Key Grids: A Protocol Family for Assigning Symmetric Keys

Amitanand S. Aiyer
University of Texas at Austin
anand@cs.utexas.edu

Lorenzo Alvisi
University of Texas at Austin
lorenzo@cs.utexas.edu

Mohamed G. Gouda
University of Texas at Austin
gouda@cs.utexas.edu

Abstract

We describe a family of $\log n$ protocols for assigning symmetric keys to n processes in a network so that each process can use its assigned keys to communicate securely with every other process. The k-th protocol in our protocol family, where $1 \leq k \leq \log n$, assigns $O(k^2 \sqrt[k]{n})$ symmetric keys to each process in the network. (Thus, our $(\log n)$-th protocol assigns $O(\log^2 n)$ symmetric keys to each process. This is not far from the lower bound of $O(\log n)$ symmetric keys *which we show is* needed for each process to communicate securely with every other process in the network.) The protocols in our protocol family can be used to assign symmetric keys to the processes in a sensor network, or ad-hoc or mobile network, where each process has a small memory to store its assigned keys. We also discuss the vulnerability of our protocols to "collusion". In particular, we show that $\sqrt[k]{n}$ colluding processes can compromise the security of the k-th protocol in our protocol family.

I. Introduction

In this paper, we investigate the following interesting question. What is the smallest number of symmetric keys that need to be assigned to each process in a network of n processes so that each process can communicate securely with each other process in the network? The answer to this question is important in securing communications within a network of processes, where each process has a relatively small memory for storing its assigned keys. Examples of such network include sensor networks [1], [2], ad-hoc networks [3], [4], and mobile networks [5], [6].

The straightforward answer to the above question is $(n-1)$. This straightforward answer is so "natural" and "compelling" that it is hard to think of another answer to the question. As it happened, Gong and Wheeler did come up [7] with a better answer, of $O(\sqrt{n})$ keys assigned to each process in the network. Their elegant protocol for assigning symmetric keys to network processes is based on the view that the processes and the keys in the network can be arranged in a $\sqrt{n} \times \sqrt{n}$ grid, and a key is assigned to a process if the relative position of the key with respect to the process satisfies some condition.

Recently, Kulkarni, Gouda, and Arora presented [8] a variation of this grid protocol and showed that this protocol achieves the lower bound of assigning the smallest number of keys to each process in the network under the assumption that no two processes share more than two keys.

This last result leaves the door open for the following question. If we allow each pair of processes to share any number of keys, can fewer than $O(\sqrt{n})$ keys be assigned to each process and still each process can communicate securely with each other process in the network? In this paper, we demonstrate that the answer to this question is a resounding "yes".

We describe a family of $\log n$ protocols for assigning symmetric keys to the n processes in a network so that each process can securely communicate with each other process in the network. The k-th protocol, $1 \leq k \leq \log n$, in this family assigns $O(\sqrt[k]{n})$ keys to each process in the network. The first protocol in this family, where $k = 1$, is the straightforward protocol that assigns $(n-1)$ keys to each process. The second protocol in this family, where $k = 2$, is the grid protocol designed by Gong and Wheeler [7]. The last protocol in this family, where $k = \log n$, assigns $O(\log^2 n)$ keys to each process.

As discussed below in some detail, the k-th protocol in our family is based on our view that the processes and the keys in the network can be arranged in multiple grids, where each grid has $\sqrt[k]{n} \times \sqrt[k]{n}$ elements. The protocol assigns a key to a process if the relative position of the key with respect to the process satisfies some condition.

We start our presentation of our protocol family by presenting the second protocol, where $k = 2$, in the family.

II. A One-Grid Protocol

We consider a network that has n processes. Each process in the network has a unique identifier in the range $0 \ldots n-1$, represented by $\log n$ bits.

1-4244-0593-9/06/$20.00 ©2006 IEEE

We partition the $log\ n$ bits of the process identifiers into two parts, called A-bits and B-bits. As much as possible, each part has the same number of bits. For example, if each process identifier consists of 7 bits, say b_0 through b_6, then the A-bits are (b_0, b_1, b_2, b_3) and the B-bits are (b_4, b_5, b_6).

The A-bits can have $umax$ distinct values, 0 through $umax - 1$, and the B-bits can have $vmax$ distinct values, 0 through $vmax - 1$. Note that both $umax$ and $vmax$ are $O(\sqrt{n})$.

Consider a $umax \times vmax$ grid, where each element corresponds to a distinct process in the network. Specifically, an element (u, v) in this grid corresponds to the process whose A-bits has the value u and whose B-bits has the value v. We refer to this process as $p(u, v)$.

