The Best Keying Protocol for Sensor Networks

Taehwan Choi
Department of Computer Science
The University of Texas at Austin
Email: ctlight@cs.utexas.edu

H. B. Acharya
Department of Computer Science
The University of Texas at Austin
Email: acharya@cs.utexas.edu

Mohamed G. Gouda
The University of Texas at Austin
The National Science Foundation
Email: mgouda@nsf.gov

***Abstract*—Many sensor networks, especially mobile networks or those networks that are deployed to monitor crisis situations, are deployed in an arbitrary and unplanned fashion. Thus, any sensor in such a network may end up being adjacent to any other sensor in the network. To secure the communications between every two adjacent sensors in such a network, each sensor x in the network needs to store $n-1$ symmetric keys that x shares with the other sensors, where n is the number of sensors in the network. This storage requirement of the keying protocol is rather severe, especially when n is large and the available storage in each sensor is modest. Earlier efforts to redesign this keying protocol and reduce the number of keys to be stored in each sensor have produced protocols that are vulnerable to collusion. In this paper, we present a collusion-proof keying protocol where each sensor needs to store $\frac{(n+1)}{2}$ keys, which is much less than the $n-1$ keys in the original keying protocol. We also show that in any collusion-proof keying protocol, each sensor needs to store at least $\frac{(n-1)}{2}$ keys.**

I. Introduction

In many sensor networks, there is a need for secure communication; we would like to ensure that one sensor cannot eavesdrop on the communication between two other sensors. However, the situation is sometimes complicated by the additional problem that the topology of the network is not known before the sensors are deployed. For example, sensors may be mobile, so the topology of the network changes continuously [5]. Or it may be the case that the deployment has to be carried out immediately, and complete information about the field of deployment is not available [6], [8]. In such cases, the system designer does not have prior knowledge of which sensor will need to communicate with which other sensor. The naive solution to this problem is to provide each sensor with symmetric keys so it can communicate with every other sensor; this leads to each sensor storing $(n-1)$ symmetric keys, where n is the number of nodes in the network. As a sensor has very limited storage and computational power, for a large network with thousands of nodes, this naive scheme uses too much storage space.

There are two major approaches to the problem of reducing the number of keys stored in each node. The first solution is probabilistic [3]. Every sensor gets a random group of keys. Adjacent sensors check to see which keys they have in common. They then use the combination of their common keys as the shared key to encrypt messages. The hope is that the probability is very small that any sensor adjacent to both has all the same keys. This scheme is vulnerable in general, though the probability is small; clearly, it becomes far more vulnerable when some nodes collude and share all their keys.

The second solution to the key storage problem is deterministic, and also makes use of multiple part-keys. The basic idea is that each sensor stores some number of part-keys, and uses a combination of these keys as the shared key to encrypt secure communication. Each node stores a different subset of the set of available small keys. Hence, for every pair of sensor nodes, we choose the set of part-keys they have in common, and use the key formed of all these parts to encrypt their communication; if no other node has all the required part-keys, it cannot decrypt the conversation. This idea is developed in [7], [4], [1] and [2]. Unfortunately, this idea is also vulnerable to collusion - a small number of nodes can get together and pool their part-keys, and thereby obtain all the part-keys in the network; this enables them to break the security of communication at will.

This raises the question of whether it is even possible to solve the problem of key storage without making the network vulnerable to collusion. In this paper, we demonstrate that it is indeed possible, and not only show a lower bound for the number of keys in such a network, but also develop a keying scheme that approaches this lower bound. (We store exactly one key per node more than the lower bound.)

II. Sensor Networks and Adversaries

In this paper, we investigate a sensor network whose topology is not planned in advance, prior to the deployment of the network. Thus, when the network is deployed, any sensor can end up being adjacent to any other sensor in the network.

(There are many occasions when a sensor network needs to be deployed before its topology can be planned in great detail. For example, when a wildfire breaks out unexpectedly, a sensor network that monitors the fire may need to be deployed in a hurry, before the network topology can be planned accurately. A second example, when a sensor network is deployed in a battlefield whose perimeter is continuously changing, the topology of the network cannot be determined fully until the time when the network is to be deployed. As a third example, if the deployed sensor network is mobile, then a detailed plan of the initial topology may be of little value.)

In this network, when a sensor x is deployed, it first attempts to identify the identity of each sensor adjacent to x, then starts to exchange data with each of those adjacent sensors.

