Rochester Institute of Technology
Computer Science Department

Independent Study Report

A Pairwise Key Pre-distribution Scheme
on Wireless Sensor Networks using Sun SPOTs.

Supervisor: Prof. James M. Kwon

Submitted by: Mesut Arik

TABLE OF CONTENTS

1.INTRODUCTION 2
2.KEY DISTRIBUTION SCHEMES 2
2.1 Secret Sharing 2
2.2 (k, n)-Threshold Scheme 3
2.3 Blom's Scheme 4
2.3.1 λ-Security Property 5
2.4 Key Pre-Distribution Scheme by Wenliang Du et al. 5
2.4.1 Pre-Deployment 5
2.4.2 Post-Deployment 7

3. INTRODUCTION TO SUNSPOT SENSOR DEVICES 7
3.1 Hardware Characteristics 7
3.2 Standard Components In a Sun SPOT 8

4. IMPLEMENTATION 9
4.1 Development Environment 9
4.2 Base Station Program 9
4.3 Node Program 10
4.3.1 Neighbor Discovery 10
4.3.2 Accessing Flash Memory To Store Persistent Data 11
4.3.3 Structure of the Initialization String 14
4.3.4 Structure of a Matrix data 14

5. CONCLUSION & FUTURE WORK 16

REFERENCES 17

1. INTRODUCTION

Recent developments in the hardware manufacturing technology make it possible to build even smaller and more efficient devices. This is one of the reasons why wireless sensor networks is a promising area of research. One of the many advantages of wireless sensor networks is that deployment of the network is extremely flexible. Therefore, wireless sensor networks can be used in a very wide spectrum of applications. Another big advantage is that sensor nodes can perform in-network processing, like data aggregation between several nodes.

On the other hand, these advantages bring some security threats as well. Deploying nodes on almost any kind of terrain in any condition makes them susceptible to node capture and physical tampering. And in-network data processing and aggregation lets adversaries capture data more easily. There are also many other types of attacks, which can compromise the confidentiality of the information or authentication of the nodes.

Since variety of attacks do exist against wireless sensor networks, it is very clear that we need a secure and efficient key distribution or pre-distribution mechanism that allows simple key establishment for large- scale sensor networks.

2. KEY DISTRIBUTION SCHEMES

In order to understand the key pre-distribution scheme we use, some background knowledge is required. Next sections explain various different key sharing schemes.

2.1 Secret Sharing

Secret sharing refers to the method of distributing a *secret* between a group of nodes. Each node is assigned a part, or *share*, of secret. Secret is constructed in such a way that individual parts of it are not usable by any of the nodes by themselves. Secret is only useful when it is reconstructed. And to reconstruct it, all of the individual pieces scattered between the nodes must be collected.

Therefore, according to the description above, we can summarize the Secret Sharing scheme in two rules;

Given a secret S, we would like N parties to share the secret so that the following properties hold:

1. All N parties can get together and recover S.
2. Less than N parties cannot recover S.

Although a single node can not reconstruct the secret S without having less than N pieces of it, it is very likely to construct an S' that is close to the value of S without having all N parts of the original secret S. This property of the method introduces the partial information disclosure problem. A share, or part, may not contain all the information about the S, but it usually does disclose some information about it. So, any node, or an adversary can use parts of the secret to calculate an estimate of the secret S. This is not a desirable behavior although the scheme still satisfies the two rules mentioned above.

In order to solve the partial information disclosure problem, there needs to be a method to generate N shares from the secret S in such a way that no information about S will be disclosed from less than N shares. Using such a method, above two rules will still be true, and partial information disclosure problem will be eliminated. To explain this method better, following example shows how to share a 4 bit binary string secret between two parties without partial information disclosure. Suppose that the secret key S is 1010. To share S between two nodes, first we create a random 4 bit binary string which will be given to node #1, let's say this random string is 1001. For calculating the second share, we do XOR S and first string. So the second binary string would be 1010 XOR 1001 = 0011, and would be given to the node #2. Using these values, it is not possible to retrieve any information about the secret from a single share. The only way to reconstruct S is to have both of the shares and XOR them, i.e 1001 XOR 0011 = 1010. Although this is the simplest example of solving a partial information disclosure problem, it makes the concept easier to understand for cases where more than 2 shares needed. To generate more than 2 shares from a secret, which is what we usually need, similar methods can be used, as long as the method satisfy the two rules plus a third rule;

3. Individual shares or combinations of less then N shares can not be used for disclose information about S.

2.2 (k, n)-Threshold Scheme

Even though the method discussed previously solves the partial information disclosure problem, it is not suitable to use in most real life scenarios, since it requires to having all shares in order to reconstruct the secret S. In a situation where you want *k* (where k <= N) nodes, or shares, to be able to generate the secret, another method must be used. That is called *(k, n)*-threshold Scheme or Shamir's Secret Sharing [8].