We specify two types of symmetric keys, called grid keys and direct keys, for the different elements in the grid, according to the following two rules.

i) For each grid element (u, v), specify a random *grid key* denoted $g(u, v)$.

ii) For each pair of grid elements (u, v) and (u', v'), where $u = u'$ or $v = v'$, specify a random *direct key* denoted $d(u, v)(u', v')$. Note that the direct key $d(u, v)(u', v')$ can also be denoted by $d(u', v')(u, v)$, since the order of the two pairs (u, v) and (u', v') is immaterial in the name of a direct key.

The specified grid and direct keys are assigned to the system processes as follows.

a) Each process $p(u, v)$ is assigned a copy of every grid key of the form $g(u, v')$ and a copy of every grid key of the form $g(u', v)$. Thus each process is assigned approximately $(umax + vmax)$ grid keys.

b) Each process $p(u, v)$ is also assigned a copy of every direct key of the form $d(u, v)(u, v')$ and a copy of every direct key of the form $d(u, v)(u', v)$. Thus each process is assigned approximately $(umax + vmax)$ direct keys.

It follows that the total number of keys assigned to each process is $2(umax + vmax)$ keys. Since each of $umax$ and $vmax$ is $O(\sqrt{n})$, the total number of keys assigned to each process is $O(4\sqrt{n})$.

When a process $p(u, v)$ needs to securely communicate with another process $p(u', v')$, $p(u, v)$ uses (u, v) and (u', v') to compute a non-empty subset SK of its own keys that satisfies the following two conditions.

1) *Sharing:*
Each key in the computed subset SK is assigned to both $p(u, v)$ and $p(u', v')$.

2) *Exclusion:*
No process, other than $p(u, v)$ and $p(u', v')$ is assigned all the keys in the computed subset SK.

After computing the subset SK, process $p(u, v)$ applies an exclusive-OR on the keys in SK in order to compute a single shared key that $p(u, v)$ can use to communicate securely with $p(u', v')$. Process $p(u', v')$ also computes the same subset SK and applies an exclusive-OR on the keys in SK in order to compute a single shared key that $p(u', v')$ can use to communicate securely with $p(u, v)$.

The algorithm that each of $p(u, v)$ and $p(u', v')$ use to compute the subset SK is shown in Figure 1.

Theorem *1:* The key subset SK computed by Algorithm 1 satisfies the two conditions of sharing and exclusion

Proof: Assume that process $p(u, v)$ needs to communicate securely with process $p(u', v')$ and so it uses Algorithm 1 to compute set SK of shared keys between $p(u, v)$ and $p(u', v')$. There are three cases to consider.

- *Case 1* ($u \neq u'$ and $v \neq v'$) :
  In this case, $SK = \{g(u, v'), g(u', v)\}$. Both $p(u, v)$ and $p(u', v')$ are assigned the two grid keys in SK and no other process is assigned both these keys. Thus, the computed SK satisfies the two conditions of sharing and exclusion.
- *Case 2* ($u \neq u'$ and $v = v'$) :
  In this case, $SK = \{d(u, v)(u', v)\}$. Both $p(u, v)$ and $p(u', v')$ are assigned the direct key in SK and no other process is assigned this key. Thus, the computed SK satisfies the two conditions of sharing and exclusion.
- *Case 3* ($u = u'$ and $v \neq v'$) :
  In this case, $SK = \{d(u, v)(u, v')\}$. Both $p(u, v)$ and $p(u', v')$ are assigned the direct key in SK and no other process is assigned this key. Thus, the computed SK satisfies the two conditions of sharing and exclusion.

■

III. A Three-Grid Protocol

In this section, we describe a second protocol for assigning symmetric keys to each process in a network of n processes. In this new protocol, we partition the $log\ n$ bits of process identifiers into three parts, called A-bits, B-bits, and C-bits. As much as possible, each part has the same number of bits. For example, if each process identifier consists of 7 bits, namely b_0 through b_6, then the A-bits are (b_0, b_1, b_2), the B-bits are (b_3, b_4), and the C-bits are (b_5, b_6).

The A-bits have $umax$ distinct values, the B-bits have $vmax$ distinct values, and the C-bits have $wmax$ distinct

```
1.  Initially , SK is empty.

2.  if u ≠ u' and v ≠ v' →
            add the two grid keys , g(u,v') and g(u',v) to SK
     [] u ≠ u' and v = v' →
            add the direct key d(u,v)(u',v) to SK
     [] u = u' and v ≠ v' →
            add the direct key d(u,v)(u,v') to SK
     [] u = u' and v ≠ v' → / impossible
            skip
    fi
```

Fig. 1. Algorithm 1 for Selecting Keys

figure

values. Note that each of the values $umax, vmax$ and $wmax$ is $O(\sqrt[3]{n})$.

