CS 588: Cryptography

Security in Sensor Networks

By:
Stavan Parikh
Tracy Barger
David Friedman

Date: December 6, 2001

Table of Contents

1. Introduction

Advances in nanotechnology, wireless communication, and computing are driving the development of new embedded computing devices. Distributed Sensor Networks (DSNs) are one class of these embedded devices. DSNs have a large number of small sensor nodes, with each node consisting of some computing power, limited memory, various sensors, and wireless communication capabilities.

This project focuses on security of DSNs. Due to various limitations faced by DSNs, like memory, processing power and battery life, it is not feasible to use the current techniques of security and authentication of data without modification. This paper looks at key distribution and encryption techniques that are in wide use today and could be adapted to a DSN environment. Moreover, this paper provides an analysis of SPINS: the only security and authentication algorithm available today written specifically for a DSN environment.

The need for security in DSNs arises due to their inherent structure and the applications envisioned for them. A DSN is expected to have thousands of sensor nodes, all of them working autonomously together to accomplish a task. This has led to a new paradigm of computer programming called Swarm Computing. The idea is to have the ability to issue high-level commands to a group of devices that autonomously can work together to carry out that command. Research in the field of swarm computing is still in its infancy. All communication is going to take place wirelessly. To allow for proper functioning, it is necessary that these swarms of nodes can authenticate the commands they receive and know that the commands are from authorized sources.

Depending on the application, it may be necessary that the wireless message exchange be secure from eavesdroppers. The defense establishment would like to use DSNs for battlefield surveillance by deploying sensor nodes in hostile environments for sensing various properties like motion, audio, video, bio/chemical weapons and others. In such an application, security and authentication of data will be crucial. A DSN can be used for product quality monitoring by having temperature/humidity sensors within food products, or having impact and vibration monitoring in consumer electronics [18]. In such an environment, it will be necessary that no one can change the data stored on the motes. Smart home/office environments are ideal candidates for DSNs as they can be used for environment monitoring, user identification or tracking and space adjustments to user preferences. This application may require security for privacy purposes. Thus security and authentication form an important aspect of development of DSNs.

The rest of this paper is organized as follows: Section 2 describes the constraints and security requirements for sensor networks for this paper. Section 3 provides a look at key distribution techniques for groups of varying size. Section 4 describes authentication and a possible technique for achieving authentication. Encryption techniques are discussed in Section 5. We provide an overview and analysis of the SPINS protocol for security in sensor networks in Section 6. Finally, we end in section 7 by summarizing our results and stating some conclusions.

2. Sensor Networks

Sensor networks are made of a collection of sensor nodes. Each node has various sensors, a small processor and memory, and a RF or Optical transmitter/receiver for communication. The nodes within a network are controlled by one or more base stations which are computers with much higher processing capabilities. The following sections explicitly define the capabilities of a sensor node for the purpose of this project and set the security requirements for a DSN.

2.1. Constraints

A sensor network is constrained due to the limitations of the hardware of each of the nodes and due to its environment. These constraints are what greatly differentiate a sensor network from other networks.

2.1.1. Hardware

As mentioned earlier the hardware capacities of a sensor node are limited. For the purpose of this project, the following are the constraints in terms of hardware for a sensor node:

- ALU: 8 bit
- Memory: 10-15 Kb

The hardware itself will be cheap, and it will be reasonable to expect thousands of nodes to be a part of a given sensor network. Due to the large number of nodes, failure of any given node is not a major concern. Data obtained from a collection of nodes can be used to generate results. The size of these nodes will be small ranging from about 1” X 0.5” presently to about 1 mm^3 in size in the future or even smaller. An example hardware are “motes” developed at the University of California, Berkeley that are 1” in width (Figure 1).

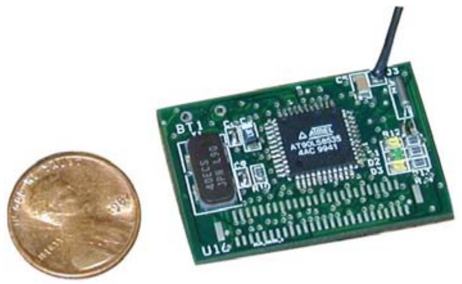
Figure 1: Motes (U.C.Berkeley)

2.1.2. Energy

The small size of the sensor node makes it difficult to package batteries with the nodes without sacrificing mobility. For this reason, the smallest possible batteries are used creating a tradeoff between battery size and total energy available. While both computation and communication take battery power, the power consumed by a communication is a thousand times that consumed by a computation. This energy limitation requires that we minimize communication in developing security solutions for DSNs. For the “motes” being developed at University of California, Berkeley, typical battery life is around 30 hours.

2.1.3. Communication & Addressing

All communication takes place wirelessly. The air inherently is a broadcast medium so all communication is via broadcast of messages. Due to the large number of nodes involved in a given DSN, it is not practical to address each node individually. So for the purposes of a security solution, it will not be possible to individually address a mote.

2.1.4. Trust Model

For the purposes of this research, the trust model is defined as follows:

- Base Station: The base station is always trusted. It is assumed to be secure (by some other method), and that it is immune to being broken into.
- Mote: A mote itself is secure i.e. a mote can trust itself to keep a secret unless someone captured the mote and looked at or tampered with it physically.
- Communication: All communication is over an insecure channel and is liable to eavesdropping. So all message communication needs to be encrypted. The encryption needs to be secure to keep a message secure until some time after the communication (for the remaining battery life of the mote).

2.2. Security Requirements

Depending on the application, DSNs will have different security requirements. However, there are some requirements of key establishment and authentication that are general to all DSNs. For secure communication between nodes in a DSN, a shared key needs to be established between the nodes. If authentication is a requirement, then it is necessary to provide a mechanism for the nodes to be able to authenticate the messages they receive. For secure communication and authentication, the problem reduces to that of key management. The following sections describe the key management requirements in more detail [2].

2.2.1. Confidentiality

The deployed shared key between nodes needs to remain confidential from unauthorized access. All communication between the nodes must employ encryption mechanisms using the keys to provide security from eavesdroppers. The keys should be established such that effect of the compromise of any given node is minimal.

2.2.2. Authenticity

If confidentiality of the key is maintained, a certain level of authentication is obtained as only a pre-defined set of nodes could have had a given key. Some applications may require additional authentication in which case special schemes for authentication may be necessary.

2.2.3. Integrity

The shared keys along with other confidential information must be immune to modification from outside sources not authorized to initiate such a modification.

