Security and Privacy in a Sensor-Based Search and Rescue System

Jyh-How Huang, John Black, and Shivakant Mishra

Department of Computer Science
University of Colorado, Campus Box 0430
Boulder, CO 80309-0430, USA

Abstract. With the emergence of small devices equipped with wireless communication, several sophisticated systems for search and rescue have been proposed and developed. However, a key obstacle in large deployment of these systems is vulnerability to users' security and privacy. On one hand, search and rescue systems need to collect as much information about a user's location and movement as possible to locate that user in a timely manner. On the other hand, this very capability can be misused by adversaries to stalk a person, which in turn drives users away from using such a system. This paper describes the design, implementation and performance of a security and privacy framework for SenSearch, which is a sensor-based search and rescue system for people in emergency situation in wilderness areas. This framework has been carefully built by employing a combination of symmetric and asymmetric key cryptography to meet the constraints of resource-limited devices and short time intervals during which most security operations have to be performed.

Keywords: sensor-based search & rescue, security & privacy.

1 Introduction

Search and rescue of people in emergency situation in a timely manner is an extremely important service. In the past, it was difficult to build such a service because of a lack of timely information needed to determine the current location of a person who may be in an emergency situation. However, with the emergence of small computing devices such as PDAs, sensors and cell phones with wireless communication capabilities, it has become feasible to build such a system. Indeed, several such systems have been proposed and prototypes of some of them have been implemented over the last five years [5,1,3,4,2].

We have designed and implemented a search and rescue system called SenSearch [6,9] for a wilderness environment[1]. A key differentiating feature of SenSearch is that it is designed for a wilderness environment. In such an environment, there is no Internet connectivity, no cellular network, and building an adhoc network from randomly-scattered mobile devices is infeasible due to an extremely sparse environment.

[1] First version of this search and rescue system was called CenWits.

J. Zheng et al. (Eds.): ADHOCNETS 2009, LNICST 28, pp. 176–191, 2010.

Since a search and rescue system like SenSearch must track the location and movement of people, there are some very obvious and important security and privacy concerns. In fact, such systems must cope with two conflicting issues. On one hand, the system requires collection of as much information about the location and movement of a person as possible. This is to ensure that a smaller and more accurate search area can be determined in case that person goes missing, or is in emergency situation. Indeed, it is in the interest of a person to give out as much information as possible about his/her location and movement to improve his/her chances of being located and rescued in case of emergency. On the other hand, a majority of people are not comfortable in giving out too much information about their location and movement for the fear that such information may be misused for malicious purposes, e.g. stalking. Indeed, this latter reason has proved to be a major hindrance in a wider deployment of SenSearch.

It is clear that appropriate security and privacy support must be provided in a search and rescue system such as SenSearch for wide acceptance. In general, a security and privacy framework for a search and rescue system must deal with three major challenges:

1. The framework must provide sufficient security and privacy guarantees to the users exploring a wilderness environment for recreation purposes, e.g. hikers, campers, or rock climbers.
2. The framework must not limit the system's ability to determine a small and accurate search area in a timely manner in case a user is in an emergency situation.
3. The framework must be implemented in a resource-constrained computing environment.

In this paper, we describe the design, implementation and evaluation of a security and privacy framework for SenSearch that addresses these challenges. We have engineered this framework from known techniques by carefully choosing a combination of symmetric and asymmetric key operations in face of resource-constrained devices and short time intervals during which most operations have to be performed. We have implemented and experimented with a prototype of this framework. Results show that the proposed framework is feasible and provides the required support for security and privacy.

This paper makes three important contributions. First, it identifies a reasonable threat model for security and privacy of a search and rescue system in a wilderness environment. This threat model includes an adversary's intent and the resources he is likely to possess. Second, it identifies important system constraints and engineers a solution with in these constraints from known security and privacy techniques. Finally, the paper demonstrates the feasibility of the proposed solution via a prototype implementation and evaluation.

2 SenSearch: A Brief Overview

SenSearch is a search and rescue system that makes use of smaller and cheaper sensor devices. It has several important advantages over the other search and

rescue systems. These advantages include a loosely-coupled system that relies only on intermittent network connectivity, power and storage efficiency, and low cost. It utilizes the concept of witnesses to propagate information, infer current possible location and speed of a user, and identify hot search and rescue areas in case of emergencies.

We explain SenSearch using a hiking application in a wilderness area. During a hike, each hiker determines his/her current location at regular intervals using a GPS receiver attached to his/her sensor node. This location is stored in a *witness record* along with user's ID, a timestamp, and a hop count (initialized to zero). The concept of witness works as follows. Whenever two hikers are with in a close range (say 50 meters) of one another, their sensors exchange all of their witness records with each other. Thus a witness record generated by a hiker's node is (redundantly) propagated to the other hiker nodes whenever hikers come with in close range of one another. Hop count in a witness record is incremented every time a record is propagated to the next node. In addition, whenever a hiker comes with in close range of an access point (static computing devices that have Internet connection to a control center), his/her node dumps all witness records to the access point. Information collected at a control center can be processed to reveal the time and location a missing hiker was last seen, and in what direction and speed he/she was moving. Furthermore, a relatively small search area can be inferred from this information in which search and rescue efforts may be focused.