In this protocol, we construct three grids called AB-grid, AC-grid and BC-grid:

- The AB-grid has $umax \times vmax$ elements.
- The AC-grid has $umax \times wmax$ elements.
- The BC-grid has $vmax \times wmax$ elements.

Each element (u, v) in the AB-grid corresponds to the set of all processes where the A-bits have the value u and where the B-bits have the value v. Thus, each element (u,v) in the AB-grid corresponds to the set of processes:
$p(u, v, 0), p(u, v, 1), \ldots, p(u, v, wmax - 1)$

Similarly, element (u, w) in the AC-grid corresponds to the set of processes:
$p(u, 0, w), p(u, 1, w), \ldots, p(u, vmax - 1, w)$. Also, element (v, w) in the BC-grid corresponds to the set of processes:
$p(0, v, w), p(1, v, w), \ldots, p(umax - 1, v, w)$.

It follows that each process $p(u, v, w)$ corresponds to three elements: (u, v) in the AB-grid, (u, w) in the AC-grid and (v, w) in the BC-grid.

Grid keys and direct keys are specified for the elements of each of the three grids. For example, keys are specified for the elements in the AB-grid according to the following two rules:

i) For each element (u, v) in the AB-grid, specify a random *grid key* denoted AB-$g(u, v)$.

ii) For each pair of elements (u, v) and (u', v'), where $u = u'$ or $v = v'$, in the AB-grid, specify a random *direct key* denoted AB-$d(u, v)(u', v')$.

In the same manner, grid and direct keys are specified for the elements of the AC-grid and for the elements of the BC-grid. Note that the grid and direct keys specified for the elements of the AC-grid are denoted AC-$g(u, w)$ and AC-$d(u, w)(u', w')$, respectively. Also, the grid and direct keys specified for the elements of the BC-grid are denoted BC-$g(u, w)$ and BC-$d(u, w)(u', w')$, respectively.

The specified grid and direct keys are assigned to the system process as follows:

a) Each process $p(u, v, w)$ is assigned a copy of every grid key of one of the following six forms:

$$\begin{aligned} &\text{AB-}g(u, v'), \quad \text{AB-}g(u', v), \\ &\text{AC-}g(u, w'), \quad \text{AC-}g(u', w), \\ &\text{BC-}g(v, w'), \quad \text{BC-}g(v', w) \end{aligned}$$

Thus, each process is assigned $2(umax + vmax + wmax)$ grid keys.

b) Each process $p(u, v, w)$ is assigned a copy of every direct key of the forms:

$$\begin{aligned} &\text{AB-}d(u, v)(u, v'), \quad \text{AB-}d(u, v)(u', v), \\ &\text{AC-}d(u, w)(u, w'), \quad \text{AC-}d(u, w)(u', w), \\ &\text{BC-}d(v, w)(v, w'), \quad \text{BC-}d(v, w)(v', w) \end{aligned}$$

Thus, each process is assigned $2(umax + vmax + wmax)$ direct keys.

Since each of $umax, vmax$, and $wmax$ is $O(\sqrt[3]{n})$, the total number of keys assigned to each process is $O(12\sqrt[3]{n})$.

When a process $p(u, v, w)$ needs to securely communicate with another process $p(u', v', w')$ to compute a nonempty subset SK of its own keys that satisfies the following two conditions.

1) *Sharing:*
Each key in the computed subset SK is assigned to both $p(u, v, w)$ and $p(u', v', w')$.

2) *Exclusion:*
No process, other than $p(u, v, w)$ and $p(u', v', w')$, is assigned all the keys in the computed subset SK.

After computing the subset SK, process $p(u, v, w)$ applies an exclusive-OR on the keys in SK in order to compute a single shared key that $p(u, v, w)$ can use to communicate securely with $p(u', v', w')$. Process $p(u', v', w')$ also computes the same subset SK and applies an exclusive-OR on the keys in SK in order to compute a single shared key that $p(u', v', w')$ can use to communicate securely with $p(u, v, w)$.

The algorithm that each of $p(u, v, w)$ and $p(u', v', w')$ use to compute the subset SK is shown in Figure 2.

From this algorithm, the number of keys in the computed subset SK depends on (u, v, w) and (u', v', w'). For example, SK has the maximum number of keys, six, when $u \neq u', v \neq v'$ and $w \neq w'$.