2.2.4. Freshness

Depending on the environment in which sensor nodes are deployed, it may or may not be possible to reuse a given set of nodes. If they cannot be reused, as DSNs normally have limited battery life, it may be sufficient to ensure that keys are secure for the length of time motes are in operation. In the case that battery life is long or motes are too be reused, it is necessary that the amount of ciphertext exchanged with a given key be less than the minimum required for any reasonable cryptanalysis attack. In the case where nodes are reused, new members join or leave, or a key is suspected to be compromised, it may be necessary that the keys be refreshed to keep data confidentiality. Furthermore, when keys are refreshed it is necessary that no keys be reused, making cryptanalysis more challenging [2].

2.2.5 Scalability and Availability

As DSNs are expected to be comprised of a widely varying number of nodes, it is necessary that any security system developed be scalable. Furthermore, it is necessary that a security system be as non-intrusive as possible to the regular functioning of a node and that it be available for use at any time (single points of failures are discouraged).

3. Keying Protocols

3.1. Overview

This section describes some of the keying protocols that can be used for key establishment and distribution in sensor networks. Once established, sensors can use these keys for the encryption schemes described in section 5. In general, we can think of key distribution protocols as falling in three categories: pre-deployed, arbitrated, and self-enforcing autonomous keying protocols [2]. **Figure 2** shows an overview of the three main categories of key distribution protocols and the algorithm types within each.

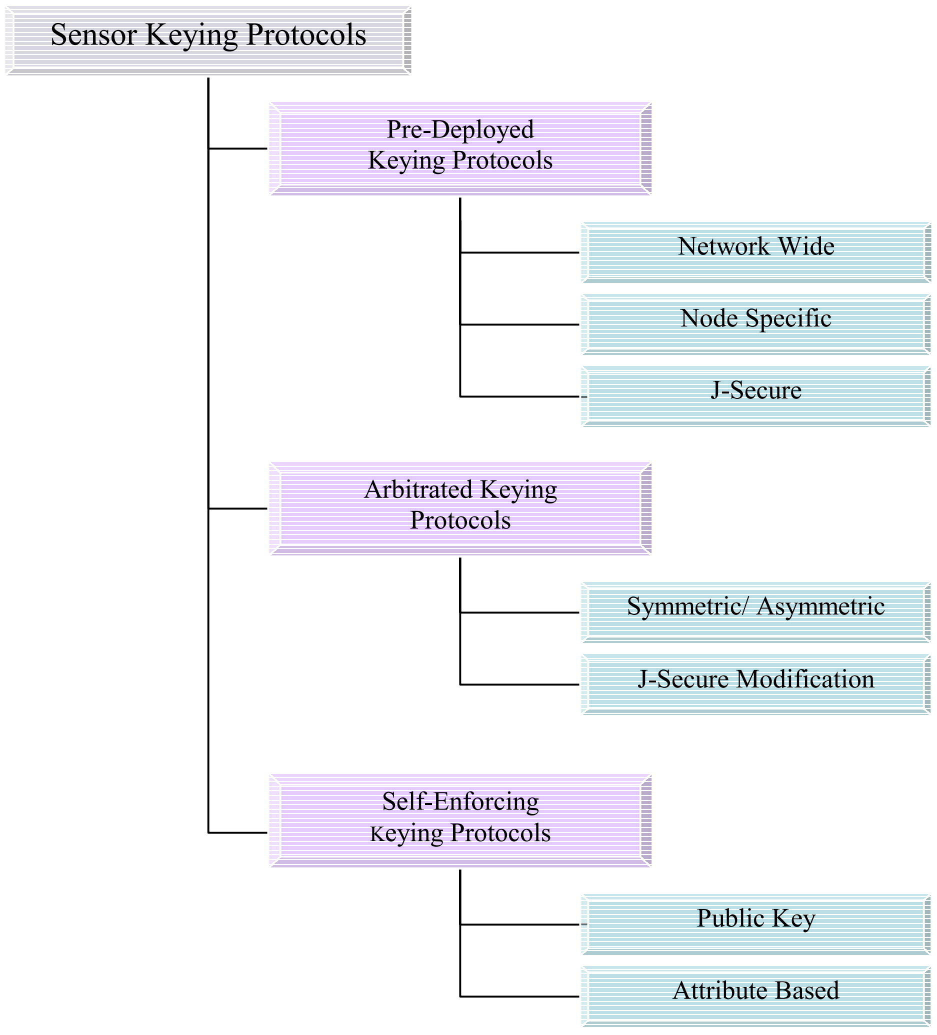

Figure 2: Keying Protocols Overview

3.2. Pre-deployed Keying

In pre-deployed keying protocols, extensive initial configuration of the sensor nodes is carried out that helps reduce transmission costs in a deployed network. Such pre-configuration reduces flexibility and may impact security if no mechanism for key refreshing is provided. There are three main schemes of pre-deploying keying:

3.2.1. Network-Wide Pre-deployed Keying

In network-wide pre-deployed keying, a single key that is to be shared by each node in the network is hard-coded in the sensors before their deployment. This allows a node to talk to any other node or to a base station by encrypting information with the given key. As there is only one key, storage is not an issue. The main advantage of this scheme lies in its simplicity. The major drawback of this scheme is that compromise of any one node makes all communication transparent to the eavesdropper.

Instead of deploying a key, it is possible that some other information is deployed that can be used to determine a key by the nodes. Such possibilities are explored further in the discussion of J-secure keying (sec. 3.2.3) and in attribute-based keying (sec. 3.4.1).)

3.2.2. Node-Specific Pre-deployed Keying

Instead of deploying one key for the whole network as done in a network-wide key scheme, in a node-specific key scheme a unique key pair is established between any two nodes and placed on the nodes. For n nodes this requires C(n,2) keys to be stored on each node. Such a scheme makes the overall network security immune to the compromise of any given node. However, placing C(n,2) keys of k bits each on each node is impractical for any reasonable sized DSN. Also if nodes are to be added at a later date, all nodes will have to be updated with a new key pair placing huge communication requirements on the network.

Alternatively, it is possible to create just n key pairs where n is the number of nodes in a network. Here each node will have one key stored to allow it to communicate with the base station. The main disadvantage here is that inter-node communication can only take place through the base station. Moreover, the base station becomes a single point of failure.

A possible scheme that takes the advantages provided by each of the above two schemes is a J-secure deployment scheme described here.

3.2.3 J-Secure Pre-Deployed Keying

The keying method described in 3.2.1 is vulnerable to compromise of any one node while the keying method described in 3.2.2. is secure against a coalition of any number of nodes. Reference [2] describes a scheme that divides the n nodes into m subgroups and issues C(m,2) keys to allow inter-group communication between any two groups. Here J in J-secure comes from the fact that there are C(m,2) = j+1 total keys. Some amount of security is available as long as less than j groups collide. This scheme has been studied and analyzed in reference [1].

