Logarithmic Keying

EHAB S. ELMALLAH
University of Alberta
MOHAMED G. GOUDA
University of Texas at Austin
and
SANDEEP S. KULKARNI
Michigan State University

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, planar networks, and dense bipartite networks. In addition, we present a construction that utilizes efficient keying schemes for general bipartite networks to construct efficient keying schemes for general networks.

Categories and Subject Descriptors: K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection; C.2.0 [**COMPUTER-COMMUNICATION NETWORKS**]: Security and Protection; C.2.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Protocols

General Terms: Algorithms, Security

Additional Key Words and Phrases: Secure communications, symmetric keys, keying scheme

A preliminary version of this paper appeared in the *"Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems"*, November 17th-19th, 2006, Dallas.
The work of E. S. Elmallah is supported by NSERC Canada under grant number OGP 36899. 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.
Author addresses: E. S. Elmallah, Department of Computing Science, University of Alberta, Edmonton, T6G 2E1, Canada; email: ehab@cs.ualberta.ca. M. G. Gouda, Department of Computer Sciences, University of Texas at Austin, 1 University Station (C0500) Austin, TX, 78712-0233; email: gouda@cs.utexas.edu. S. Kulkarni, Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824; email: sandeep@cse.msu.edu

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.

In 1985, Rolf Blom has published a pioneering paper [Blom 1985] where he showed that if the communication network is fully connected (consisting of $d+1$ processes), then each process in the network needs to store only $O(\log d)$ bits in order to guarantee the confidentiality of its d communications with every other process in the network. Blom's scheme takes as a design parameter the minimum number of colluding processes that is needed to compromise other keys.

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. Since Blom's paper, two research directions have emerged to explore efficient keying schemes of fully connected communication networks.

(1) **The Algebraic Direction:** In this research direction, it is assumed that each process in a communication network is assigned private pieces of information (of arbitrary types). When a process p in the network needs to communicate with another process q, process p executes a sequence of algebraic operations on its assigned pieces of information and on the identity of process q and computes a symmetric key that p can use to communicate securely with q. Similarly, process q executes a similar sequence of algebraic operations on its assigned pieces of information and on the identity of process p and computes the same symmetric key that is computed by process p. The advantage of this research direction is that the storage needed to store the assigned pieces of information to each process in the communication network can be relatively small. The disadvantage of this research direction is that the time needed to compute the symmetric keys can be relatively large.

(2) **The Shared Key Direction:** In this research direction, it is assumed that each process in a communication network is assigned a distinct set of symmetric keys. When a process p in a network wants to communicate with another

process q, process p identifies the subset of keys that are assigned to both p and q and applies the bit-wise exclusive-or operator to these common keys to compute one symmetric key that process p can use to communicate securely with q. Similarly, process q identifies the subset of keys that are assigned to both p and q and applies the bit-wise exclusive-or operator to these common keys to compute the same symmetric key that is computed by p. The disadvantage of this research direction is that the storage needed to store the assigned keys to each process can be relatively large. The advantage of this research direction is that the time to compute the shared keys can be relatively small.

The algebraic direction has produced many efficient keying schemes and interesting lower and upper bounds; see, for example, [Blundolz et al. 1993; Beimel and Chor 1996; Stinson 1997].

By contrast, the shared key direction is relatively new and has produced fewer results. In [Gong and Wheeler 1990], the authors describe a keying scheme where $O(\sqrt{d})$ symmetric keys are assigned to each process in a fully connected communication network. In [Kulkarni et al. 2006], the authors describe a variation of the scheme in [Gong and Wheeler 1990], and showed that this scheme is optimal if each pair of distinct processes share no more than two symmetric keys. In [Aiyer et al. 2006], the authors describe 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 to communication networks that are fully connected. So far, there are no corresponding results to arbitrary communication networks; hence, this paper.

In this paper, we describe logarithmic keying schemes, that belong to the shared key direction, for assigning symmetric keys to the processes in rich classes of communication networks. These classes include star networks, acyclic networks, cycle-limited networks, planar networks, and dense bipartite networks. We also describe an efficient keying scheme for arbitrary networks. In this scheme, each process in the network is assigned $O((\log d) * (\log n))$ symmetric keys, where d is the network degree, and n is the number of processes in the network. Thus, for a fully connected network consisting of $d + 1$ processes, the above $O(\log^2 d)$ keying result of [Aiyer et al. 2006] becomes a special case of our $O((\log d) * (\log n))$ result.