A prototype of SenSearch has been implemented on MicaZ motes, running Mantis OS 1.0 beta. MicaZ is equipped with a 8-MHz, 8-bit Atmel ATmega128 CPU, 4 KB of RAM, and 128 KB of flash memory. We have performed a number of experiments, both indoors and outdoors. Figure 1 illustrates an example of the information collected at an AP during a hike in Mt. Sanitas in Boulder, CO. Seven hikers participated in this hike. The figure illustrates the location and time information of various hikers.

While the information illustrated in this figure is very useful for inferring a small search area for rescue efforts, the figure also illustrates a need for security and privacy in SenSearch. An adversary can easily get all this data by simply sitting at some strategic location in the hiking trail with a SenSearch node and collecting all witness records from hikers as they pass by. With this information, the adversary can infer when and where a particular hiker started his/her hike, what hiking trail he/she is on, at what speed and in which direction he/she is going, and where he/she is likely to be at present. Our goal in this paper is to build security and privacy support in SenSearch to prevent these adversaries from inferring such information.

3 Threat Model

We assume that the hikers hike in a typical wilderness environment such as a national forest for recreation. In this scenario, our goal is to protect a hiker

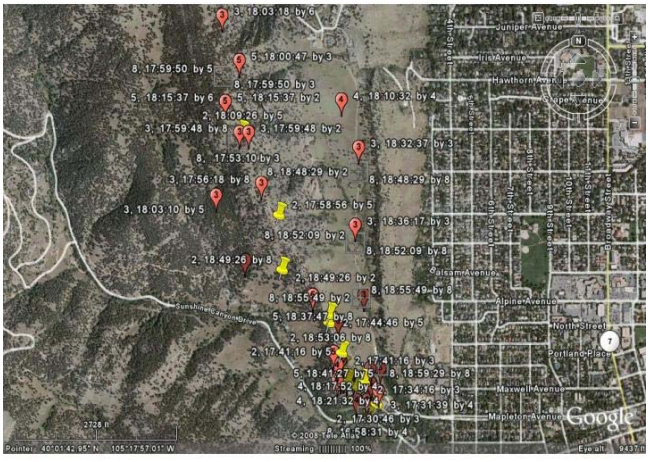

Fig. 1. Result from SenSearch deployment in Mt. Sanitas, Boulder

from being located or tracked (stalked) by adversaries. Our threat model is as follows:

1. An adversary's main goal is to locate and track some specific person in the wilderness environment without physically stalking that person.
2. An adversary doesn't attempt to mislead or thwart the search and rescue effort, e.g. he/she doesn't introduce any false information in the system, unless such information aids him/her in tracking a specific hiker.
3. An adversary doesn't attempt to launch any denial-of-service attack in the system.
4. An adversary may have unlimited storage capacity, and access to moderate computing resources, e.g. a high-end laptop to analyze the packets that he/she intercepts.
5. An adversary may have moderately strong antenna that can receive signals with in about one kilometer radius.
6. An access point (AP) is completely secure and robust, and an adversary cannot compromise an AP.

The key observation here is that in a wilderness environment being used for recreational purposes, we do not anticipate adversaries with determined intent to break down the search and rescue system, neither do we anticipate adversaries employing very expensive, powerful computing resources to locate and track a person. Protecting a system from such determined adversaries will require a much stronger security and privacy framework than presented in this paper.

4 Security and Privacy Framework

While the particular security and privacy problems in a search and rescue system are not fundamentally different from those in other networks, the environment in which such systems are deployed has a significant impact on the available solution space. Alongwith a dynamic, mostly unconnected network, such systems have to deal with limited interaction times and resource constrained devices. This makes efficiency in terms of minimizing time overhead and power consumption a necessary requirement in any security and privacy framework built for such systems.

4.1 System Constraints

There are two sources of constraints in designing a security and privacy framework for SenSearch. First, SenSearch nodes are resource constrained in terms of CPU, memory and power. Thus, the framework must employ only those cryptographic operations that require relatively low power, memory and computation. To this end, we have carefully chosen a combination of both symmetric and asymmetric key cryptography in our framework. In particular, it has been shown that symmetric key operations, both encryption and decryption using RC5 are feasible on sensor nodes, and consume relatively low power and memory [11,12]. However, only two asymmetric key operations, signature verification and encryption are feasible in sensor nodes such as MICA motes, while asymmetric key decryption and signature creation consume too much storage and compute power [14]. Based on this, we have designed our security and privacy framework that requires sensor nodes to perform symmetric key encryption and decryption, and a small number of asymmetric key encryption and signature verification.