Any sensor z in this network can be an "adversary", and can attempt to disrupt the communication between any two legitimate sensors (say sensors x and y) by launching the following two attacks:

1) **Impersonation Attack:**
   Sensor z notices that it is adjacent to sensor x while sensor y is not. Thus, sensor z attempts to convince sensor x that it (z) is in fact sensor y. If sensor z succeeds, then sensor x may start to exchange data messages with sensor z, thinking that it is communicating with sensor y.
2) **Eavesdropping Attack:**
   Sensor z notices that it is adjacent to both sensors x and y, and that sensors x and y are adjacent to one another. Thus, when sensors x and y start to exchange data messages, sensor z can copy each exchanged data message between x and y.

To defend against these two types of attacks, sensors x and y need to share a symmetric key, denoted $K_{x,y}$. The key $K_{x,y}$ needs to be stored in both x and y, and not in any other sensor in the network, before these two sensors are deployed in the network. In Sections IV and V below, we show how sensors x and y can use their shared key $K_{x,y}$ to defend against these two types of attacks.)

It follows from this discussion that each sensor x should store a symmetric key $K_{x,y}$, for every sensor y that is adjacent to sensor x in the network, before the network is deployed. Unfortunately, as mentioned above, the facts of which sensor is adjacent to which other sensor can be deduced only after the network is deployed. Therefore, each sensor x should store a symmetric key $K_{x,y}$ for every other sensor y in the network, before the network is deployed.

If the network has n sensors, then each sensor in the network needs to store $(n-1)$ symmetric keys before the network is deployed. If n is large, then the storage requirement, just to store the required shared keys, is relatively large, especially since the size of storage in each sensor is relatively small.

To solve this problem, we present the following two results in this paper:

1) **Efficiency:**
   There is a keying protocol, where each sensor ends up sharing a distinct symmetric key with every other sensor in the network, and yet each sensor needs to store exactly $\frac{n+1}{2}$ symmetric keys, before the network is deployed.
2) **Optimality:**
   In every keying protocol, where each sensor ends up sharing a distinct symmetric key with every other sensor in the network, each sensor needs to store at least $\frac{n-1}{2}$ symmetric keys, before the network is deployed.

III. The Keying Protocol

We consider a network of n sensors. Without loss of generality, we assume that n is an odd positive integer. Each sensor in the network has a unique identifier in the range $0 \ldots n-1$. We use ix and iy to denote the identifiers of sensors x and y, respectively, in this network.

Each two sensors, say sensors x and y, share a symmetric key denoted $K_{x,y}$ or $K_{y,x}$. Only the two sensors x and y know their shared key $K_{x,y}$. And if sensors x and y ever become neighbors in the network, then they can use their shared symmetric key $K_{x,y}$ to perform two functions:

1) **Mutual Authentication**:
   Sensor x authenticates y and sensor y authenticates x.
2) **Confidential Data Exchange**:
   Encrypt all the exchanged data messages between x and y.

(Note that sensors x and y can become neighbors in the network in two occasions. First, the two sensors x and y could be mobile and their movements cause them to become adjacent to one another. Second, the two sensors could be stationary and they are deployed adjacent to one another.)

It follows from the above discussion that each sensor x in the network needs to store $n-1$ shared symmetric keys, namely $K_{x,y}$ for every y different from x.

In the remainder of this section, we show that if the shared symmetric keys are designed to have a "special structure", then each sensor needs to store only $\frac{(n+1)}{2}$ shared symmetric keys. But before we present the special structure of the shared keys , we need to introduce two new concepts: "universal keys" and "a circular relation, named below, over the sensor identifiers".

Each sensor x in the network stores a symmetric key, called the *universal key* of sensor x. The universal key of sensor x, denoted ux, is known only to sensor x.

Let ix and iy be two distinct sensor identifiers. (Recall that both ix and iy are in the range $0 \ldots n-1$, where n is the odd number of sensors in the sensor network.) Identifier ix is said to be *below* identifier iy iff exactly one of the following two conditions holds:

1) $ix < iy$ and $(iy - ix) < n/2$
2) $ix > iy$ and $(ix - iy) > n/2$

The below relation is better explained by an example. Consider the case where $n = 5$. In this case, the sensor identifier s are 0, 1, 2, 3, and 4, and we have:

- Identifier 0 is below identifiers 1 and 2.
- Identifier 1 is below identifiers 2 and 3.
- Identifier 2 is below identifiers 3 and 4.
- Identifier 3 is below identifiers 4 and 0.
- Identifier 4 is below identifiers 0 and 1.