Taking this scheme and combining it with the above schemes, we have developed a different j-secure scheme. In our case j stands for the number of subgroups. The idea is to divide a DSN of n nodes into j subgroups giving each subgroup a different key (Figure 3).

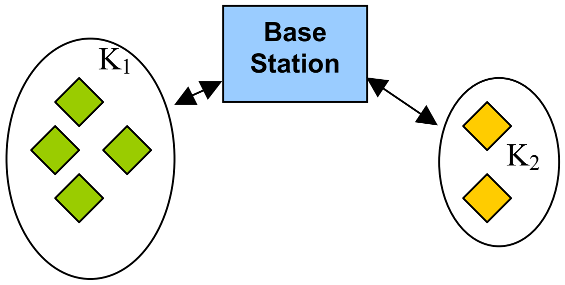

Figure 3: J-Secure Key Distribution

As shown in Figure 3, each group is given a symmetric key K_i that is known to all members of the group and the base station. Communication can take place between members of the group and between a group and base station securely using the given key. For inter-group communication, a message has to be sent through a base station. The groups are pre-determined at the time of deployment (we describe a better scheme for determining groups in section 3.3.1.).

The number of groups and the size of each group are what determine the security of this scheme. The size of a group can be selected based on application needs that may require some specific types of sensors to communicate. As described in 3.3.1, the location of the sensors on deployment can also be used to determine groups. This is useful because if nodes that are close together have the same key then they can use less energy while transmitting messages. Furthermore, smaller sized groups result in a greater number of groups and keys, leading to better security.

In one extreme, if there are n keys (each group size is 1), then this scheme is very secure and the same as the one described in 3.2.2. On the other end, if the group size is n (one key) then this scheme is not very secure and is similar to the scheme described in 3.2.1. The smaller the group size the less the effect of the compromise of a mote.

Ideally, we want small group sizes; however, there is a tradeoff between the group size and the number of communications in a multi-hop environment. This is illustrated through the following example. Consider that all motes are within a one hop distance of the base station (i.e. all motes can receive a base station transmission directly without other motes having to forward them). In

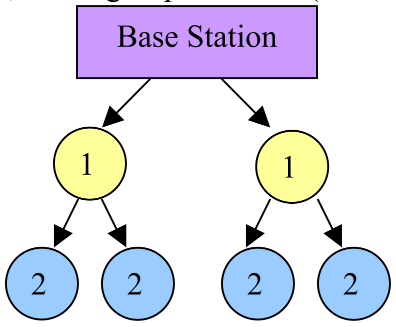

Figure 4: Multi-hop Communication

such cases, it may be ideal to have each group size equal one. Now consider that the group size is one, and there are a total of 6 groups as shown in Figure 4. Here the motes marked in blue are 2 hops away from the base station so any messages to motes labeled 2 have to be transmitted by a mote labeled 1. If each of the groups have different keys, then the base station makes six transmissions with each of the one-hop nodes making two more transmissions. On the other hand if the motes marked 2 had the same keys, then the nodes marked 1 would only have to make one transmission. Thus, the example demonstrates the tradeoff between the number of transmissions and the number of keys based on the distance of motes from the base station.

J-secure uses a delicate balance between security and complexity (in terms of the number of keys). It allows adjustments for memory restrictions and communication distance. Combined with an encryption technique it allows for secure message transmission. In addition, limited authentication is provided as the nodes having the same keys are predefined. Furthermore, compromise of any node just affects the nodes in that group keeping the rest of the network secure. Thus J-secure is an ideal technique to use for pre-deploying keys.

3.3. Arbitrated Protocols

Arbitrated protocols are protocols in which motes utilize a trusted third party (server or base station) during key establishment. These protocols require a limited amount of pre-deployed material to be placed on the sensor nodes. This material is then used to generate keys. Such protocols involve either secret or public-key protocols. Due to the limited computing power available to sensor nodes, exponentiation is impractical on the nodes and this makes public-key systems not viable. Therefore, any scheme that is considered for arbitrated protocols has to be a secret-key scheme.

To establish symmetric keys using a server, there are various options. We did not find any viable secret-key schemes in the literature that fit this environment correctly. As a result, we only describe one scheme in this section. Our scheme is based on the premise that once the nodes are deployed a server could use a pre-deployed one-time pad to establish keys for nodes. There are various ways to implement schemes based on this premise – we describe one particular technique that enhances the J-secure keying scheme discussed earlier.

3.3.1. J-Secure Modification

One significant problem with using pre-deployed J-secure keys is that the groups cannot be optimized to contain motes that are located close to each other. Optimization of groups helps in reducing the amount of power that must be expended in communications and the overall number of communications that must take place to pass on a message. To minimize this problem, we developed a scheme to allow J-secure keys to be distributed after the motes are deployed. It is important in developing such a scheme to minimize both computation and communication so that they do not outweigh the advantages of having more optimized groups.

In this protocol a one-time pad and random number generation scheme will be stored on each mote. A portion of the motes will be considered super nodes and given a number of precedence; outside of the number there is no difference between a super node and an ordinary mote and the designation is only for use in initially forming groups. After deployment of all motes, the base station will send out an unencrypted broadcast telling all the motes that they are activated. Next, each of the super nodes in order of precedence will send out an unencrypted message containing their number to all motes in a radius r. The message will be sent out at a specified amount of time after the initial receiving of the base station message (i.e. mote 1 sends out a message 1 ms after the message, mote 2 sends one out 2 ms after the message, etc.). Once nodes receive a message they become part of the specified group number. If they were originally designated super nodes, they do not send out a message later with their group number if they have already received another group number. This continues until either all super nodes have sent out a message or are in a group. There is a chance that some motes may never be put into a group, but this number should be minimal if the number of super nodes is high enough. Also, there is some tradeoff between the probability of all nodes being in a group (probability increases with larger number of super nodes) and time required to complete this initial protocol of putting all motes into groups (less super nodes, means less time).

At a specified amount of time after the initial activation of the motes, the base station will broadcast another message. The message will consist of a list of group numbers followed by seeds for random number generation encrypted with the one time pad found on each mote (each seed will be encrypted with a different part of the one time pad as to not use an insecure multi-time pad). Each mote will then decrypt the seed for its group and use the seed to generate a random number that will serve as the group key. The rationale behind the motes generating their own keys instead of receiving an encrypted message from the base station with the entire key is that it would likely require less power for a mote to generate its own key than for the transmissions necessary to distribute a larger message containing complete keys. Finally, after calculating their group keys, the motes will delete the initial one-time pads, random number generation code, and seeds to prevent someone who captures a mote from being able to discover information that would be valuable in determining other group keys.