Note that logarithmic keying schemes, as those described in this paper, are especially useful for communication networks where each process has limited processing power and limited memory for storing its assigned symmetric keys. Examples of such networks include sensor networks [Eschenauer and Gligor 2002; Perrig et al. 2002], ad-hoc networks [Hubaux et al. 2001; Yang et al. 2004], and mobile networks [Tatebayashi et al. 1990; Mu and Varadharajan 1996].

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:

$$B.i \quad = \quad \{ \quad s.(j, b.j) \quad | \quad 0 \leq j < \log d \quad \}$$

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

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

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 $s.(0, b.0), \cdots, s.(\log d - 1, b.(\log d - 1))$ where the bit string $b.0, b.1, \cdots, b.(\log d - 1)$ 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:

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

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.

$$(m, md.(0,0), md.(0,1), ..., md.(\log d - 1, 1))$$

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.

As an application of logarithmic keying of star networks, consider a secure sensor network that consists of one base station and d sensors. The wireless communication between the base station and the sensors proceeds in steps, where each step consists of two parts. First, the base station broadcasts a "command" to all the sensors in the network. Second, each sensor receives the broadcasted command, performs some sensing task, then sends the results of the sensing task back to the base station.

In this network, the broadcasted command needs to be authenticated so that each sensor can be certain that the base station is the one that issued the command. Also the sensing results, sent from a sensor to the base station, need to be encrypted so that no adversary can obtain these results. These security measures can be achieved, using our logarithmic keying scheme of star networks in Section 2, by assigning $(2 * \log d)$ symmetric keys to the base station, and assigning $(\log d)$ symmetric keys to each sensor in the network. Also, each broadcasted command needs to have $(2 * \log d)$ digests. For example, if the network has a large number of sensors, say $d = 1024$. Then the base station needs to be assigned a mere 20 symmetric keys, and each sensor needs to be assigned a mere 10 symmetric keys. Also, each broadcasted command needs to have 20 digests. (Had we used the straightforward linear keying scheme, each sensor would have needed to store only one symmetric key instead of 10, but then each broadcasted command would have ended up with

a whopping 1024 digests.)

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 $p.j$, 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 $p.k$.

(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

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

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

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

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

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

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

$$(m, \text{ authentication code of } m)$$

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.

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

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 that any planar graph G can be decomposed into at most three acyclic subgraphs (see, e.g., [Colbourn 1987]), 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 network G, if a node has two incident edges labeled (r, i) and (r', i) with identical value i in the second component of each label, 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 (for a definition of treewidth see, for example, [Downey and Fellows 1999]). Therefore, using the decomposition method, any bounded treewidth graph can be logarithmically keyed.

7. LOGARITHMIC KEYING OF DENSE BIPARTITE NETWORKS

In this section, we present a keying scheme that provides secure communications between a set $X = \{0, 1, \cdots, nx - 1\}$ of nx client processes, and a set $Y = \{0, 1, \cdots, ny - 1\}$ of ny server processes. Assuming no two clients (or servers) require secure communications, the topology of such network is a bipartite graph $G = (X \bigcup Y, E)$, where each edge $\{u, v\}$ of E corresponds to a communication channel that connects a process u in X to a process v in Y. Each process in such network has at most $dmax = \max(nx, ny)$ neighboring processes, and the network degree d is at most $dmax$. We call such a network *dense* if $d \approx dmax$. Our main result in this section is to present a keying scheme for bipartite networks that assigns at most $(3 * \log dmax)$ keys to each process in the network. Thus, the presented scheme is logarithmic for dense bipartite networks.

Throughout this section, we denote the ith bit (of weight 2^i) in the binary representation of an integer w by $[w]_i$. Our logarithmic keying scheme can then be presented as follows.

(1) Specify two matrices of random symmetric keys. The first matrix $KX[0..(\log nx) - 1, 0..1]$ has $(\log nx)$ rows and two columns. The second matrix $KY[0..(\log ny) -$

$1, 0..1]$ has $(\log ny)$ rows and two columns. When no confusion arises, we also use KX and KY to denote the set of keys in each matrix, respectively.
Example. The network in Figure 1 has $nx = 4$, and $ny = 8$. The corresponding matrices are written as

$$KX = \begin{bmatrix} KX[0,0] & KX[0,1] \\ KX[1,0] & KX[1,1] \end{bmatrix}, \text{and } KY = \begin{bmatrix} KY[0,0] & KY[0,1] \\ KY[1,0] & KY[1,1] \\ KY[2,0] & KY[2,1] \end{bmatrix}.$$