Shamir's scheme describes how to divide data D into *n* pieces in such a way that D is reconstructable from any k pieces. At the same time, it is not possible to reveal any information about D from any k-1 or less pieces. This scheme provides both confidentiality and availability to the peers.

Threshold variable k can be chosen depending on the number of nodes or shares, and it represents the secrecy of the key S. Choosing the value of k is a trade-off between secrecy and reliability, therefore it makes this scheme very flexible to use.

Adi Shamir describes the areas in which this scheme can be used best as; “Threshold schemes are ideally suited to applications in which a group of mutually suspicious individuals with conflicting interests must cooperate.”[8]. This description highly resembles the organization and behavior of the nodes in wireless sensor networks. That is why, this idea constitutes the basis for both Blom's Scheme and Du et al's Scheme.

2.3 Blom's Scheme

Blom's scheme is a symmetric key generation system developed by Rolf Blom [1]. In fact, this scheme was not developed essentially for wireless sensor networks, but it is very flexible and various derivatives of the scheme are being used in many areas of technology, including wireless sensor networks. Blom's scheme provides a technique for calculating a unique key for each pair of users (nodes), which enables them to encipher/decipher messages exchanged. Therefore, messages are protected using symmetric key cryptography.

The simplest idea for distributing cryptographic keys would be storing the keys in the user devices. That is, for a network of N nodes, every node must carry N-1 keys in order to communicate with all other nodes in the network securely. Although this scheme is simple, it brings two very serious problems with it. First, flexibility of the network will be severely poor, because the number of nodes must be specified before deployment, and each node will only have keys to communicate with those N-1 initially specified nodes. Any additional nodes included to the network after the deployment phase will not be recognized by other nodes in the network since they do not have a symmetric key with the new node. Second, if number of nodes in the network N is too large, storing N-1 keys may not be possible in sensor nodes because of their extremely constrained memory size.

Blom's key pre-distribution scheme allows any pair of nodes to find a secret pairwise key between them and only uses $\lambda+1$ memory spaces with λ much smaller than N. It is based on secret sharing method discussed in the previous section.

2.3.1 λ-Security Property

The threshold λ is a security parameter in Blom's scheme. Selection of a larger λ leads to a more secure network. As long as no more than λ nodes are compromised, the network (uncompromised nodes) is perfectly secure, where λ is a pre-determined integer value in the range [1, N). When an adversary compromises more than λ nodes, all pairwise keys of the entire network will get compromised. However, λ also determines the amount of memory to store key information. Large λ value leads to higher memory usage.

2.4 A Pairwise Key Pre-distribution Scheme by W. Du et al.[2]

We user the pairwise key pre-distribution scheme, which was proposed by W. Du et al [2]. This scheme improves the resilience of the network while requiring less memory than other schemes. It is in fact based on Blom's Scheme, therefore, it exhibits all the benefits of that scheme such as λ-security property. And it also combines random key pre-distribution method [11] with Blom's scheme.

This scheme consists of two phases, Pre-deployment (or key pre-distribution) phase and post-deployment phase.

2.4.1 Pre-Deployment

During the pre-deployment phase, base station generates a Galois Field (or Finite Field), which will be used to generate Matrices with linearly independent columns.

Generate G Matrix

We generate a matrix G of size (λ+1) x N using numbers from the Galois Field GF(q), where N is the number of nodes and q is the smallest prime number larger than the key size; i.e 128 bit. Note that any λ + 1 columns of G must be linearly independent in order to achieve the λ-secure property.[2] An example G matrix can be seen in Figure 1. S here is a primitive element of GF(q), where each nonzero element in GF(q) can be represented by some power of s, namely S^i for any $0 < i \leq q - 1$.

$$G = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ s & s^2 & s^3 & \cdots & s^N \\ s^2 & (s^2)^2 & (s^3)^2 & \cdots & (s^N)^2 \\ & & & \vdots & \\ s^\lambda & (s^2)^\lambda & (s^3)^\lambda & \cdots & (s^N)^\lambda \end{bmatrix}$$

Figure 1. Matrix G

As there are N columns, each node will be given a specific column of this matrix G.