Theorem *2:* The key subset SK computed by Algorithm 2 satisfies the two conditions of sharing and exclusion

Proof: The proof of this theorem is similar to that of Theorem 1 ■

IV. A General Multi-Grid Protocol

In this section, we generalize the protocol in the previous section (where the bits of the process identifiers are partitioned into three parts) into a protocol, where the bits of the process identifiers are partitioned into k parts, A_0-bits, A_1-bits,... A_{k-1}-bits. As for the protocol in the previous section, these parts have equal number of bits, as much as possible.

For every $i, 0 \leq i < k$, the A_i-bits have $umax_i$ distinct values, 0 through $umax_i - 1$. Note that each $umax_i$ is $O(\sqrt[k]{n})$.

In this protocol, we construct $\binom{k}{2}$ grids. Each grid is called A_{ij}-grid, where i is in the range $0 \ldots k-2$, j is in the range $1 \ldots k-1$ and $i < j$. Each A_{ij}-grid has $umax_i \times umax_j$ elements.

Each element (u_i, u_j) in an A_{ij}-grid corresponds to the set of all processes where the A_i-bits have the value u_i, and where the A_j-bits have the value u_j. Each process $p(u_0, u_1, \ldots, u_{k-1})$ corresponds to $\binom{k}{2}$ elements: element (u_0, u_1) in the A_{01}-grid, element (u_0, u_2) in the A_{02}-grid, ..., and element (u_{k-2}, u_{k-1}) in the $A_{(k-2)(k-1)}$-grid.

Grid and direct keys are specified for the elements of every A_{ij}-grid according to the following two rules.

i) For each element (u_i, u_j) in the A_{ij}-grid, specify a random *grid key* denoted A_{ij}-$g(u_i, u_j)$.
ii) For each pair of elements (u_i, u_j) and (u'_i, u'_j), where $u_i = u'_i$ or $u_j = u'_j$, in the A_{ij}-grid, specify a random *direct key* denoted A_{ij}-$d(u_i, u_j)(u'_i, u'_j)$.

The specified grid and direct keys are assigned to the system processes as follows.

a) Each process $p(u_0, \ldots, u_{k-1})$ is assigned a copy of every grid key of one of the following two forms A_{ij}-$g(u_i, u'_j)$ and A_{ij}-$g(u'_i, u_j)$. Thus each process is assigned $(k-1)(umax_0 + umax_1 + \ldots + umax_{k-1})$ grid keys.
b) Each process $p(u_0, \ldots, u_{k-1})$ is assigned a copy of every direct key of one of the following two forms. A_{ij}-$d(u_i, u_j)(u_i, u'_j)$ and A_{ij}-$d(u_i, u_j)(u'_i, u_j)$ Thus, each process is assigned $(k-1)(umax_0 + umax_1 + \ldots + umax_{k-1})$ direct keys.

Since each $umax_i$ is $O(\sqrt[k]{n})$, the total number of keys assigned to each process is $O(2k(k-1)\sqrt[k]{n})$.

When a process $p(u_0, \ldots, u_{k-1})$ needs to securely communicate with another process $p(u'_0, \ldots, u'_{k-1})$, $p(u_0, \ldots, u_{k-1})$ computes a non-empty subset SK of its own keys that satisfies the following two conditions.

1) *Sharing:*
Each key in SK is assigned to both $p(u_0, \ldots, u_{k-1})$ and $p(u'_0, \ldots, u'_{k-1})$
2) *Exclusion:*
No process, other than $p(u_0, \ldots, u_{k-1})$ and $p(u'_0, \ldots, u'_{k-1})$ is assigned all the keys in SK.

After computing SK, process $p(u_0, \ldots, u_{k-1})$ applies an exclusive-OR on the keys in SK in order to compute a single shared key that $p(u_0, \ldots, u_{k-1})$ can use to communicate securely with $p(u'_0, \ldots,$
$u'_{k-1})$. Process $p(u'_0, \ldots, u'_{k-1})$ also computes the same subset SK and applies an exclusive-OR on the keys in SK in order to compute a single shared key that $p(u'_0, \ldots, u'_{k-1})$ can use to communicate securely with $p(u_0, \ldots, u_{k-1})$.

The algorithm that each of $p(u_0, \ldots, u_{k-1})$ and $p(u'_0, \ldots, u'_{k-1})$ use to compute the subset SK is shown in Figure 3.

The number of keys in the computed subset SK depends on $(u_0, \ldots, u_{k-1})$ and $(u'_0, \ldots, u'_{k-1})$. For example, SK has the maximum number of keys, $k(k-1)$, when $u_i \neq u'_i$ for every $i = 0, \ldots, k-1$.