The example shown in Figure 5 illustrates how this protocol would work. About one millisecond after the initial base station message, super node 1 would send the message "1" to the area shown in the group 1 circle. All those motes would then become part of group one. Since super node 2 already has a group number, it will not send out a message. Next, three milliseconds after the initial message, super node 3 will send out a message and all motes shown in the group 3 circle will become part of group three. The base station would then send out the message {1 E(Seed 1), 2 E(Seed 2), 3 E(Seed 3)}. All motes in group one would decrypt the first seed using the agreed number of first characters in the one time pad and use the seed to generate a random number to serve as the group key. Likewise, all motes in group three would decrypt the third seed using the third group of agreed number of characters in the one time pad, and then use the third seed to generate the group three key.

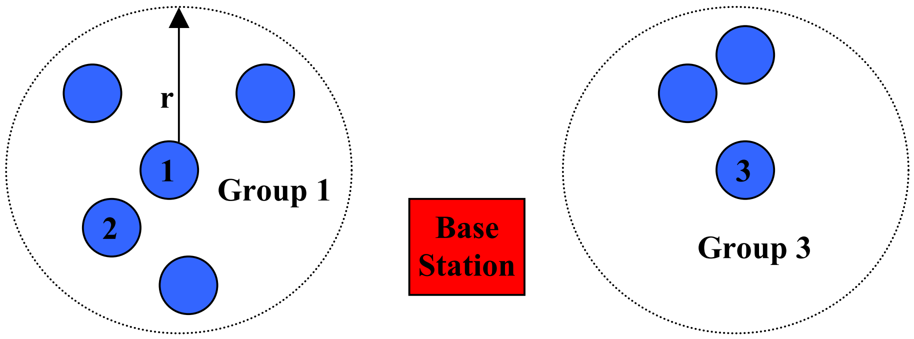

Figure 5: Modified J-Secure Protocol Example

This protocol has the advantage in comparison to pre-deployed keying of allowing groups of motes to form after deployment, enabling a more optimal arrangement of groups, saving power needed for communications. The algorithm does involve some additional computation and communication. Additional messages include relaying of the first and second base station messages that are sent, and the sending by super nodes of the group number. This latter communication will require less power than a message sent to an area farther away and should not have a huge effect on the resources of the mote. The relaying of the messages involves a maximum of only about two extra communications per mote. Additional computations involved in this algorithm include decryption of the seed, which should require minimum computation since it uses a one-time pad; generation of the random number to be used as a group key; and deletion of memory related to the one-time pad and random number generator. It is believed that the advantages in saved communication power outweigh these additional communications and computations.

It is necessary for any scheme that uses a server to establish keys after deployment, that messages from the server can be authenticated. In the case of the above scheme, the one-time pads are only stored on the server and the motes so the server is implicitly authenticated. For any other scheme another authentication scheme will be needed to be used. Also in sensor networks communication is what consumes the most energy. So if too much communication is required to establish or refresh keys, it may not be feasible to use such arbitrated protocols. Because of the large amount of communication required in most arbitrated schemes we analyzed, they are not presented here. In the case where all communication to the base station is single hop or when energy is not a major constraint, it may be viable to use other arbitrated protocols. In such cases, some schemes are suggested in [2].

3.4. Self-Enforcing Autonomous Keying Protocols

Unlike arbitrated protocols, self-enforcing autonomous keying protocols use point-to-point communication between nodes for key establishment and communication. Point-to-point communication is ideal as routing algorithms can be devised to use the shortest path between the sender and receiver, minimizing communication and conserving energy.

Most of the autonomous protocols use a public key scheme for authentication and security. As described earlier the current environment does not lend itself to a public key system. Attribute-based keying is one promising scheme within autonomous keying schemes that does not depend on public key but uses physical characteristics of the sensor, its location, and other pre-stored information [2].

3.4.1. Attribute-Based Keying [2]

In this scheme only nodes that have matching attributes to the sender would be able to decrypt a given message. It uses one way functions that compute a unique property based on sensor capabilities or location. This is then used for encryption. To make this more clear – take for example a group of n sensors with b of them being heat sensors and the remaining being motion sensors. Then the heat sensors can use a unique ID of their sensor type to encrypt a message so that the motion sensors cannot decrypt it. Another attribute could be distance from originator where a sensor node could correctly decrypt a message if it was within a given distance of the sending node.

A general scheme for attribute based keying is as follows:

A sending node creates the following packet to send:
E(K, Message) || SensorAttribute || Nonce

Where K is the key calculated as:
K = H(SensorAttribute || Nonce)

Here the SensorAttribute could be location, sensor types, or other preloaded material onto the sensors before deployment.

Such a scheme would be easy to implement as it would not require much preloaded material. Key uniqueness is provided by the physical characteristics of a sensor node or its environment. As the base station knows sensor attributes and algorithms to generate keys, it could also generate the right keys for all communications.

Though this scheme is easy to implement, it has many flaws. In reality, it would be easy for an adversary to determine the attribute being used for keying and thus make regenerating the key and compromising the messages easy to achieve. Thus, this scheme does not provide very good cryptographic protection, but does protect against casual eavesdropping. Furthermore, this scheme limits communication to a given area or given sensor type. For inter-group communication, a different scheme is required or communication is required to occur through a base station.

3.4. Conclusion

This section described a few keying schemes that we believe are viable options for DSNs. Each of these schemes meets the security requirements to different levels and the application of the DSN can be used to determine among the scheme chosen. Table 1 shows an overview of the scheme with respect to the security requirements described in section 2.

The scale used is as follows:
1 – None
2 – Little
3 – Medium
4 – High
5 – Total

	Pre-Deployed			Arbitrated	Self-Authenticating
	Network-wide	Node Specific	J – Secure	J-Secure Enhanced	Attribute Based
Confidentiality	2	5	4	4	2
Authenticity	1	5	3	3	2
Integrity	2	4	3	3	3
Freshness	5	1	3	3	4
Availability	5	5	3	3	3
Scalability	4	2	4	4	5

Table 1: Key Distribution Schemes – Security Requirements

Notes about table values:

- *Freshness* analysis is based on how easy it would be to refresh keys using a different key distribution algorithm like Key graphs given that nodes already have a means for secure communication.
- *Integrity* is based on how many keys need to be changed if an attacker wants to control all nodes in a DSN.
- *Availability* is based on how dependent a scheme is on the base station for node to node communication.

4. Authentication

Authentication allows the recipient of a message to verify who the sender of the message is. For our scheme it is necessary that messages sent by the base station are authenticated as these messages include commands for the functioning of the DSN. We have not considered mote to mote authentication because we defined our motes to not trust any other motes. Even if authentication existed on motes, if a mote is compromised then it could still send messages using the authentication technique even though the messages may be rogue. So there is no value in authenticating motes. The idea is that as long as enough motes are un-compromised the commands of the base station will be carried out correctly.