Generate Matrices D1*....D*ω

The most significant improvement proposed by W. Du et al.[2] over Blom's Scheme [1] is use of multiple key spaces and Random Key Pre-distribution method [11]. Although it increases the amount of memory needed, using this method makes it harder for adversaries to calculate the key spaces and find encryption keys, therefore it provides ***better resilience against node capture.***

Base station creates ω randomly created (λ+1)x(λ+1) symmetric matrices D_1....Dω over the Galois Field GF(q).

Calculate Matrices A_1....Aω

This scheme uses multiple key spaces to employ Random Key Pre-distribution method. Calculation of key spaces is done by using the matrices that are created in the previous two steps. We compute the matrices using the following formula;

$$A_i = (D_i \times G)^T$$

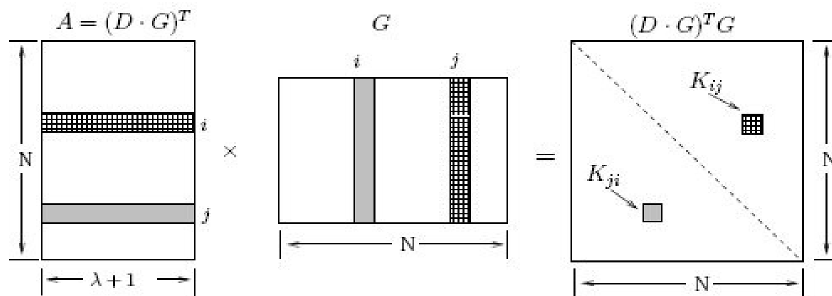

Figure 2. Calculation of Matrices A_1....*Aω*

Key Assignment

After all matrices are calculated, each node receives a column from matrix G, and one row from each of the τ random key spaces. Since we do the transfer of this data in a central(safe) office before the deployment, it is assumed that the data can be sent clear text over the air.

For example, j'th node will be assigned j'th column of G and j'th column of each of the τ random A matrices that were generated in the previous step.

2.4.2 Post-Deployment

We can deploy all sensor nodes once each node is assigned necessary information, i.e a column from G matrix, and a row of each of the τ random key spaces. After this point, if two nodes are neighbors and they want to calculate the encryption key to establish a secure connection, they can do so performing the following steps. Assuming two neighbor nodes, node i and node j, want to calculate an encryption key;

- Node i sends it's column of G, Gc(i), to Node j
- Node j sends it's column of G, Gc(j), to Node i
- Each node multiplies received column of G with their rows of τ rows of A.

As explained before, and can be seen in Figure 2, because of the matrix generated by multiplying A by G is symmetric, keys calculated by the tow nodes will be the same;

$$K_{ij} = K_{ji} = A(i) \times Gc(j) = A(j) \times Gc(i)$$

3. INTRODUCTION TO SUN SPOT SENSOR DEVICES

Sun SPOT (Small Programmable Object Technology) is an experimental platform to inspire developers to build wireless sensor networks applications using Sun Technologies. Sun SPOTs are the one of the most powerful wireless sensor network devices in the industry today. And it runs on other well-known and successful Sun Technologies, that is Java, which has been used on over 6 billion devices throughout the world.

A Sun SPOT Device is a small, wireless, battery powered sensor network node, which also contains several other components to sense the environmental and other factors that surrounds it.

3.1 Hardware Characteristics

This new technology is targeted for developers who are interested in development of wireless sensor networks (WSNs), robotics and other tiny devices running java applications. The SPOTs are powered by low-power battery and run Java 2 Platform, Micro Edition (Java ME). Also, for the SPOTs,

Sun Microsystems developers had applied a “sandwich” technology. All the SPOTS are built from almost the same basic elements: Battery, Main board, Daughter board (includes eDemo Board) and plastics keeping parts together. The main difference of various SPOTs is only in the combination of these parts.