(2) Assign to each process u of X a subset SX_u of the keys in $KX \bigcup KY$. We view SX_u as an incomplete matrix of $nx + ny$ rows and two columns. Whether both keys of row i of matrix KY, or exactly one of them is assigned to SX_u depends on bits $\{[v]_i |\ v \text{ is a neighbor of } u\}$. Namely, key $KY[i, j]$ is assigned to SX_u if for some neighbor v of u we have $[v]_i = j$.
Example. In Figure 1, process 1 in X has neighbors $\{0, 2, 4\}$ in Y. Applying the above rule to the two neighbors 0 and 2, for example, gives the following key assignments:
—Since $[0]_{i=0} = 0$ then $KY[0, 0]$ is in SX_1.
—Since $[0]_{i=1} = 0$ then $KY[1, 0]$ is in SX_1.
—Since $[0]_{i=2} = 0$ then $KY[2, 0]$ is in SX_1.
—Since $[2]_{i=0} = 0$ then $KY[0, 0]$ is in SX_1.
—Since $[2]_{i=1} = 1$ then $KY[1, 1]$ is in SX_1.
—Since $[2]_{i=2} = 0$ then $KY[2, 0]$ is in SX_1.

Additionally, matrix SX_u is assigned exactly one key from each row of matrix KX. Which of the two keys from row i of matrix KX is assigned to SX_u depends on the bit $[x]_i$. That is, key $KX[i, j]$ is assigned to SX_u if $[x]_i = j$.
Example. In Figure 1, matrix SX_1 of process 1 in X is shown in the figure.

(3) We also assign to each process v of Y a subset SY_v of the keys in $KX \bigcup KY$. The rules used here are symmetric with the rules in (2), obtained by exchanging the roles of X and Y. So, SY_v is assigned exactly one key from each row of matrix KY, and at most two keys from each row of matrix KX.
Example. Figure 1 shows matrix SY_4 of process 4 in Y.

(4) Next, assign to each channel $\{u, v\}$, $u \in X$ and $v \in Y$, a label constructed by concatenating the binary representation of u followed by the binary representation of v, so that the u part becomes the most significant part of the label. Notice that each label has $\log nx + \log ny$ bits.
Example. In Figure 1, the channel between process 1 in X, and process 4 in Y is labeled 01 100.

When process u needs to send a secure message to process v then u applies the bit-wise exclusive-or operator to the keys corresponding to the label assigned to channel $\{u, v\}$ to compute a single symmetric key. This scheme ensures that all keys required for this step are stored in both SX_u, and SY_v.
Example. In Figure 1, when process 1 in X wants to communicate securely with process 4 in Y then, knowing the channel label 01 100, process 1 selects the keys $KY[0, 0]$, $KY[1, 0]$, $KY[2, 1]$, $KX[0, 1]$, and $KX[1, 0]$, and apply the bit-

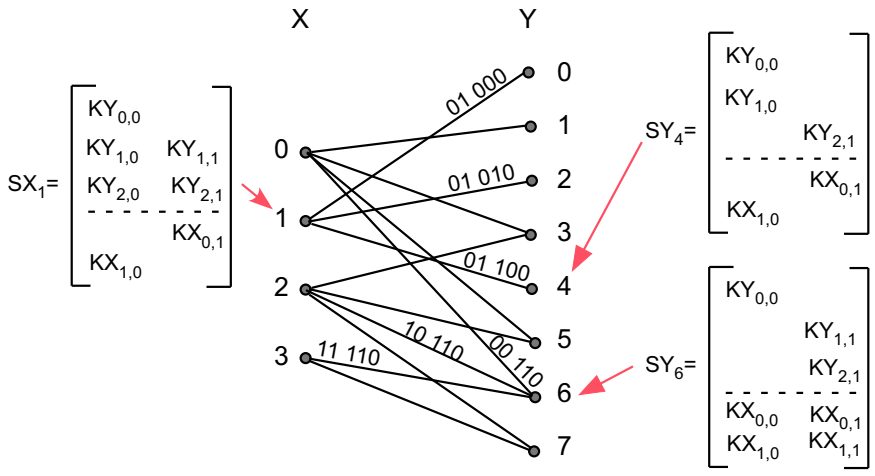

Fig. 1. Keying of a bipartite network

wise exclusive-or operator to compute a single key that process 1 can then use to communicate with process 4.