We looked at many authentication schemes and realized that most were based on public-key, which is not feasible to use in a DSN environment. So we propose a scheme based on the S-key hash-based authentication scheme.

4.1 Hash-based Authentication Scheme

Our main candidate for authentication was μTESLA (see section 6.5 for description). However, it was determined that the scheme made an invalid assumption in assuming that the base station and the nodes can stay time synchronized. We feel that this is an unfounded assumption given the hardware constraints of these devices. Because of this we developed our own protocol which is based on μTESLA and S-key.

These are the steps of our protocol:

1. The base station generates a X_n and a key chain: $H(X_n) = X_{n-1} \dots H(X_1) = X_0$.

2. The base station distributes X_0 to all of the nodes using a secure channel or X_0 is initialized in all nodes before deployment

3. The base station sends a message encrypted with the previously established symmetric key. The message contains whatever command the base station is sending to the motes concatenated with X_1, where the $H(X_1) = X_0$. The node then decrypts the message, calculates the hash of X_1, and checks that it is equivalent to the stored value for X_0; if so the node knows that the message came from the base station. Later messages are authenticated in a similar manner. Figure 6 below illustrates the scheme. Here K is a symmetric encryption key predetermined by some other scheme.

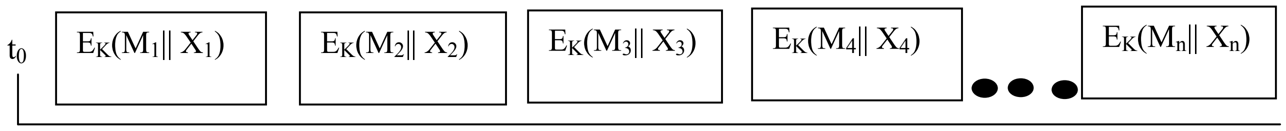

Figure 6: Hash-based Encryption Scheme

If a node misses a message due to some type of network obstruction, it can still authenticate the next message. Instead of calculating one hash, it would simply calculate two to reach the stored value. Motes may be able to tell if they have missed messages and so know how many additional hashes to calculate. Otherwise, a set amount of hashes to keep trying to authenticate a message would need to be established to avoid motes continuously calculating hashes in an attempt to authenticate an invalid message.

The number *n* in this protocol should be choosen high enough so that it does not exceed the number of broadcasts the base station needs to make during the lifetime of the sensor network.

This protocol does not provide for authentication of messages sent from other motes. Using one version of the J-secure key distribution scheme, motes would only be able to communicate directly with motes in their own group. They would be able to authenticate these messages because only members of the group and the base station know the group key. Furthermore, authentication of messages from motes is less important than that of the base station, since unlike the base station, other motes are not trusted.

5. Symmetric Encryption Schemes

Whatever key distribution and authentication schemes are selected, a symmetric cipher needs to be employed for message encryption in the DSN. Several symmetric key encryption schemes were analyzed based on the feasibility of implementing them on DSN devices, the security they provide, and the speed of encryption and decryption.

5.1 DES

DES (Data Encryption Standard) is a 16 round, 64-bit block Feistel cipher and uses a 56-bit key. During each round of DES, the F-function is run on half of the 64 bits, expanding and permutating the 32 bits into 48 bits, then calculating the XOR of the bits with the key, substituting with S-boxes to get 32 bits, and finally performing another permutation. The brute force key search space for DES is 2^{56}. Though this may seem large, a brute force, distributed attack has been used to break DES in 22 hours [3]. However, in the mote environment the battery life of motes may be less than this, making DES still a feasible alternative. However, as processing speeds for computers increase, the time needed to break DES will likely decrease, meaning that the security of this algorithm will also decrease with time, preventing it from being a long-term solution to security in DSNs.

With the security of DES being questioned, Triple DES is used in many circumstances requiring high security. Triple DES uses two 56bit keys, encrypting the message with the first key, then decrypting with the second, and finally encrypting with the first again. This gives a brute search key space of 2^{112}. However, it triples the amount of work the processor must do in comparison to DES, making the algorithm very slow, computationally expensive, and unpractical for use in the DSN environment.

5.2 Rijndael

The Rijndael encryption algorithm was recently chosen as the Advanced Encryption Standard (AES). Rijndael is a 128-bit block symmetric cipher that has from 10-14 rounds and a key size ranging from 128 to 256 bits. For a network of motes, the 128-bit key, 10 round version is probably appropriate for most applications as this version provides the fastest implementation and least memory requirements. Also, the additional security of larger keys is likely not needed due to the short battery life of sensor network devices. During a round, each 8-bit byte is first replaced by its substitute in an S-box. Next, rows are shifted over four different offsets with Row 0 being unmoved, Row 1 rotated left 1 byte, Row 2 rotated left 2 bytes, and Row 3 rotated left 3 bytes. Then the bytes in the columns are linearly combined using matrix multiplication, and finally the round subkey is XORed in for the current round [6].

There exists a known academic attack for breaking 9 of the 10 rounds of Rijndael [17]. However, no attacks have succeeded in showing that all 10 rounds can be broken, and the cipher's selection as the AES standard shows confidence that it will not be broken in the immediate future. Furthermore, the short-battery life of DSN devices means there is very little time for a cipher to be broken.

5.3 Twofish

The Twofish algorithm was another AES candidate. It is a 16-round Feistel network cipher that uses 128-bit blocks, and supports a key size of up to 256 bits. During a round of Twofish, two 32-bit words are input into the F function with each word being further divided into four bytes. Each byte is then sent through a different key-dependent S-box with the four output bytes being combined into a 32-bit word using a Maximum Distance Separable (MDS) matrix. Next the two 32-bit words are combined using a Pseudo-Hadamard Transform (PHT). They are then added to two round subkeys and XORed with the other half of the text. Also, two 1-bit rotations occur, one before and the other after the XOR. In addition to the encryption during each round, before the first round and after the last "prewhitening" and "postwhitening" take place where additional subkeys are XORed into the text block [16].

The Twofish algorithm provides some interesting tradeoffs between speed and required memory storage. The table below summarizes these tradeoffs in an 8-bit smart card 6805 processor. Though this processor is still more powerful than that which will likely be in the motes, it provides a good example of performance tradeoffs in an 8-bit processor [16].

Code and Table Size	Clocks per Block
2270	29100
2150	32900
2000	35000
1750	37100

Table 2: Twofish Tradeoff

The code and table size memory requirements include code for decryption and encryption. The clocks are for key set-up and either encryption or decryption.