3.2 Standard Components In a Sun SPOT

Battery

The battery is a 3.7 v 720 maH rechargeable lithium-ion prismatic cell. The battery has internal protection circuit to guard against over discharge, under voltage and overcharge conditions. The battery can be charged from either the USB type mini-B device connector or from an external source with a 5V ±10% supply. Typical shelf life losses at room temperature are about 2% of the batteries capacity per month and the rate can increase with the rise in temperature. Unfortunately, the battery cannot be changed for another type of battery. According to the SPOTS documentation, the charging and power management systems are tuned for this given battery [6]. Eventually, the life of battery is sufficient for multiple purposes. The time during which it can operate in deep sleep is 909 days and by maximum consumption of energy by all 8 LEDs it can run up to full 3 hours of work. [5]

Main Board

The eSPOT main board contains the: main processor, memory, power management circuit, 802.15.4 radio transceiver and antenna, battery connector and daughterboard connector. [5]

SPOT Daughter Board

According to the theory of operation, The SPOT daughter board should satisfy at least three conditions:

1. Connect to the eSPOT main board with a Hirose DF17-30 connector
2. Act as an SPI slave in communication with the eSPOT main board
3. Contain SPI flash memory for storing configuration information

One of the examples of SPOTs daughter boards is the eDemo Board

eDemo Board

The eDEMO board is provided as an example of daughter boards that are compatible with the eSPOT main board. The eDemo board has on it a 3-axis accelerometer, an ambient light sensor, eight tricolor LEDs, two push buttons, six analog input pads, four high current high voltage output pads, and five general I/O pads. [6]

For more information please see in the Sun SPOT theory of operation [6] and Sun SPOT Owner's Manual.

4. IMPLEMENTATION

Our implementation of the key pre-distribution scheme by W. Du et al. consists of two programs, one running on a computer that is connected to the base station and the other is running on the all nodes.

4.1 Development Environment

Since Sun SPOT is a MIDP(CLDC) device, it runs "MIDlet"s. MIDlet is a part of the naming style Sun Microsystems have been following for their Java language components, like Applet and Servlet. Although J2ME does not include many useful APIs that are in J2SE, programming language used is still Java. That is why writing MIDlets is not very different from J2SE programming for a fairly experienced Java programmer.

MIDlets are designed to run on small devices, like SPOTs, but they are developed on a regular laptop or desktop computer. Development cycle for MIDlets is a bit different than J2SE develop-compile-run cycle. While programming SPOTs the method I used was like; develop-compile-package-deploy-run. I used the development kit provided with the Sun SPOTs, which includes Netbeans IDE, J2ME plugin for netbeans and Sun SPOT J2ME libraries.

4.2 Base Station Program

During the pre-deployment phase, base station starts with constructing a $(\lambda + 1) \times N$ matrix G over a finite field GF(q). We use a laptop computer to calculate Galois Field, matrices G, D and A. Then, using Sun SPOT base station, we transmit data to each sensor node, so that they can be able to calculate keys between each other after deployment. This operation is done in central office, which is assumed to be safe, therefore we can send the data over the air in clear text.

Base station program is written in Java, using J2ME and J2SE libraries. It calculates Galois field GF(q), where q is the smallest prime number bigger than 128 bits, as we are generating 128 bit encryption keys. Then it generates matrices G and D_1.....D_w and all key spaces (A_1 to A_w). Once all matrices are generated, each sensor node will be given a column of G and row of each A matrix in the key space. This data is sent in clear text.

4.3 Node Program

Node program is a Java J2ME MIDlet. All free range nodes (nodes other than the base station) in the network runs this program. It manages neighbor discovery, neighbor table, key calculation and persistent storage (RecordStore).

4.3.1 Neighbor Discovery

In order for a node to calculate encryption key to establish a secure communication between a particular node and itself, it needs to know that the node is in the transmitting range. In wireless networks this is referred to as two nodes are neighbors.

I have implemented a basic neighbor discovery mechanism, that runs continuously along with the main thread of execution and discovers other SPOT devices within the transmission range. Discovery mechanism consists of two threads. First thread is responsible from periodically broadcasting “hello” packets to let other nodes in the range know of it's existence. Second thread listens for messages on a specific port and captures the hello requests from other nodes.