To see that this keying scheme is secure, let $\{u, v\}$ be any channel and consider any other process w in G. If w is in X then at least one key of the matrix KX is in SX_u but not in SX_w. This is true since the binary representation of u differs from that of w in at least one bit. In addition, computing the shared $\{u, v\}$ key requires all of the $\log nx$ keys in SX_u. Likewise, if w is in Y then at least one key of the matrix KY is in SY_v but not in SY_w, and computing the shared $\{u, v\}$ key requires all the $\log ny$ keys in SY_v.

The above keying scheme assigns at most $(\log nx + 2 * \log ny)$ keys to each process in X, and by symmetry, at most $(2 * \log nx + \log ny)$ keys to each process in Y. Hence, the devised scheme assigns at most $(3 * \log dmax)$ keys to each process, as mentioned earlier. This result implies that any general bipartite network admits an $O(\log n)$ keying scheme, where n is the number of processes in the network.

8. LOGARITHMIC KEYING OF GENERAL BIPARTITE NETWORKS

In this section we describe a "construction" of an efficient keying scheme for general networks using an efficient keying scheme for general bipartite networks. We utilize this construction in two ways. First, we show that utilizing a logarithmic keying scheme for general bipartite networks, if such scheme exists, our construction yields an $O(\log^2 d)$ keying scheme for a general network, where d is the network degree. Second, we show that utilizing the $O(\log n)$ keying scheme for general bipartite networks, discussed in Section 7, our construction yields an $O((\log d) * (\log n))$ keying scheme for general networks, or yields an $O(\log^2 n)$ for fully connected networks.

Our construction has two main components: a decomposition procedure, and a keying scheme with an associated protocol, as described below.

The Decomposition Procedure. Given an arbitrary network $G = (V, E)$, the following procedure partitions the set E of channels into disjoint subsets, each

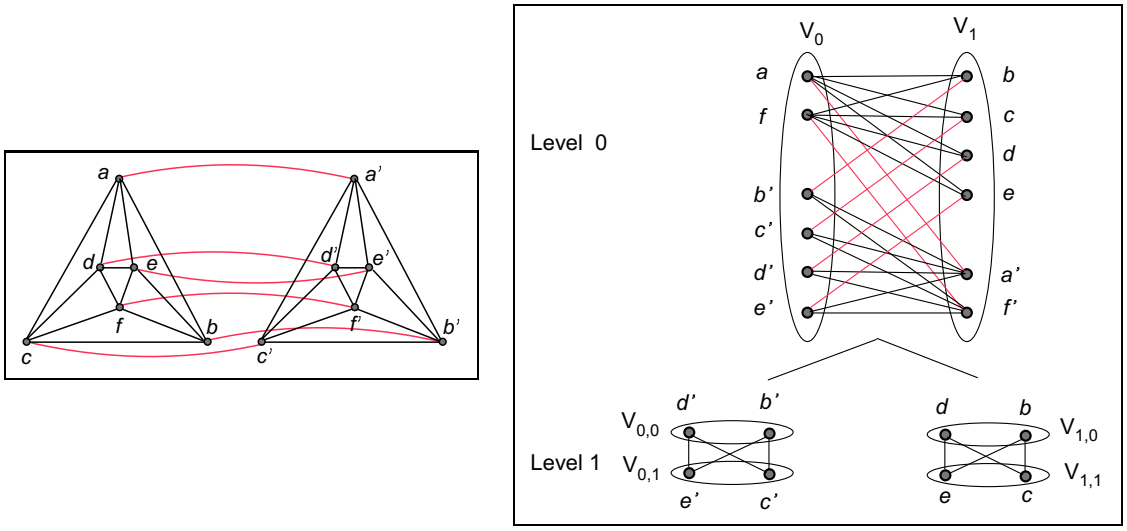

Fig. 2. A network with $d = 5$, and its recursive decomposition into bipartite networks.

subset forms a bipartite network. The procedure results in a decomposition tree T where each node corresponds to one of the computed bipartite networks (as shown in Figure 2).

—At level 0, the root node of T corresponds to a bipartite network obtained by splitting the processes in G into two subsets V_0 and V_1 such that each process of V_0 has at least half of its neighbors in V_1, and each process in V_1 has at least half of its neighbors in V_0. Such partitioning of processes is known to exist for any network using the following efficient algorithm: start by any random partitioning, and subsequently move a process from one partition to the other if it has more neighbors in its current partition. Each time a process switches sides, the bipartite network made of channels between V_0, and V_1 gains a new channel. The algorithm terminates eventually with the required configuration. The bipartite subnetwork at this level contains all channels connecting processes in V_0 to processes in V_1.