5.4 RC5 and RC6

RC5 is another symmetric block cipher. It allows a variable key size and number of rounds and relies heavily on the use of data-dependent rotations for security. Before beginning the rounds, the block is first split in two and a subkey is added to each part. During each round, each part is XORed with the other, shifted by the other, and then added to a unique subkey for that half of the round. The pseudo-code for the encryption during each round is as follows [8]:

```
$A = [(A \otimes B) << B] + S(2i)$
$B = [(B \otimes A) << A] + S(2i + 1)$
```

During an RSA challenge, a distributed network attack was able to break an implementation of RC5 with a 56-bit key in 250 days [15]. The attack illustrates that there are some security weaknesses in RC5, especially when used with a smaller key.

However, it does not show that it would be possible to break RC5 in a short period time, and thus it is still a fairly practical algorithm for use in a DSN where there is a very limited network lifetime.

RC6 is an adopted version of RC5 to meet the requirements for AES. The main changes were to use four working registers instead of two and to employ 32-bit integer multiplication in the encryption process, increasing the amount of encryption per round and decreasing the needed amount of rounds [9]. However, for the 8-bit processors that would likely be found in motes, the 32-bit multiplication greatly decreases the speed and practicality for this platform.

5.5 TEA

TEA, which stands for Tiny Encryption Algorithm, was designed to be a simple algorithm that would require minimal memory space and would derive its security from a large number of rounds rather than the complexity of the algorithm. TEA uses a 128-bit key and has 32 cycles each of which includes 2 rounds. Diffusion during encryption is complete after 6 rounds (3 cycles). Examples of implementations of the TEA encryption and decryption algorithms are shown below:

Encryption Algorithm:
```
void code(long* v, long* k)  {
unsigned long y=v[0],z=v[1], sum=0,   /* set up */
 delta=0x9e3779b9, n=32 ;             /* a key schedule constant */
while (n-->0) {                       /* basic cycle start */
  sum += delta ;
    y += (z<<4)+k[0] ^ z+sum ^ (z>>5)+k[1] ;
    z += (y<<4)+k[2] ^ y+sum ^ (y>>5)+k[3] ;   /* end cycle */
```

Decryption algorithm:
```
void decode(long* v,long* k)  {
 unsigned long n=32, sum, y=v[0], z=v[1],
 delta=0x9e3779b9 ;
sum=delta<<5 ;
                       /* start cycle */
while (n-->0) {
    z-= (y<<4)+k[2] ^ y+sum ^ (y>>5)+k[3] ;
    y-= (z<<4)+k[0] ^ z+sum ^ (z>>5)+k[1] ;
   sum-=delta ;  }
                       /* end cycle */
v[0]=y ; v[1]=z ;  }
```

TEA has small memory requirements when compared to other algorithms, and its authors claim it is three times as fast as DES [19]. The main problem with TEA is that its security is still relatively untested. One attack has been published that claims to be able to break TEA with 2^{23} chosen plaintexts and one-related key query [4]. The attack demonstrates that there are potential weaknesses in TEA due to its simplicity, but does not demonstrate a practical way of breaking the cipher, especially in a DSN where both the number of messages sent and the time frame are limited.

5.6 Performance and Security Comparison

Several groups have analyzed the performance of the various AES candidates in different platforms. Such analyses provide some basis for comparison. The following table shows a comparison of memory and computation requirements for the Motorola 6805 8-bit processor [4]:

Cipher	Ram (bytes)		Rom (bytes)			Time(cycles)	
	Scheduled Key	Encrypt	Schedule	Encrypt	Encrypt + Schedule	Encrypt	Schedule
DES	96	21	19	680	1036	17458	12320
Rijndael	16	34	0	879	879	9464	0
RC6	56	55	38	1342	1374	105987	78808
Twofish	24	36	?	?	2200	26500	1750

Table 3: Encryption Algorithm Comparison

The chart shows that the Rijndael algorithm requires both less memory and less clock cycles than the other algorithms. However, Rijndael also has a lower safety factor than both RC6 and Twofish in that academic attacks have been made on a greater percentage of its rounds. However, since the DSN will only be operating for a short period of time, the performance advantage seems worth any potential decrease in security.

A comparison between Rijndael, RC5, and TEA is more difficult as less work has been done comparing these algorithms to each other. If TEA is three times as fast as DES, then it would provide comparable encryption speed to Rijndael, and possibly faster decryption speed. It also has potential to require less memory. However, the lack of sufficient analysis of TEA in the literature makes its security questionable, making Rijndael a better choice at the current time for high security endeavors. TEA should continue to be evaluated and considered as an option.

When choosing an encryption algorithm for the SPINS protocol for security in sensor networks, RC5 was selected over Rijndael. The authors of the protocol claimed that faster versions of Rijndael required 10KB of memory for tables and that RC5 was significantly faster than the smaller implementation of Rijndael. The group also claimed that the RC5 algorithm required 1.6KB of memory on an Atmel 8-bit processor and 80 bytes of RAM. Assuming that the memory required in the DSN platform would be similar to that in the 6805 Motorola processor for which Rijndael was evaluated, RC5 requires about 30 more bytes of RAM and 300 less bytes of memory than the code for both encryption and decryption in Rijndael. This gives comparable memory requirements for the two described implementations. Based on the claim of the SPINS group that RC5 is indeed faster, it is recommended as the symmetric cipher to be used in DSNs. However, the performance of RC5 and Rijndael in the specific mote environment should continue to be evaluated.

6. Security Protocol for Sensor Networks (SPINS)

The field of security for sensor networks is very much in its infancy. In our research we came across only one algorithm written specifically for DSNs. This algorithm is called SPINS and is explained and analyzed in full detail here.

Researchers at the University of California Berkeley are working on security in sensor networks [7]. The performance capabilities of their devices are very similar to our model:

CPU	8-bit 4 MHz
Storage	8 KB instruction flash 512 bytes RAM 512 bytes EEPROM
Communication	916 MHz Radio
Bandwidth	10 Kilobits per second
Operating System	TinyOS
OS code space	3500 bytes
Available code space	4500 bytes

Table 4: SPINS Hardware Configuration

These devices like our hypothetical ones do not have the capability to implement asymmetric encryption schemes. Thus, it is necessary to have protocols without relying on these mechanisms.

6.1 SPINS Requirements

The SPINS paper [7] specifies the following requirements for any sensor network.

Confidentiality: Eavesdroppers should not be able to determine what information the sensors are sending back to the base station.

Authentication: It should not be possible for an adversary to spoof messages as though they were coming from either nodes or the base station.

Integrity: Authentication usually implies integrity since it will be difficult for an adversary to modify a signed message, and maintain the signature while modifying the data. It can also mean that if the message is modified by "noise," the signature is still intact.

Freshness: It should be clear to the base station or the nodes what messages are current, and which are old.