Theorem *3:* The key subset SK computed by Algorithm 3 satisfies the two conditions of sharing and exclusion

Proof: The proof of this theorem is similar to that of Theorem 1 ■

V. The Key Grid Protocol Family

In the previous sections, we described a family of protocols for assigning symmetric keys to n processes so that every

1. Initially, SK is empty.

2. Use (u, v), (u', v'), and the keys specified for the elements of the AB–grid to add some keys to SK as follows:
   if $u \neq u'$ and $v \neq v' \rightarrow$
   add the two grid keys, AB–$g(u, v')$ and AB–$g(u', v)$ to SK
   [] $u \neq u'$ and $v = v' \rightarrow$
   add the direct key AB–$d(u, v)(u', v)$ to SK
   [] $u = u'$ and $v \neq v' \rightarrow$
   add the direct key AB–$d(u, v)(u, v')$ to SK
   [] $u = u'$ and $v \neq v' \rightarrow$
   add the direct key AB–$d(u, v)(u, v)$ to SK
   fi

3. Repeat Step 2, but this time use (u, w), (u', w'), and the keys specified for the elements of the AC–grid to add more keys to SK.

4. Repeat Step 2, but this time use (v, w), (v', w'), and the keys specified for the elements of the BC–grid to add more keys to SK.

Fig. 2. Algorithm 2 for Selecting Keys

figure

1. Initially, SK is empty.

2. For each A_{ij}–grid do
   if $u_i \neq u'_i$ and $u_j \neq u'_j \rightarrow$
   add the two grid keys, A_{ij}–$g(u_i, u'_j)$ and A_{ij}–$g(u'_i, u_j)$ to SK
   [] $u_i \neq u'_i$ and $u_j = u'_j \rightarrow$
   add the direct key A_{ij}–$d(u_i, u_j)(u'_i, u_j)$ to SK
   [] $u_i = u'_i$ and $u_j \neq u'_j \rightarrow$
   add the direct key A_{ij}–$d(u_i, u_j)(u_i, u'_j)$ to SK
   [] $u_i = u'_i$ and $u_j = u'_j \rightarrow$
   add the direct key A_{ij}–$d(u_i, u_j)(u_i, u_j)$ to SK
   fi

Fig. 3. Algorithm 3 for Selecting Keys

figure

pair of processes can communicate securely, using the key assigned to the process pair. The k-th protocol in this family is distinguished by partitioning the process identifiers into k parts of almost equal number of bits, where $k = 1, \ldots, \log n$. The protocol where $k = 2$ is described in Section II, the protocol where $k = 3$ is described in Section III, and the protocol where k has any value in the range $2 \ldots \log n$ is described in Section IV.

In this section, we focus our attention on the first protocol, where $k = 1$, and on the last protocol, where $k = \log n$, in the protocol family .

In the first protocol in our protocol family, each process identifier is partitioned into one part of $log\ n$ bits. In other words, each process is identified as $p(u)$, where $u = 0, \ldots, n-1$. In this case, each process $p(u)$ corresponds to element u in a one-dimensional grid.

For each pair u and u' of distinct elements in the one-dimensional grid, specify direct key $d(u)(u')$. (Note that no grid key is specified for each element in the one-dimensional grid.)

Assign to each process $p(u)$, every direct key of the form $d(u)(u')$. Thus each process is assigned $(n-1)$ direct keys.

When a process $p(u)$ needs to communicate securely with another process $p(u')$, $p(u)$ uses the direct key $d(u)(u')$. Note

that the direct key $d(u)(u')$ is assigned to the two processes $p(u)$ and $p(u')$ but not to any other process, and so $d(u)(u')$ satisfies the two conditions of sharing and exclusion.

In the last protocol in our protocol family, each process identifier is partitioned into $\log n$ parts of one bit each. Thus each process is identified as $p(u_0, \ldots, u_{\log n-1})$. This protocol is the one described in Section IV, when $k = \log n$. It follows that this protocol has $\binom{\log n}{2}$ grids. Also, each process is assigned $O(2\log n(\log n - 1)\sqrt[\log n]{n}) = O(4\log^2 n)$ keys. Moreover, subset SK has at most $\log n(\log n - 1)$ keys.

Table I summarizes the results of the different protocols in the protocol family. This table shows that our protocol family exhibits a trade-off between two important parameters: the number of keys assigned to each process and the maximum number of keys in the subset SK. (Note that the first parameter measures the size of storage needed to store the assigned keys in each process. The second parameter measures the length of time needed by each process to compute the key that this process can use to communicate securely with another process.) If the value of k is small, say 1, 2, or 3, then the number of keys assigned to each process is relatively large and the maximum number of keys in SK is relatively small. On the other hand, if the value of k is large, say $\log n$, then the number of keys assigned to each process is relatively small, but the number of keys in SK is relatively large.