—At level 1 of the decomposition tree T, at most 2 residual networks are considered for further decomposition. Each residual network is formed by the set V_i ($i = 0$ or 1), of processes, and the channels not included in any of the previous levels. Note that, by the previous step, each residual network has degree $\leq d/2$. Again, if such residual network is not bipartite then we split each set V_i ($i = 0, 1$) into $V_{i,0}$, and $V_{i,1}$ so that each process in $V_{i,0}$ has at least half of its neighbors in $V_{i,1}$ (and vice versa). Each bipartite subnetwork at this level then contains all channels connecting processes in $V_{i,0}$ to processes in $V_{i,1}$, for $i = 0, 1$.

—At a general level $i \geq 1$ of T, at most 2^i residual networks are considered for further decomposition. Each residual network is formed by the processes in one partition of a bipartite subnetwork at the previous level, and the channels that are not included in any of the previous levels. By induction, each such residual network has degree $\leq d/2^i$.

If H is a subnetwork in T we denote its level by $\ell(H)$. The following properties of T can then be easily verified.

(i) T has at most $(\log d)$ levels.
(ii) If channel $\{x, y\}$ belongs to subnetwork H, then processes x and y don't appear together in any subnetwork H' where $\ell(H') > \ell(H)$.

Property (ii) above makes it possible for the keying scheme to avoid storing $\ell(H)$ in either x, or y. Rather, when x and y want to communicate securely, they invoke the search procedure outlined below to find $\ell(H)$.

The Keying Scheme. Our keying scheme, and the associated secure protocol work as follows.

(1) Each bipartite subnetwork H in T with nh processes is keyed as follows:
   (a) The processes of H are assigned local labels $\{0, 1, 2, \cdots, nh-1\}$.
   (b) Using the assigned local labels, H is keyed using a unique set S_H of keys.
   (c) If x is a process in H then x stores its local label at level $\ell(H)$, and the subset of the S_H keys assigned to it by the utilized scheme for keying bipartite networks.

   Note that each process belongs to at most one bipartite subnetwork at any given level of T. Hence, each process stores keys from at most $(\log d)$ distinct sets of keys. The stored sets in each process are indexed by their respective tree levels.

(2) Now, suppose that $\{x, y\}$ is a channel of G that belongs to subnetwork H in the decomposition tree. If x wants to send a secure message to y, then both x and y invoke the following search procedure to identify $\ell(H)$, and the assigned shared key. The search procedure works in rounds, and uses a count-down counter, denoted ℓ, whose value is synchronized among the two processes. Initially, ℓ is set to the highest level in T (so, $\ell \leq (\log d) - 1$), and subsequently ℓ is decremented at the start of each new round.

   Each round determines whether ℓ is the desired level $\ell(H)$. The basic operations performed by x and y in each round are:
   (a) Each process checks if it has been assigned a local label at the level ℓ under test. If not, the round is aborted.
   (b) x and y exchange their assigned local labels. Using such labels, each process checks that it has the necessary keys (from the set of assigned keys at level ℓ) to compute the shared key. If a process detects that is has insufficient subset of keys to construct the shared key, the round is aborted.
   (c) Each process verifies the computed key by encrypting a mutually agreed upon test message. The encrypted messages are exchanged, and each process attempts to decrypt the message. If any process fails to decrypt, and verify the received message, the round is aborted.

   If a round is aborted, a new round is started after decrementing ℓ. Otherwise, the round finishes successfully.

To see that a successful round occurs only when $\ell = \ell(H)$, and hence the two keys computed in a successful round by x and y are identical to the intended shared key, we note the following. If $\ell > \ell(H)$, in some round, then x and y use two disjoint sets of keys in the computations done in that particular round. This is true since for

each bipartite subnetwork in T, the scheme utilizes a unique set of keys to compute all shared keys in that particular subnetwork. Moreover, by property (ii) of the decomposition tree, processes x and y don't appear together in any subnetwork H' with level $\ell(H') > \ell(H)$, and the search considers the possible values of ℓ in a decreasing order.

We conclude by showing that the above construction achieves the results mentioned at the beginning of the section. Indeed, if we assume that a logarithmic keying scheme exists for general bipartite networks then utilizing such scheme within our construction results in each process storing a total of $O(\log d + \log \frac{d}{2} + \log \frac{d}{2^2} + \cdots + 1)$ keys. Thus, each process is assigned $O(\log^2 d)$ keys, as mentioned above. On the other hand, utilizing the $O(\log n)$ bipartite network keying scheme of section 7 results in each process storing a total of $O((\log d) * (\log n))$ keys, as claimed.