6.2 SPINS Assumptions

SPINS [7] makes several different assumptions about the sensor network that are similar to our assumptions for this paper:

1. Communications fall into three different categories:
- Node to base station.
- Base station to node.
- Base station broadcasting to all nodes.

For the first two, SNEP is used, and for the third, they use μTESLA.
Note that it is possible to send a message from one node to another node, but this would involve hopping through the base station.

2. All nodes trust the base station.
This is a reasonable assumption since the base station is at a secure location.

3. The nodes mutually mistrust each other.
It is important that we make this assumption since if a node is compromised then its loss would hopefully not compromise the whole network.

4. Each node M_i has a master key K_i which it shares with only the base station.
This master key can be installed when the node is created, or prior to deploying it in the network.

The two protocols used in a SPINS sensor network: SNEP and μTESLA will be described next.

6.3 Sensor Network Encryption Protocol (SNEP): Overview

All unicast communications either from the base station to a node or from a node to the base station rely on SNEP. In SNEP, each node has a different key, which it shares with the base station. This provides both confidentiality and authentication.

To ensure freshness each node also maintains a counter with the base station. It uses this counter as the initialization vector for the stream cipher used, which is RC5 [8]. Whenever a message is sent from either the base station to a node *i* or from a node *i* to the base station, the i^{th} counter is incremented. Thus, the node can know the order in which messages arrived from the base station. It is conceivable that another cipher could be used (i.e. TREYFER [21] or TEA [20]).

If some messages are lost, then the node can try decrypting with counter values around the current counter value it has stored. If it cannot decrypt the message, then it will need to resynchronize the counter with the base station.

6.4 SNEP: Details

In each node two keys are generated from the key shared with the base station. That is if node *i* has key K_i it shares with the base station, then it generates and sends $K_{i,encr}$ and $K_{i,mac}$ to the base station. $K_{i,encr}$ is used for confidentiality, $K_{i,mac}$ is used for message authentication.

For authentication, a message authentication code or MAC is used. Since node *i* only shares its particular $K_{i,mac}$ with the base station, the base station knows that any message encrypted with that particular $K_{i,mac}$ must have originated from node *i*. Likewise in the case that node *i* is receiving a message encrypted with $K_{i,mac}$, node *i* knows that it must have originated from the base station. In the case of SPINS, in order to save memory space on the nodes, the MAC is just the same as the encryption algorithm: RC5.

A message sent from either the base station to node *i* or from node *i* to the base station consists of the following:

$A \rightarrow B$: $\{D\}_{<Kencr,C>}$, $MAC(K_{mac}, \{D\}_{<Kencr, C>})$

The data is sent encrypted with K_{encr} using as an initialization vector, the counter C. The MAC is computed using key K_{mac} and plaintext $\{D\}_{<Kencr, C>.}$

Although it is possible to know which messages came in which order from either the base station or node *i*, it is not possible to know whether a message originated in response to a specific request from the base station. To enable this, the base station sends a random bit string or nonce with each request it sends to node *i*. When node *i* is responding to that request, it includes the random bit string in its response so the base station knows that it is responding to that request.

6.5 μTESLA: Authenticated Broadcast

Authenticated broadcasts cannot be done with SNEP because if every node getting the broadcast knows the same key, then any node could masquerade as the base station
(one of the assumptions is mutual mistrust between nodes)

The problem would be easy if the base station could securely distribute its public key and then just sign all its messages with its private key using a public key cryptosystem like RSA [11]. The problem is that asymmetric cryptography is too computationally intensive for the devices.

An asymmetric mechanism is still needed. The broadcaster, must be distinct from the broadcastees, otherwise any of the broadcastees could masquerade as the broadcaster. Another possibility would be to simply send out each message using SNEP to all the nodes. However, this would be prohibitively computationally expensive for the base station. An efficient mechanism can be found under the constraints by using a one-way crytographic hash like MD5 [10].

6.6 *µTESLA: Details.*

To do the µTESLA protocol the base station must calculate a key chain. That is, it generates some key K_n, and then using a one way hash function F, it calculates $F(K_n) = K_{n-1}$, and $F(K_{n-1}) = K_{n-2}$ ….. $F(K_1) = K_0$. It sends to all the nodes K_0 (it can use SNEP to do this).

All the nodes are time synchronized with the base station. The protocol is that at periodic intervals the keys K_1, K_2, …. K_n are broadcast. Say at t_1, K_1 is revealed, at t_2, K_2 is revealed, and at t_i, K_i is revealed. In the span t_0 to t_1 all packets broadcast by the base station are encrypted with K_1. The nodes can not decrypt it because they do not know K_1. However, once K_1 is revealed they can be certain it came from the base station, because only the base station knows K_1 and they can verify that $F(K_1) = K_0$. If any imposter broadcasts a message m_i encrypted with key K'_i and then broadcasts the key K'_i the nodes would know that it is an imposter because $F(K'_i)$ would not equal K_{i-1}. See figure 7:

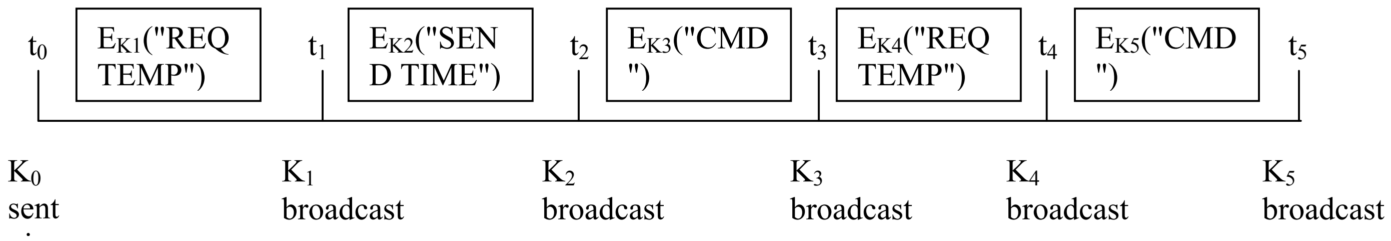

Figure7: µTesla

7. Conclusion

The distributed sensor network environment is very resource constrained, making adaptation of existing security protocols and development of good new protocols a challenging endeavor. Sensor devices, such as the motes developed at Berkeley, have small 8-bit processors, limited memory, and limited power. All communication in DSNs is wireless, making it implicitly insecure to both passive and active eavesdroppers. Furthermore, security is important to many DSN applications for military, practical, and privacy reasons. Therefore, both encryption and authentication methods that meet the constraints of this environment and provide a high level of security are needed.