VI. The Lower Bound

The last protocol in our protocol family assigns a small number of keys, namely $O(4\log^2 n)$ keys, to each of the n processes. This observation suggests the following important question: What is the smallest number of keys that need to be assigned to each process in order that each process can communicate securely with each other process? In this section we show that the answer to this question is $O(\log n)$ keys. This indicates that the last protocol in our protocol family is not far from being optimal in this regard.

Theorem *4:* In a a network of n processes, each process needs to be assigned at least $O(\log n)$ symmetric keys in order that each process is able to communicate securely with each other process in the network.

Proof: Assume that each process p in this network is assigned x keys. Process p needs to use a different non-empty subset of its x keys to communicate securely with each other process in the network. Because there are $2^x - 1$ non-empty subsets of the set of x keys, and process p needs to communicate securely with $(n-1)$ processes, we have

$$2^x - 1 \geq n - 1$$

Thus, $x \geq \log n$

■

Theorem 4 establishes a lower bound on the number of keys that need to be assigned to each process in a network of n processes so that each process can communicate securely with each other process in the network. In the remainder of this section, we show that this lower bound is tight by proving that it is possible to assign $O(\log n)$ keys to each process in the network and still allow each process to communicate securely with each other process in the network.

Consider the case where $12\log n$ distinct keys are to be distributed randomly among n processes in a network, and assume that each process is assigned $9\log n$ distinct keys. In this case, each two distinct processes are assigned at least $6\log n$ keys in common. Because the keys are assigned randomly to the network processes, the resulting key assignment can be either secure or insecure. In what follows we show that the probability that the resulting key assignment is insecure is strictly less than one. This implies that the probability that the resulting key assignment is secure is strictly more than zero.

probability that the resulting key assignment is insecure

= probability that there exists three distinct processes p, p' and p'' such that all the keys that are assigned to both p and p' (and so can be used by p and p' to communicate securely with one another) are also assigned to p''.

$$\leq \binom{n}{2}(n-2).\frac{X}{Y}$$

$$< \frac{n^3}{2}\frac{X}{Y}$$

Note that the factor $\binom{n}{2}$ is the number of choices of the distinct processes p and p' from the set of n processes in the network. Factor $(n-2)$ is the number of choices of process p'' that is distinct from both p and p'. Factor X is the number of ways of assigning $9\log n$ keys to process p'' under the assumption that $6\log n$ of those keys are those shared keys between p and p'. Thus, $X = \binom{6\log n}{3\log n}$. Factor Y is the number of ways of assigning $9\log n$ keys to process p''. Thus $Y = \binom{12\log n}{9\log n}$. Therefore,

probability that the resulting key assignment is insecure

$$< \frac{n^3}{2}\frac{\binom{6\log n}{3\log n}}{\binom{12\log n}{9\log n}}$$

$$= \frac{n^3}{2}\frac{(6\log n)(6\log n - 1)\ldots(3\log n + 1)}{(12\log n)(12\log n - 1)\ldots(9\log n + 1)}$$

$$< \frac{n^3}{2}\underbrace{\frac{(6\log n)(6\log n)\ldots(6\log n)}{(12\log n)(12\log n)\ldots(12\log n)}}_{3\ \log n \text{ times}}$$

k-th Protocol, where $k =$	1	2	3	k	$\log n$
Number of Grids	1, one-dimension	$\binom{2}{2}$	$\binom{3}{2}$	$\binom{k}{2}$	$\binom{\log n}{2}$
Number of keys assigned to a process	n - 1	$4\sqrt[2]{n}$	$12\sqrt[3]{n}$	$2k(k-1)\sqrt[k]{n}$	$4\log^2 n$
Max number of keys in SK	1	2	6	$k(k-1)$	$\log n(\log n - 1)$

TABLE I

SUMMARY OF RESULTS FOR THE PROTOCOL FAMILY

table

$$\begin{aligned} &= \frac{n^3}{2}\left(\frac{1}{2}\right)^{3\log n} \\ &= \frac{1}{2} \\ &< 1 \end{aligned}$$

This completes our proof of the following theorem.

Theorem *5:* Given a network of n processes and a set of $12\log n$ distinct keys, there is a protocol for assigning $9\log n$ distinct keys from the given set of keys to each process in the network such that each process can use its assigned keys to communicate securely with each process in the network ■

Note that our proof of Theorem 5 is non-constructive: it proves the existence of a secure protocol that assigns $O(\log n)$ keys to each process in the network, but it does not specify how to design such a protocol. This leaves the last protocol in our protocol family, which assigns $O(4\log^2 n)$ keys to each process in the network as the known most efficient protocol for securely assigning keys.