As it can also be seen in the flowchart in Figure 3, behavior of the listening thread can is as follows;

1. Wait for a “hello” message
2. Check if sender of the message is in the neighbors table, (using MAC address)
3. If sender is not in the neighbors table,
3.1 Create a new instance of the class Node
3.2 Set all fields with appropriate values from hello packet
3.3 Calculate encryption key
3.4 Add node object to the neighbors table, mapped by it's MAC address
4. If sender is in the neighbors table, do nothing
5. return to step 1.

Node class is used for representing neighboring classes. As a node discovers it's neighbors, it creates a node object for each neighbor.

```
class Node {

  String MAC;
  String EncryptionKey;
  int ID;
  int[] index;
}
```

After creating the Node object for it's neighbor, the node will put this object in the neighbor table, which is basically a Hashtable. Neighbors table holds Node objects and uses nodes' MAC addresses as keys.

```
Hashtable neighbors;
neighbors = new Hashtable();

private void addNeighbor(String address, String[] inTokens) {

   Node n = new Node();
   n.MAC = address;
   n.ID = Integer.parseInt(inTokens[1]);
   for (int i = 0; i < sizeA; i++) {
       n.index[i] = Integer.parseInt(inTokens[2+i]);
   }
   neighbors.put(address, n);
   n.EncryptionKey = calcEncKey(n);
}
```

4.3.2 Accessing Flash Memory To Store Persistent Data

SunSPOT API provides two mechanisms to store (read & write) data permanently on the flash memory. A SPOT node needs permanent memory facilities to store it's initialization string, so that it does not require the string to be transmitted from the base station after each reboot.

One way to access the flash memory is to is by using an object that implements the IFlashMemoryDevice interface. This interface provides access to the whole flash memory, which may be used by the Squawk VM and other system services. That is why, direct access to the memory using this interface is not recommended. Instead, as a safer method, streams should be used to do read/write operations on the available flash memory locations.

Example:

```
IFlashMemoryDevice flash = Spot.getInstance().getFlashMemoryDevice();
int startSector = flash.getFirstAvailableSector();
```

```
DataOutputStream dos = new DataOutputStream(mem.getOutputStream(startSector, 2));
dos.writeUTF("writing this to flash...");
```

Second, and I believe what is a better way to access the memory, is to use Record Management Store, which is a simple, record based mechanism. It basically consists of a collection of records stored in a uniquely-named RecordStore. A RecordStore is a class which provides basic functions such as, addRecord() and getRecord(). An object of RecordStore class must be created using a unique store name in order to do a read or write.Record store implementations ensure that all individual record store operations are atomic, synchronous, and serialized, so no corruption will occur with multiple accesses. [5]

Example:

```
RecordStore rms = RecordStore.openRecordStore("DATA", true);
if (rms.getNumRecords() != 0 )
String rms_str = new String(rms.getRecord(1));
```

Although nodes do not heavily use flash memory as they only do read/write once in every reboot, they must be able to read the initialization string from the RMS with no corruption. Therefore, I decided to use Record Management Store in the project, since it is a more reliable and intuitive way to store persistent data on flash memory.

When the program starts, it checks the RMS to see if there exists an initialization string from a probable previous execution. If the initialization string does not exist in the RecordStore, node goes in to the PreListening state and waits for an initialization string from the base station, assuming that it is in the central office. Once the node receives a valid initialization string from the base station, it creates a record in a RecordStore called "DATA" and writes the string in that record. This RecordStore and records in it will remain intact after reboots, unless another MIDlet suite is loaded in the SPOT or user wants to clear the RMS contents manually by pressing the Button2 on the SPOT. After the program reads an initialization string, exits the PreListening state and enters the Listening state, another thread for watching the Button2 starts with the Listening thread. Button2 is programmed in such a way that when it is pressed, device will clear all the records in its "DATA" RecordStore and restart the system, no matter what state it is in. The only exception to this is when the node is in PreListening state; the only running thread is PreListeningThread and no thread is running to watch Button2.