The next three theorems, concerning the *below* relation, are in order.

Theorem 1. *For any two distinct sensor identifiers ix and iy, one of the following two statements is true.*

1) *ix is below iy.*
2) *iy is below ix.*

Proof: Let ix and iy be any two distinct sensor identifiers. Thus, ix and iy are two distinct integers in the range $0 \ldots (n-1)$. Without loss of generality, assume that $ix < iy$. Because n is an odd integer, exactly one of the following two statements holds.

(1) $iy - ix < \frac{n}{2}$
(2) $iy - ix > \frac{n}{2}$

If statement (1) holds then ix is below iy. Otherwise statement (2) holds and iy is below ix. ■

Theorem 2. *For each sensor identifier ix, the number of distinct sensor identifiers iy, where ix is below iy, is $\frac{(n-1)}{2}$.*

Proof: Each of the following $\frac{(n-1)}{2}$ sensor identifiers is below ix:

$$(ix - 1) \bmod n$$
$$(ix - 2) \bmod n$$
$$\ldots$$
$$(ix - \frac{n-1}{2}) \bmod n$$

Also, ix is below each of the following $\frac{(n-1)}{2}$ sensor identifiers:

$$(ix + 1) \bmod n$$
$$(ix + 2) \bmod n$$
$$\ldots$$
$$(ix + \frac{n-1}{2}) \bmod n$$

Thus, the number of distinct sensor identifiers iy, where iy is below ix, is $\frac{(n-1)}{2}$.

Also, the number of distinct sensor identifiers iy, where ix is below iy, is $\frac{(n-1)}{2}$. ■

Theorem 3. *For each sensor identifier ix, the number of distinct sensor identifiers iy, where iy is below ix, is $\frac{(n-1)}{2}$.*

Proof: The proof is similar to that of Theorem 2. ∎

The *special structure* of the symmetric key $K_{x,y}$, in the case where ix is below iy, is defined as follows:

$$K_{x,y} = H(ix|uy)$$

where

H is a secure hash function
$|$ is the concatenation operator
ix is the identifier of sensor x
uy is the universal key of sensor y

Note that in this case (where ix is below iy), the symmetric key $K_{x,y}$ needs to be stored in sensor x only since sensor y can compute this key (using $H, |, ix$, and uy) whenever it needs it.

Note also that in the other case (where iy is below ix), the special structure of the symmetric key $K_{x,y}$ is $H(iy|ux)$. And in this case, $K_{x,y}$ needs to be stored in sensor y only since sensor x can compute this key whenever it needs it.

The correctness of this keying protocol follows from the next theorem.

Theorem 4. *If a sensor identifier ix is below a sensor identifier iy, then the symmetric key $K_{x,y} = H(ix|uy)$ is stored in sensor x and can be computed by sensor y when needed. No other sensor stores $K_{x,y}$ or can compute it.*

Proof: Assume that a sensor identifier ix is below a sensor identifier iy. By our keying protocol the symmetric key that is shared between sensors x and y, namely $H(ix|uy)$, is stored in sensor x only. Moreover, because sensor y is the only one that knows the universal key uy, only sensor y can compute the key $H(ix|uy)$. ∎

Theorem 5. *Each sensor x stores one universal key ux and $\frac{(n-1)}{2}$ symmetric keys $K_{x,y}$ for every sensor y, where ix is below iy.*

Proof: According to the above keying protocol, each sensor x stores its universal key ux. Also, each sensor x stores the symmetric keys $K_{x,y}$ that sensor x shares with every sensor y where ix is below iy. From Theorem 2, there are $\frac{(n-1)}{2}$ sensors y where ix is below iy. Therefore, each sensor x stores $\frac{(n-1)}{2}$ symmetric keys. ∎

IV. A Mutual Authentication Protocol

Before our sensors are deployed in the network, each sensor x is supplied with the following items:

1) One distinct identifier ix in the range $0 \dots n-1$
2) One universal key ux
3) $\frac{(n-1)}{2}$ symmetric keys $K_{x,y} = H(ix|uy)$ each of which is shared between sensor x and another sensor y, where ix is below iy

After every sensor is supplied with these items, the sensors are deployed in random locations in the network.

