

and ends up with a single key that $p.k$ uses to decrypt the received message and obtain the original message m.

Clearly, the symmetric key that $p.j$ used to encrypt message m needs to be identical to the symmetric key that $p.k$ used to decrypt the received message. This can be achieved by requiring that the following $\log d$ equalities hold

$$\begin{array}{lll} s.j.(0, b.0) & = & s.k.(0, b.0), \\ s.j.(1, b.1) & = & s.k.(1, b.1), \\ \ldots & & \\ s.j.(\log d - 1, b.(\log d - 1)) & = & s.k.(\log d - 1, b.(\log d - 1)) \end{array}$$

These $\log d$ equalities can be written more succinctly as the following condition.

For every i in the range $0..(\log d - 1)$, $s.j.(i, b.i) = s.k.(i, b.i)$

We refer to this condition as the *key consistency condition*.

The key consistency condition states that half the keys in a process $p.j$ are equal to the corresponding keys in a process $p.k$, provided that $p.j$ and $p.k$ are neighbors, i.e., they are connected by a two-way channel. Hence, in computing the symmetric keys in each process in an acyclic network, one needs to ensure that the keys in each pair of neighboring processes satisfy the key consistency condition.

An algorithm for computing the $(2*\log d)$ keys in each process in an acyclic network consists of the following two steps.

```
Step 0: choose any process $p.j$ in the network and randomly selects its $(2 * \log d)$ keys:
        $s.j.(0,0), \cdots, s.j.(\log d - 1, 1)$

Step 1: while the network has two neighboring processes $p.j$ and $p.k$ such that
            a. the secrets in $p.j$ are already computed,
            b. the secrets in $p.k$ are not yet computed, and
            c. the connecting channel between $p.j$ and $p.k$ has an index whose bit
               representation is $b.0, \cdots, b.(\log d - 1)$
        do
        for each $i$ in the range $0..(\log d - 1)$, compute the $i$-th secrets in $p.k$ as follows
        $s.k.(i, b.i)$       $:= s.j.(i, b.i)$
        $s.k.(i, 1 - b.i)$   $:=$ any random value
        od
```

Note that this algorithm is written under the reasonable assumption that the network is connected. It is straightforward to extend this algorithm to the general case where the network is partitioned into two or more components.

The $(2 * \log d)$ keys assigned to each process $p.j$ in an acyclic network can also be used by $p.j$ to compute the authentication code for any message m that $p.j$ needs to broadcast to all its neighboring processes. Specifically, the authentication code for message m consists of $(2 * \log d)$ digests, and each digest is of the form $MD.(m|s.j.(x, y))$ where MD is the message digest function, "|" is the concatenation operation, and $s.j.(x, y)$ is one of the symmetric keys assigned to process $p.j$ by logarithmic keying.

When a neighboring process $p.k$ receives a copy of the broadcast message (along with its authentication code) via a channel whose index has the binary representation $b.0, ..., b.(\log d - 1)$, then $p.k$ computes, for every i in the range $0..(\log d - 1)$, the message digest $MD.(m|s.k.(i, b.i))$ and checks whether this message digest is part of the authentication code of the received message. If every computed message digest is part of the authentication code of the received message m, then $p.k$ concludes correctly that message m is sent by $p.j$ and accepts m. Otherwise, $p.k$ rejects message m.

5 Logarithmic Keying of Limited-Cycle Networks

In this section and the next, we describe two methods for extending our logarithmic keying scheme for acyclic networks to networks with cycles. These two methods are called superimposition and decomposition.

In the *superimposition method*, we start with an acyclic network. We then observe that some of the keys that are assigned to the network processes using our logarithmic keying are *spare*, i. e. they are not used in encrypting or decrypting any message that is exchanged over any edge in the acyclic network. Thus, we superimpose new edges on the acyclic network to add cycles to it, and use the spare keys to encrypt and decrypt the messages that are exchanged over the superimposed edges.