During this project a variety of existing protocols for key distribution, authentication, and encryption were analyzed on their ability to be adapted to the DSN environment. Most of the algorithms analyzed did not work well in a DSN environment and are not described here. In addition, one protocol, SPINS, was found that was specifically developed for the sensor network environment. Analysis of this protocol showed that it provided some viable security solutions for sensor networks, but that as a whole did not seem to be optimized for or meet all the constraints for the environment we had defined. As a result many of its ideas were incorporated with other existing security protocols to develop a recommendation for a security protocol for the DSN environment.

Based on the analysis of the various security methods presented in this paper, we recommend a combination of protocols that covers the three main security concerns for DSNs of key distribution, authentication, and encryption. For key distribution the modified J-secure protocol (see sections 3.2.3 and 3.3.1 for complete description) appears to provide the best solution. The J-secure protocol divides the nodes into groups where all nodes in the same group have the same key, all other groups have different keys, and the base station knows all keys. This protocol provides a compromise between the efficiency of all nodes having the same key and the security of all nodes having different keys and allows customization of the group size according to security needs. The modified version groups nodes after deployment, enabling nodes to be grouped in a more optimized manner while expending minimal additional energy in the grouping process. This extra step of allowing groups to be established after deployment may not be needed in all situations, such as those where there are natural groupings or the location of nodes is known before deployment.

For authentication of broadcast messages by the base station, the hash-based protocol described in section 4 is recommended. This protocol allows motes to authenticate that a message came from the base station and has the expense of the additional bits needed to send a key with each message and the computation required to calculate the hash of the key. This communication and computation is fairly minimal, while the scheme provides a good level of authentication.

Finally, various symmetric encryption algorithms were analyzed (see section 5). The initial analysis showed RC5 to be the best choice. However, further investigation should be conducted on the exact memory requirements and processing speed of Rijndael on these devices in comparison to that of RC5. Also, the security of TEA should continue to be evaluated, and if significant analysis shows this scheme to be secure, it may be a better choice for encryption than RC5.

It is believed that the combination of key distribution using the modified J-secure scheme, authentication with the hash-based authentication scheme, and encryption with RC5 provides a fair level of security for DSNs with rather low required amounts of communication, memory, and computation. However, these results are in no way conclusive or optimal, as security in DSNs is still a relatively new area of research and should continue to be extensively investigated.

This project focused mostly on examining existing security methods and attempting to apply them to sensor networks. There exists an interesting tradeoff between adapting existing methods to a new environment versus developing methods specifically for that environment. Existing methods are tested and have proven security; however, they may not be the optimal solution for the new environment. This latter case is especially true in environments that differ radically from that for which the protocol was designed. DSNs differ significantly from most current applications of security protocols due to the extensive constraints. As a result, the optimal security solution for DSNs will likely not be found in existing protocols. We recommend that new protocols be sought that work for widely distributed systems with communication restrictions.

The DSN environment is still a very new field and will continue to develop rapidly in the coming years. Such developments will likely alter both the security needs and constraints of the environment. Ultimately multiple schemes of security will become prevalent in DSNs, where the choice of a particular scheme will be driven by the application of use.

8. References

1. Blundo, C., et al. "Perfectly secure key distribution for dynamic conferences," Advances in Cryptology:Proceedings of Crypto92, E.F.Brickell, ed., LNCS740, Springer-Verlag(1992), 471-486.

2. Carman, David W., Peter S. Kruus, and Brian J. Matt. NAI Labs Technical Report # 00-010 "Constraints and Approaches For Distributed Sensor Network Security." September 1, 2000. The Security Reseach Division NAI Inc. (c) 2000

3. Evans, David. "Lecture 4 Notes." Computer Science 588, University of Virginia, Fall 2001.

4. Keating, Geoffrey. "Performance Analysis of AES Candidates on the 6805 Core." 15 April 1995. http://members.ozemail.com.au/~geoffk/aes-6805/paper.pdf

5. Kelsey, John; Bruce Schneier, and David Wagner. " Related-Key Cryptanalysis of 3- WAY, Biham-DES,CAST, DES-X, NewDES, RC2, and TEA" http://www.counterpane.com/related-key_cryptanalysis.pdf

6. Merrion, Sarah. "Rijndael the Future of Encryption." 27 November 2000. *Sans Institute webpage.* http://www.sans.org/infosecFAQ/encryption/rijndael.htm

7. Perrig, Adrian, Robert Szewczyk, Victor Wen, David Culler, and Doug Tygar. "SPINS: Security Protocols for Sensor Networks." In *Proceedings of Seventh Annual International Conference on Mobile Computing and Networks MOBICOM 2001*, July 2001.

8. Rivest R. L. "The RC5 encryption algorithm." *Proc. 1st Workshop on Fast Software Encryption*, pages 86-96, 1995.

9. Rivest, Ronald; M.J.B. Robshaw, R. Sidney, and Y.L. Yin. "The RC6 Block Cipher."

10. Rivest, Ronald L. "The MD5 message-digest algorithm." Internet Request for Comments, April 1992. RFC 1321.

11. Rivest, Ronald L., Adi Shamir, and Leonard M. Adleman. "A method for obtaining digital signatures and public-key cryptosystems." *Communications of the ACM*, 21(2):120-126, 1978.

12. RSA Security webpage. "Has DES been broken?" http://www.rsa.com/rsalabs/faq/3-2-2.html

13. RSA Security webpage. "What are RC5 and RC6?" http://www.rsa.com/rsalabs/faq/3-6-4.html

14. RSA Security webpage. "WAP Forum Specifies RSA Security's RC5™ Encryption for Wireless Environments." http://www.rsa.com/news/pr/000420-2.html

15. Seminerio, Maria. "Programmers Crack 56-bit RC5 Encryption." 28 October 1997. http://www.zdnet.com/zdnn/content/zdnn/1023/174826.html

16. Schneier, Bruce. "Block Encryption for the 21st Century." *Dr. Dobb's Journal*: December 1998. http://www.ddj.com/documents/s=913/ddj9812b/9812b.htm

17. Schneier, Bruce, John Kelsey, Doug Whiting, et al. "The Twofish Team's Final Comments on AES Selection." 15 May 2000.

18. "Smart Dust, Autonomous Sensing and Communication in a Cubic Millimeter." *Smart Dust Project webpage.* http://robotics.eecs.berkeley.edu/~pister/SmartDust/

19. Szewczyk, Robert, Adrian Perrig, and Victor Wen. "Security Protocols for Sensor Networks." Ninja retreat, 11 January 2001.

20. Wheeler, David and Roger Needman. "TEA a Tiny Encryption Algorithm." November 1994. http://www.ftp.cl.cam.ac.uk/ftp/papers/djw-rmn/djw-rmn-tea.html

21. Yuval, Gideon. "Reinventing the Travois: Encryption/MAC in 30 ROM bytes." In *Proc. 4th Workshop on Fast Software Encryption*, 1997.