9. 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, planar networks, and dense bipartite networks. We also described two methods, namely superimposition and decomposition, for extending logarithmic keying schemes of acyclic networks to networks with cycles. Moreover, we presented a construction that utilizes an efficient keying scheme for general bipartite networks to construct an efficient keying scheme for arbitrary networks. We showed that combining such construction with the keying scheme described in section 7 yields an $O(\log^2 n)$ keying scheme for a fully connected network, where n is the number of processes in the network.

The logarithmic keying scheme described in section 2 is used in [Wang and Kulkarni 2007] for authentication in reprogramming sensor networks. Logarithmic keying, however, is vulnerable to collusion. In [Kulkarni and Bezawada 2006], for example, the authors show how a symmetric key instantiation protocol can provide a tradeoff between the number of secrets maintained by each user and the level of collusion resistance. Thus, logarithmic keying needs to be used in situations where collusion is unlikely to occur.

Two open problems, pertaining to the shared key research direction, 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. (Note that Blom [Blom 1985] has presented a solution, pertaining to the algebraic research direction, for the first problem.)

Acknowledgment

We are thankful to the editors, Drs. Ajoy Datta and Giovanna Di Marzo Serugendo, and to the anonymous reviewers for their helpful comments.

REFERENCES

Aiyer, A. S., Alvisi, L., and Gouda, M. G. 2006. Key grids: A protocol family for assigning symmetric keys. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP-06)*.

Beimel, A. and Chor, B. 1996. Communication in key distribution schemes. *IEEE Transactions on Information Theory 42,* 1 (January), 19–28.

Blom, R. 1985. An optimal class of symmetric key generation systems. *Lecture Notes in Computer Science: Advances in Cryptology - EUROCRYPT '84 LNCS 209*, 335–338.

Blundolz, C., Santis, A. D., Herzberg, A., Kutten, S., Vaccaro, U., and Yung, M. 1993. Perfectly-secure key distribution for dynamic conferences. *Lecture Notes in Computer Science: Advances in Cryptology - CRYPT0 '92 LNCS 740*, 471–486.

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

Downey, R. G. and Fellows, M. R. 1999. *Parameterized complexity*. Springer, New York.

Eschenauer, L. and Gligor, V. D. 2002. A key-management scheme for distributed sensor networks. In *CCS '02: Proceedings of the 9th ACM Conference on Computer and Communications Security*. ACM Press, New York, NY, USA, 41–47.

Gong, L. and Wheeler, J. 1990. A matrix key-distribution scheme. *Journal of Cryptology 2,* 1 (February), 51–59.

Hubaux, J.-P., Buttyán, L., and Čapkun, S. 2001. The quest for security in mobile ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing*. ACM Press, New York, NY, USA, 146–155.

Kulkarni, S. S. and Bezawada, B. 2006. A family of collusion resistant protocols for instantiating security. In *Proceedings of the IEEE International Conference on Network Protocols (ICNP-05)*. 279–288.

Kulkarni, S. S., Gouda, M. G., and Arora, A. 2006. Secret instantiation in ad-hoc networks. *Computer Communications 29*, 200–215.

Mu, Y. and Varadharajan, V. 1996. On the design of security protocols for mobile communications. *Lecture Notes in Computer Science: Information Security and Privacy - ACISP '96 LNCS 1172*, 134–145.

Perrig, A., Szewczyk, R., Tygar, J. D., Wen, V., and Culler, D. E. 2002. SPINS: security protocols for sensor networks. *Wireless Networks 8,* 5, 521–534.

Stinson, D. R. 1997. On some methods for unconditionally secure key distribution and broadcast encryption. *Designs, Codes and Cryptography 12*, 215–243.

Tatebayashi, M., Matsuzaki, N., and Newman, D. 1990. Key distribution protocol for digital mobile communication-systems. *Lecture Notes in Computer Science: Advances in Cryptology - CRYPTO '89 LNCS 435*, 324–333.

Wang, L. and Kulkarni, S. S. 2007. Authentication in reprogramming of sensor networks for mote class adversaries. In *15th International Workshop on Parallel and Distributed Real-Time Systems*.

Yang, H., Luo, H., Ye, F., Lu, S., and Zhang, L. 2004. Security in mobile ad hoc networks: challenges and solutions. *IEEE Wireless Communications 11*, 38–47.