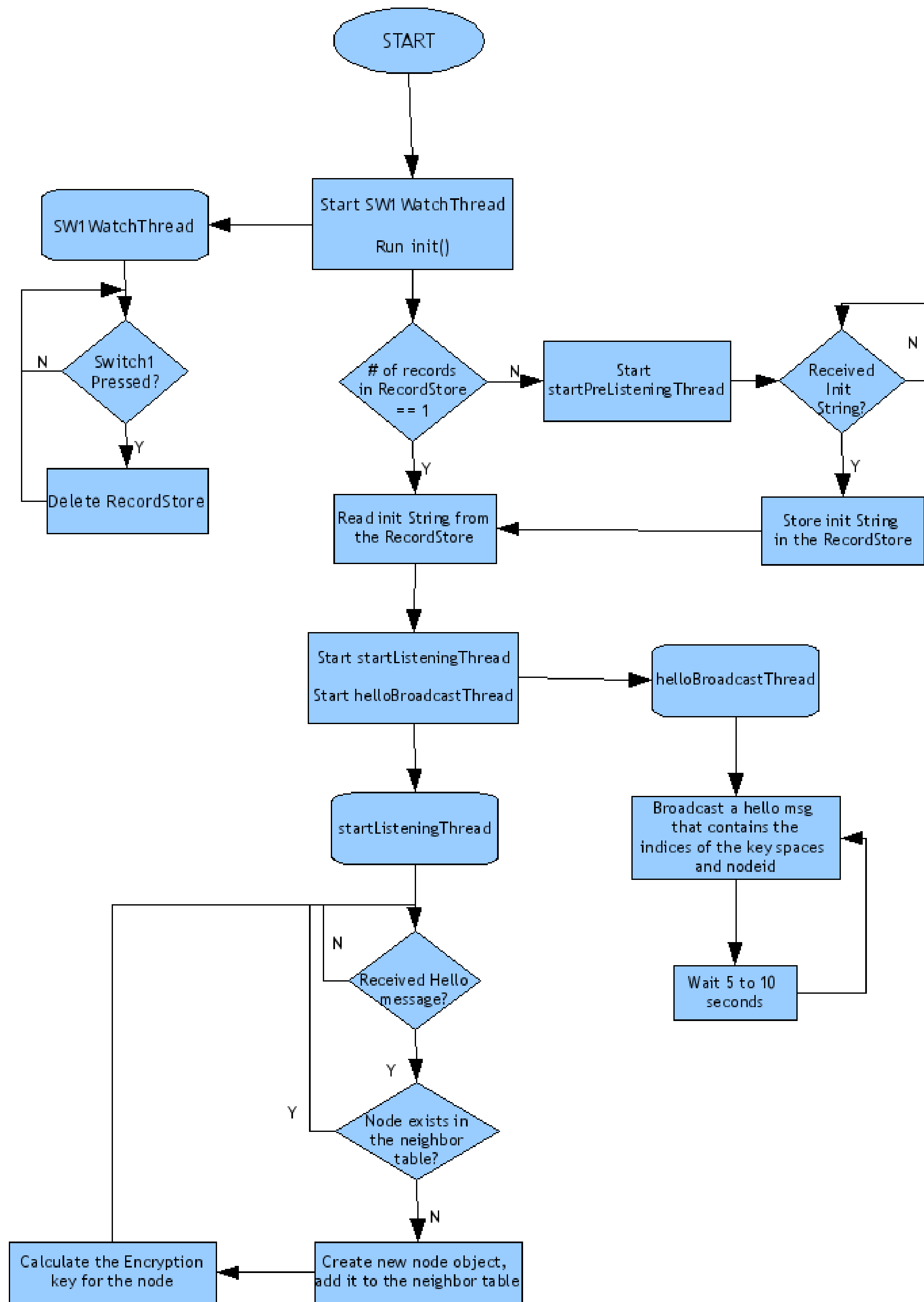

Figure 3. Flowchart of the node program

4.3.3 Structure of the Initialization String

Initialization strings are unique to each node and sent to nodes in the first phase of the key pre-deployment process, i.e in the central office. Initialization strings are stored in stored in the persistent storage, called Record Store, on the nodes. When a node starts up for the first time, it looks for this string in the Record Store, if the Record Store is empty, node goes in to PreListening mode and waits for the Initialization string from the base station. Each string is unicasted to each node using its MAC address.