In the *decomposition method*, we start with a network with cycles. We then partition this network into a small number of edge-disjoint acyclic subnetworks. Then, we use our logarithmic keying scheme, described in the previous section, to assign symmetric keys to each process in each acyclic subnetwork. The net effect is that each process is assigned $O(\log d)$ symmetric keys, where d is the degree of the original network with cycles. Thus, the resulting keying scheme is logarithmic.

In the remainder of this section, we show that the superimposition method can be used in the logarithmic keying of a special class of communication networks, called limited-cycle networks. (In the next section, we show that the decomposition method can be used in the logarithmic keying of a special class of networks, called planar networks.)

Consider an acyclic network whose degree is d, and without loss of generality, assume that d is at least 2. This network has at least two processes $p.j$ and $p.k$ such that the following two conditions hold. (For example, these two conditions hold for any two leaf processes in the network.)

1. Process $p.j$ has $a.j$ incident edges and $(\log d - \log a.j)$ is at least one.

2. Process $p.k$ has $a.k$ incident edges and $(\log d - \log a.k)$ is at least one.

As the network is acyclic, the network processes are assigned symmetric keys according to the logarithmic keying scheme described in the previous section. From Condition 1, at least one of the keys assigned to process $p.j$ is spare, i.e. this key is not used to encrypt or decrypt any message sent or received by process $p.j$ over any of its incident edges. Similarly, from Condition 2, at least one of the keys assigned to process $p.k$ is spare.

Let $s.j.(x.j, y.j)$ be a spare key assigned to $p.j$, and let $s.k.(x.k, y.k)$ be a spare key assigned to $p.k$. Because these two keys are spare, they are selected at random by the two-step algorithm in the previous section. Now, assume that these two keys are selected to be identical. In this case, a new edge can be superimposed between the two processes $p.j$ and $p.k$ in the acyclic network causing the network to have a cycle. For convenience, we refer to this superimposed edge as a *c-edge* in order to distinguish it from the edges in the original acyclic network, which we call *a-edges*.

As mentioned above, each a-edge has an index in the range $0..d-1$. Now, we adopt the convention that the superimposed c-edge has two indices: one index $(x.j, y.j)$ is known only to process $p.j$, and the other index $(x.k, y.k)$ is known only to process $p.k$.

When process $p.j$ needs to send a message over the c-edge $(x.j, y.j)$ to process $p.k$, $p.j$ encrypts the message using its symmetric key $s.j.(x.j, y.j)$ before sending the message over the c-edge. When process $p.k$ receives the encrypted message over the c-edge $(x.k, y.k)$ from process $p.j$, $p.k$ decrypts the message the message using its symmetric key $s.k.(x.k, y.k)$ after receiving the message over the c-edge.

When process $p.j$ needs to broadcast a message m to all its neighbors, $p.j$ computes the authentication code of m using all the symmetric keys assigned to $p.j$, as described in the previous section. Then $p.j$ sends a copy of the message

(m, authentication code of m)

over every edge incident at $p.j$, including the c-edge $(x.j, y.j)$. When process $p.k$ receives the broadcasted message over the c-edge $(x.k, y.k)$, $p.k$ computes the message digest $MD.(m|s.k.(x.k, y.k))$ and checks whether this digest is part of the authentication code in the received message. If so, $p.k$ accepts m. Otherwise, $p.k$ discards m.

So far, we discussed how to superimpose one (the first) c-edge on the original acyclic network to create one cycle in the network. In fact, many c-edges can be superimposed, sequentially one after the other, in order to create many cycles in the network. The only requirement needed to superimpose one more c-edge between two nodes in the network is that each of the two nodes satisfies the following condition.

$(\log d - \log a - c)$ is at least one

where d is the network degree, a is the numbers of a-edges that are currently incident at the node, and c is the number of c-edges that are currently incident at the node.