VII. ANALYSIS OF COLLUSION

The security of each of our protocols can be threatened and attacked by "collusion" among some of the processes in the protocol. In particular, the colluding processes can pool their grid keys together (but not their direct keys), and use these pooled keys to listen on communications between some non-colluding processes.

For example, consider a network that consists of $n = 16$ processes and assume that the grid and direct keys are assigned to each process in the network according to the one-grid protocol discussed in Section II. Thus, the grid keys $g(0,2)$ is assigned to process $p(0,0)$ while grid key $g(1,3)$ is assigned to process $p(1,1)$. Now if the two processes $p(0,0)$ and $p(1,1)$ collude, then they can listen on the communication between the two processes $p(0,3)$ and $p(1,2)$, which are not colluding.

Note that the colluding processes do not pool their direct keys together since these keys cannot be used by the colluding processes to listen on any communication between any two non-colluding processes.

Note also that the non-zero number r of colluding processes in a network is in the range

$$2 \leq r \leq n-2$$

where n is the total number of processes in the network.

Recently, Kulkarni and Bezawada [9] have identified a metric for measuring the resistance of a key assignment protocol to collusion. This metric, called the r-collusion resistance, is defined as follows.

$$\text{r-collusion resistance} = \lim_{n\to\infty} \frac{Y}{Z}$$

where r is the number of colluding processes in the protocol, n is the total number of processes in the protocol, Y is the number of communications that are encrypted using keys other than those pooled together by the colluding processes, and Z is the total number of communications in the protocol. In our context, $Y =$ the number of communications that are encrypted using direct keys + the number of communications that are encrypted using grid keys other than those assigned to the colluding processes, and $Z = \frac{n(n-1)}{2}$.

Note that the value of the r-collusion resistance for any protocol is in the closed interval $[0,1]$. If the r-collusion resistance for a protocol is 0, then this protocol is said to offer no resistance to r colluding processes. If the r-collusion resistance for a protocol is 1 then this protocol is said to offer full resistance to r colluding processes. Otherwise, the protocol is said to offer a limited resistance to r colluding processes.

The next three theorems specify the r-collusion resistance for each protocol in our family.

Theorem *6:* For the first protocol in our protocol family, and for any value of r in the range $2,\ldots,n-2$,

$$r\text{-collusion resistance} = 1$$

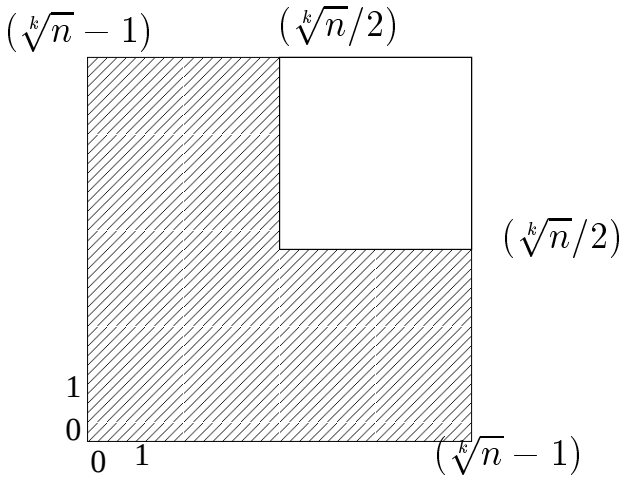

Fig. 4. Grid keys pooled by colluding processes

figure

Proof: All communications in the first protocol are encrypted using direct keys. Thus $Y = \frac{n(n-1)}{2}$, and so $Y/Z = 1$, and the r-collusion resistance = 1. ■

Theorem *7:* For the k-th protocol, where $2 \leq k \leq \log n$, in our protocol family, if $r \leq \frac{\sqrt[k]{n}}{2}$, then

$$r\text{-collusion resistance} \geq \frac{1}{16}$$

Proof: In the k-th protocol, where $2 \leq k \leq \log n$, each process is identified as $p(u_0, \ldots u_{k-1})$, where each u_i is in the range $0, \ldots (\sqrt[k]{n} - 1)$.

The first grid in this protocol is denoted A_{01}-grid. Assume that $(\frac{\sqrt[k]{n}}{2})$ of the processes collude. For the colluding processes to be assigned and pool together the maximum number of grid keys from the A_{01}-grid, no two colluding processes can be on the same row or on the same column in the A_{01}-grid. Thus, without loss of generality, let $u_0 = 0$ and $u_1 = 0$ for the 0^{th} colluding process, let $u_0 = 1$ and $u_1 = 1$ for the 1^{st} colluding process, and so on. The A_{01}-grid can be represented by Figure 4.