"Init"	nodeId	S	λ	I_{Ai}	M_i	I_{Aj}	M_j	...

nodeId: Node number of the node sending the message.
S: Primitive element to generate the matrix G, (128 bit number).
λ: Lambda security value.
I_{Ai} :Index number of the key space (matrix) A_i
M_i : nodeId'th row of matrix A_i in serialized form. It is a 1x(λ+1) matrix.
I_{Aj} :Index number of the key space (matrix) A_j
M_j : nodeId'th row of matrix A_j in serialized form. It is a 1x(λ+1) matrix.

4.3.4 Structure of a Matrix data

Matrix is the fundamental data type that is used in the implemented key pre-distribution scheme. Implementation of the Matrix type is made in Matrix class. Since storing Matrices and passing Matrices over the network are two essential actions we do in this project, Matrix class needs to be serializable. The format of the serialized matrix is as follows.

height	width	q	num1	num2	...

height: number of rows in the matrix
width: number of columns in the matrix
q: prime number that is larger than 2^{128}
numX: element in the matrix. Position of the element is calculated by (i*j)+j

5. CONCLUSION & FUTURE WORK

In this project, we have experimented software development on Sun SPOT sensor devices, researched key distribution schemes in wireless sensor networks and developed working code for SPOTs that performs basic neighbor discovery operations and calculates cryptographic keys.

We believe that this is just an initial effort to investigate security aspects of wireless sensor networks and specifically of Sun SPOTs. Although our implementation of the scheme runs on Sun SPOTs without any problems, it can be further improved in many ways. Following is some of the ideas for future development;

1) Using the code we have already developed, create a framework that would run constantly on SPOTs and work as a service to encrypt/decrypt data. Therefore, applications running on top of the service do not have to include any code for the cryptographic functions, and just make use of our service.

2) Perform scalability tests on Sun SPOTS. So far we have developed the code and done basic tests with SPOTs. But have not run our tests on a network larger than 10 nodes. Since our program has not been tweaked for performance or optimizing memory used, running the program on a large number of nodes may introduce bugs or performance issues

3) Measure the amount of energy used by a particular node for calculating encryption key for a single node. That way we can find out how expensive it is to calculate a key comparing to other operations such as encryption/decryption and use of radio. It also allows us to do theoretical studies on the life of the network with a number of nodes and an estimated amount of traffic.

4) Integrate an encryption/decryption algorithm to the service/framework mentioned above. Bouncy Castle Cryptographic API [10] can be used to provide this capability. This API features implementations of many cryptographic algorithms such as AES, DES and RSA.

REFERENCES

[1] R. Blom. An optimal class of symmetric key generation systems. Advances in Cryptology: Proceedings of EUROCRYPT 84 (Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, eds.), Lecture Notes in Computer Science, Springer-Verlag, 209:335–338, 1985.

[2]W. Du , J. Deng , Y. S. Han , P. K. Varshney , J. Katz , A. Khalili, 2003. A pairwise key predistribution scheme for wireless sensor networks. ACM Transactions on Information and System Security

[3]Chan, H., Perrig, A., and Song, D. 2004. Key distribution techniques for sensor networks. In *Wireless Sensor Networks*, C. S. Raghavendra, K. M. Sivalingam, and T. Znati, Eds. Kluwer Academic Publishers, Norwell, MA, 277-303.

[4]Arik, M., Maketov, M., Nathan, K., Rajagopal, K. 2008. Sun SPOTs Investigation Report. Department of Computer Science, Rochester Institute of Technology.

[5] Sun™ Small Programmable Object Technology (Sun SPOT) Developers' Guide, 2007. Sun Microsystems.

[6] Sun™ Small Programmable Object Technology (Sun SPOT) Theory of Operation, 2006, Sun Microsystems.

[7] Rivest, R., Shamir. A., and Adleman, L. A method for obtaining digital signatures and public key cryptosystems. Comm. ACM 21, 2 (Feb. 1978), 120-126.

[8] Shamir, A. 1979. How to share a secret. *Commun. ACM* 22, 11 (Nov. 1979), 612-613. DOI= http://doi.acm.org/10.1145/359168.359176

[9] 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, November 2002.

[10] Bouncy Castle Cryptographic API, http://www.bouncycastle.org/

[11] Chan, H., Perrig, A., Song, D., Random Key Predistribution Schemes for Sensor Networks, Security and Privacy, 2003. Proceedings. 2003 Symposium on, May 2003