We are now ready to define the class of limited-cycle networks that can be logarithmically keyed using the above superimposition method. A *limited-cycle network* is one where each edge in its topology graph G can be classified as either an a-edge or c-edge such that the following two conditions hold.

1. The subgraph of G that consists of a-edges only is acyclic.

2. For each process p in G, $(\log d - \log a - c)$ is at least zero, where
   d is the network degree,
   a is the number of a-edges incident at p, and
   c is the number of c-edges incident at p.

6 Logarithmic Keying of Planar Networks

In this section, we utilize a decomposition method to extend our logarithmic keying scheme for acyclic networks to a scheme for planar networks, where a *planar network* is one whose topology is a planar graph.

It is well known, e.g. [4] and [6], that any planar graph G can be decomposed into at most three acyclic subgraphs, called *factors*, such that the following two conditions hold. First, every factor has the same nodes as the original graph G. Second, each edge in the original graph G appears in exactly one factor. It follows that the degree of each factor is at most the degree of the original graph G.

The decomposition method works as follows. Given a planar network G, whose degree is d, the keying scheme proceeds by decomposing the edges of G into k factors, where $0 \leq k \leq 3$. Each edge in G is then given an index (r, i), where r is an index of the factor that contains the edge, $0 \leq r < k$, and i is the index of the edge in factor r, $0 \leq i < d$. (Recall that any two edges that are incident to the same node in a factor are assigned distinct indices.) Hence, in the network G, if a node has two incident edges labeled (r, i) and (r', i), then these two edges must belong to two different factors (i.e., $r \neq r'$). The logarithmic keying scheme for acyclic graphs mentioned above is then applied independently to each of the k factors. As a result, each node is assigned k sets of keys, where each set has at most $(2 * \log d)$ keys. Since computing the key of any given edge in G can be deduced from the index of that edge, we conclude that $O(\log d)$ keys per process are sufficient for keying any planar network.

We note that this decomposition method can be equally applied to other classes of networks. For example, any graph with *treewidth* $\leq k$, for constant $k > 0$, can be decomposed into k acyclic factors, as discussed in [5] and [6]. Therefore, using the decomposition method, any bounded treewidth graph can be logarithmically keyed.

7 Concluding Remarks

In this paper, we described logarithmic keying schemes for assigning symmetric keys to the different processes in several classes of communication networks, which include acyclic networks, limited-cycle networks, and planar networks. We also described two methods, namely superimposition and decomposition, for extending logarithmic keying schemes of acyclic networks to networks with cycles.

Two open problems are suggested by the investigation described in this paper. The first problem is to design a logarithmic keying scheme that can be used in any fully connected communication network. The second problem is to design a logarithmic keying scheme that can be used in any communication network, regardless of the network topology. Note that the second open problem is a generalization of the first problem. But we believe that solving the first problem first will make the second problem easier to tackle.

Acknowledgment

The work of M. G. Gouda is supported in part by the National Science Foundation under Grant No. 0520250. The work of S. Kulkarni is supported in part by the National Science Foundation under Grant No. CCR-0092724. The work of E. S. Elmallah is supported by NSERC Canada.

References

[1] L. Gong and J. Wheeler. A Matrix Key-Distribution Scheme. *Journal of Cryptology: The Journal of the International Association for Cryptologic Research*. Vol. 2, No. 1, pp. 51-59, 1990.

[2] S. S. Kulkarni, M. G. Gouda, and A. Arora. *Computer Communications*. Vol. 29, pp. 200-215, 2006.

[3] A. S. Aiyer, L. Alvisi, and M. G. Gouda. Key Grids: A protocol Family for Assigning Symmetric Keys. Proceedings of the IEEE International Conference on Network Protocols (ICNP-06), 2006.

[4] B. Bollobás. *Modern Graph Theory*. Springer-Verlag, 1998.

[5] A. Brandstädt, V. B. Le, and J. Spinrad. *Graph Classes: A Survey*. SIAM Monographs on Discrete Mathematics and Applications, 1999.

[6] C. J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, 1987.