Now if two sensors x and y happen to become adjacent to one another, then these two sensors need to execute a mutual authentication protocol so that sensor x proves to sensor y that it is indeed sensor x and sensor y proves to sensor x that it is indeed sensor y.

The mutual authentication protocol consists of the following six steps.

Step 1: Sensor x selects a random nonce nx and sends a hello message that is received by sensor y.

$$x \rightarrow y : hello(ix, nx)$$

Step 2: Sensor y selects a random nonce ny and sends a hello message that is received by sensor x.

$$x \leftarrow y : hello(iy, ny)$$

Step 3: Sensor x determines whether ix is below iy. Then it either fetches $K_{x,y}$ from its memory or computes it. Finally, sensor x sends a verify message to sensor y.

$$x \rightarrow y : verify(ix, iy, H(ix|iy|ny|K_{x,y}))$$

Step 4: Sensor y determines whether iy is below ix. Then it either fetches $K_{x,y}$ from its memory or computes it. Finally, sensor y sends a verify message to sensor x.

$$x \leftarrow y : verify(iy, ix, H(iy|ix|nx|K_{x,y}))$$

Step 5: Sensor x computes $H(iy|ix|nx|K_{x,y})$ and compares it with the received $H(iy|ix|nx|K_{x,y})$. If they are equal, then x concludes that the sensor claiming to be sensor y is indeed sensor y. Otherwise, no conclusion can be reached.

Step 6: Sensor y computes $H(ix|iy|ny|K_{x,y})$ and compares it with the received $H(ix|iy|ny|K_{x,y})$. If they are equal, then y concludes that the sensor claiming to be sensor x is indeed sensor x. Otherwise, no conclusion can be reached.

V. A Data Exchange Protocol

After two adjacent sensors x and y have authenticated one another using the mutual authentication protocol described in the previous section, sensors x and y can now start exchanging data messages according to the following protocol. (Recall that nx and ny are the two nonces that were selected at random by sensors x and y, respectively, in the mutual authentication protocol.)

Step 1: Sensor x concatenates the nonce ny with the text of the data message to be sent, encrypts the concatenation using the symmetric key $K_{x,y}$, and sends the result in a data message to sensor y.

$$x \rightarrow y : data(ix, iy, K_{x,y}(ny|text))$$

Step 2: Sensor y concatenates the nonce nx with the text of the data message to be sent, encrypts the concatenation using the symmetric key $K_{x,y}$, and sends the result in a data message to sensor x.

$$x \leftarrow y : data(iy, ix, K_{x,y}(nx|text))$$

Sensors x and y can repeat Steps 1 and 2 any number of times to exchange data between themselves.

VI. Optimality of Keying Protocol

According to our keying protocol, described in Section III, each sensor in the network is required to store only $\frac{(n+1)}{2}$ keys. Thus, the total number of keys that need to be stored in the sensor network is $\frac{n(n+1)}{2}$. (This is much better than storing $n(n-1)$ keys in the sensor network as dictated by the straightforward keying protocol.)

Despite the big saving in storage, that is achieved by our keying protocol, one wonders "Is there another keying protocol that requires the network to store much less than $\frac{n(n+1)}{2}$ keys?" The following theorem indicates that the answer to this question is "No".

Theorem 6. *Each keying protocol requires the sensor network to store at least $\frac{n(n-1)}{2}$ keys.*

Proof: There are $\frac{n(n-1)}{2}$ distinct symmetric keys in our sensor network. Thus, to prove that this theorem holds, it is sufficient to prove that every one of those symmetric keys, say $K_{x,y}$, causes a distinct key to be stored in sensor x or in sensor y. We carry out this proof by contradiction.

Assume that some symmetric key $K_{x,y}$ does not cause a distinct key to be stored in either sensor x or in sensor y. In this case, sensor x stores a key kx that x can use to compute at least two symmetric shared keys $K_{x,y}$ and $K_{w,x}$ as follows.

$$K_{x,y} = F(iy, kx) \tag{1}$$

$$K_{w,x} = F(iw, kx) \tag{2}$$

where F is a well-known function that can be used by each sensor to compute its shared keys from its stored keys.