In Figure 4, all the grid keys in the dashed area are assigned to the colluding processes and can be pooled together. On the other hand all the grid keys in the undashed area are not assigned to any of the colluding processes. Then, if two uncolluding processes correspond to two elements in the undashed area, then the communication between these two processes cannot be listened on by the colluding processes. Because the undashed area is one fourth of the total area of the A_{01}-grid, it follows that $Y \geq \frac{(n/4)(n/4-1)}{2}$ and so

$$\begin{aligned} r\text{-collusion resistance} &= \lim_{n\to\infty} \frac{Y}{Z} \\ &\geq \lim_{n\to\infty} \frac{(n/4)(n/4-1)}{n(n-1)} \\ &= \frac{1}{16} \end{aligned}$$

■

Theorem *8:* For the k-th protocol, where k is a constant that does not depend on, in our protocol family, if $r \geq \sqrt[k]{n}$, then

$$\text{r-collusion resistance} = 0$$

Proof: The proof of this theorem is similar to that of Theorem 7 ■

VIII. Concluding Remarks

We presented a family of $\log n$ protocols for efficiently and securely assigning symmetric keys to the n processes in a network. Unfortunately, except for the first protocol, all other protocols in our protocol family are vulnerable to collusion attacks. The k-th protocol, $1 \leq k \leq \log n$, in the protocol family assigns $O(\sqrt[k]{n})$ symmetric keys to each process in the network, but the security of most communications is compromised if $\sqrt[k]{n}$ processes in the network decide to collude (and pool their grid keys together).

Deciding which protocol in our protocol family one should use to assign symmetric keys to the processes in a network depends on what one considers more important: efficiency or resistance to collusion. If one considers efficiency more important (than resistance to collusion), then one should use a k-th protocol, where k is relatively large say $(\log n)$ or $(\log n - 1)$. On the other hand, if one considers resistance to collusion more important (than efficiency), then one should use a k-th protocol, where k is relatively small say 1,2,3, or 4.

In this paper, we showed, in a non-constructive manner, that there is a protocol that assigns mere $O(\log n)$ keys to each of the n processes in the network (and still allows each process to communicate securely with each other process in the network). So far, we are unable to design such a protocol, but our search for this "amazing protocol" continues.

IX. Acknowledgements

The work of Lorenzo Alvisi is supported in part by the NSF under grants 0430510 and 0509338, and by a grant from the Texas Advanced Technology Program. The work of Mohamed G. Gouda is supported in part by the NSF, under grant 0520250.

REFERENCES

[1] L. Eschenauer and V. Gligor, "A key management scheme for distributed sensor networks," 2002. [Online]. Available: citeseer.ist.psu.edu/eschenauer02keymanagement.html

[2] A. Perrig, R. Szewczyk, V. Wen, D. E. Culler, and J. D. Tygar, "SPINS: security protocols for sensor netowrks," in Mobile Computing and Networking, 2001, pp. 189–199. [Online]. Available: citeseer.ist.psu.edu/perrig01spins.html

[3] J. Hubaux, L. Buttyan, and S. Capkun, "The quest for security in mobile ad-hoc networks." in ACM Symposium on Mobile Ad Hoc Networking and Computing, 2001.

[4] M. Tatebayashi, N. Matsuzaki, and J. David B. Newman, "Key distribution protocol for digital mobile communication systems," in CRYPTO '89: Proceedings on Advances in cryptology. New York, NY, USA: Springer-Verlag New York, Inc., 1989, pp. 324–334.

[5] J. Kong, P. Zefros, H. Luo, and L. Zhang, "Providing probust and ubiqutious security support for mobile ad-hoc networks," in IEEE International Conference on Network Protocols, 2001.

[6] V. Varadharajan and Y. Mu, "Design of secure end-toEnd protocols for mobile systems," in IFIP World Conference on Mobile Communications, 1996, pp. 258–266. [Online]. Available: citeseer.ist.psu.edu/varadharajan96design.html

[7] L. Gong and D. 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. [Online]. Available: citeseer.ist.psu.edu/gong90matrix.html

[8] S. S. Kulkarni, M. G. Gouda, and A. Arora, "Secret instantiation in ad-hoc networks," in In Workshop on Dependability Issues in Wireless Ad Hoc Networks and Sensor Networks (DIWANS) 2005. A final version of this paper has been published in Computer Communications, Vol. 29, pp. 200-215, 2006.

[9] S. S. Kulkarni and B. Bezawada, "A family of collusion resistant protocols for instantiating security," in ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols (ICNP'05). Washington, DC, USA: IEEE Computer Society, 2005, pp. 279–288.