Similarly, sensor y stores a key ky that y can use to compute at least two symmetric shared keys $K_{x,y}$ and $K_{y,z}$ as follows.

$$K_{x,y} = F(ix, ky) \tag{3}$$

$$K_{y,z} = F(iz, ky) \tag{4}$$

From (1) and (3) above, we have

$$F(iy, kx) = F(ix, ky) \tag{5}$$

Sensor x should not be allowed to utilize (5) and deduce key ky (in order that x be prevented from computing the shared key $K_{y,z}$). Therefore, there should not be any effectively computable function G, such that

$$G(ix, F(ix, ky)) = ky \tag{6}$$

Similarly, sensor y should not be allowed to utilize (5) and deduce key kx (in order that y be prevented from computing the shared key $K_{w,x}$). Therefore, there should not be any effectively computable function H, such that

$$H(iy, F(iy, kx)) = kx \tag{7}$$

From (6) and (7), we conclude the following.

(i) Because there is no effectively computable function G that satisfies (6), there is no effective way to compute key ky in sensor y from key kx in sensor x before the two sensors x and y are deployed in the network.

(ii) Because there is no effectively computable function H that satisfies (7), there is no effective way to compute key kx in sensor x from key ky in sensor y before the two sensors x and y are deployed in the network.

From (i) and (ii), we conclude that the two secrets kx and ky cannot be computed and stored in sensors x and y respectively. Contradiction! ∎

A keying protocol is called *uniform* iff this protocol requires each sensor in the network to store the same number of keys. Notice that the keying protocol described in Section III is uniform. Notice also that the next theorem, concerning uniform keying protocols, follows from Theorem 6.

Theorem 7. *Each uniform keying protocol requires each sensor in the network to store at least* $\frac{(n-1)}{2}$ *keys.*

From Theorem 7, our keying protocol requires each process to store no more than one key beyond the number of keys that need to be stored in each process by the best uniform keying protocol. Thus, for all practical purposes, our protocol is the best uniform keying protocol for sensor networks.

VII. Concluding Remarks

Typically, each sensor in a sensor network with n sensors needs to store $n-1$ shared symmetric keys to communicate securely with each other. Thus, the number of shared symmetric keys stored in the sensor network is $n(n-1)$. However, the optimal number of shared symmetric keys for secure communication, theoretically, is $\binom{n}{2} = \frac{n(n-1)}{2}$. Although there have been many approaches that attempt to reduce the number of shared symmetric keys, they lead to a loss of security: they are all vulnerable to collusion. In this paper, we show the best keying protocol for sensor networks, that needs to store only $\frac{(n+1)}{2}$ shared symmetric keys to each sensor. The number of shared symmetric keys stored in a sensor network with n sensors is $\frac{n(n+1)}{2}$, which is close to the optimal number of shared symmetric keys for any key distribution scheme that is not vulnerable to collusion.

It may be noted that in addition to the low number of keys stored, and the ability to resist collusion between sensors, our keying protocol has two further advantages. Firstly, it is uniform: we store the same number of keys in each sensor. Secondly, it is computationally cheap, and thus suitable for a low-power computer such as a sensor: when two sensors are adjacent to each other, the computation of a shared symmetric key requires only hashing, which is a cheap computation and can be done fast. As our protocol has many desirable properties, such as efficiency, uniformity and security, we call this protocol the best keying protocol for sensor networks.

References

[1] A. Aiyer, L. Alvisi, and M. Gouda. Key grids: A protocol family for assigning symmetric keys. *Network Protocols, IEEE International Conference on*, 0:178–186, 2006.

[2] E. S. Elmallah, M. G. Gouda, and S. S. Kulkarni. Logarithmic keying. *ACM Trans. Auton. Adapt. Syst.*, 3:18:1–18:18, December 2008.

[3] L. Eschenauer and V. D. Gligor. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS '02, pages 41–47, 2002.

[4] L. Gong and D. J. Wheeler. A matrix key-distribution scheme. *Journal of Cryptology*, 2:51–59, January 1990.

[5] A. Howard, M. J. Mataric, and G. S. Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. pages 299–308, 2002.

[6] S. Hynes and N. C. Rowe. A multi-agent simulation for assessing massive sensor deployment. *JOURNAL OF BATTLEFIELD TECHNOLOGY*, 7:23–36, 2004.

[7] S. S. Kulkarni, M. G. Gouda, and A. Arora. Secret instantiation in ad-hoc networks. *Special Issue of Elsevier Journal of Computer Communications on Dependable Wireless Sensor Networks*, 29:200–215, 2005.

[8] S. Sana and M. Matsumoto. A wireless sensor network protocol for disaster management. In *Information, Decision and Control, 2007. IDC '07*, pages 209 –213, 2007.