The second source of constraints comes from the SenSearch system itself. An encounter between two SenSearch nodes (hikers) typically lasts only five to seven seconds[2]. Furthermore, in order to preserve power, SenSearch nodes send out beacons only at an interval of three seconds. This means that there may be as little as two seconds available for communication between two nodes after they detect each other. During this limited time interval, nodes need to exchange all of their witness records with each other. Clearly, our security frame work must not put too much time overhead during a node-node encounter. Similarly, a typical encounter between a SenSearch node and an AP lasts about ten seconds. With a beaconing time interval of three seconds, this means that there is as little as seven seconds available for message exchange during an AP-node encounter. A node needs to get all initial system-related information, including all security related information in this limited time interval. Also, a node needs to dump all its witness records to the AP in this limited time interval during a hike. Once again, this implies that our security frame work must not put too much time overhead during an AP-node encounter. To address these time constraints, we have chosen RSA over other forms of asymmetric key cryptography. It has

[2] Time intervals such as this and those mentioned later in this paper have been obtained from the data collected from a number of outdoor experiments we have done using SenSearch (see [9] for more details).

been shown that encryption and signature verification operations of RSA on sensor nodes are much faster than other forms of asymmetric key cryptography, e.g. ECC [8,10]. Another reason for choosing RSA is that there are many open source RSA libraries available. For symmetric key operations, we have chosen RC5, because it has been shown that RC5 provides strong security and can be efficiently implemented on sensor nodes [11,12].

4.2 Design Overview

Every SenSearch hiker registers with the SenSearch system to receive a SenSearch node. This registration can be done at the trailhead or in the visitor center of a park. To understand the security and privacy threat, we distinguish between an outsider adversary and an insider adversary. As the name suggests, an outsider adversary is not a part of the SenSearch system, i.e. he/she does not register with the SenSearch system and hence does not carry a registered SenSearch node. An insider adversary on the other hand registers with the SenSearch system and hence has access the all information that a normal registered hiker has, including a registered SenSearch node.

An outsider adversary can intercept and analyze messages exchanged between any two hikers, and between a hiker and an AP with in one KM radius. In addition, he/she can pretend to be a registered hiker, send fake messages, and get information from other hikers or APs. To prevent an outsider adversary from interacting with APs or registered hikers, we introduce mutual authentication mechanisms between an AP and a registered hiker, and between two registered hikers. Mutual authentication between an AP and a registered hiker involves both asymmetric and symmetric key cryptography, while mutual authentication between two registered hikers is done using symmetric key cryptography.

To prevent revealing any useful information from the intercepted messages by an insider or an outsider adversary, we introduce three security features. First, we assign dynamic IDs to the hikers, so that their real IDs are never included in messages they exchange. A mapping between real IDs and dynamic IDs is maintained at APs, which are assumed to be completely secure. A stronger solution to hide real ID of a hiker in a witness record would be to use a different ID for every record. These IDs can be chosen from a set of IDs preassigned by an AP to a hiker. However, the SenSearch system uses IDs to purge records from limited memory and decide which records to exchange first during a node-node encounter. This is feasible only if all records generated by a hiker contains the same ID. Our framework uses a variety of techniques to prevent an adversary from mapping a dynamic ID to a real ID.

Second, all information in a witness record except the dynamic ID, timestamp and hop count are encrypted using a symmetric key that a registered node shares with the AP. In particular, the location information is encrypted, so that an outsider adversary cannot locate a hiker, even if the dynamic ID of the hiker is leaked. The symmetric key used for this encryption is different for each registered node, so the encrypted information is secure even from other registered nodes, and hence from insider adversaries. This encryption is done as soon as a

witness record is generated by a node. So, all witness records stored in a node are encrypted this way, and are ready to be transmitted during a node-node or AP-node encounter, i.e. the cost of encryption is not incurred at the time of an encounter, which as mentioned earlier lasts for only a small time interval. Some information such as dynamic ID, timestamp and hop count are not encrypted, because nodes use this information to prioritize which records to transmit first and which records to transmit later during a limited contact time interval [9].

Finally, if there is sufficient time during a contact, we incorporate a third security feature in which a session key is created between two nodes during a contact, and this key is used to encrypt all unencrypted information in a witness record as well as to generate a MAC. The main limitation of this security feature is that the creation of a session key, encryption and MAC generation is done during a node-node encounter. So the time delay incurred due these operations is crucial as a typical node-node encounter time interval is relatively short.

With these security features, a hiker is prevented from being stalked by both insider and outsider adversaries. An outsider adversary cannot interact and get useful information from an AP or any hiker because of mutual authentication mechanisms. He/she can intercept messages exchanged between a hiker and an AP, or between two hikers. However, these messages do not reveal any useful information regarding the identity or location of a hiker. An insider adversary can interact with an AP or other hikers. However, because a witness record does not contain real ID and the location field is encrypted, he/she cannot determine the identity or location of a hiker. Furthermore, if a session key is used to encrypt messages, even the dynamic ID is not revealed to an outsider adversary, while an insider adversary will have to explicitly interact (mutually authenticate) with a hiker to get his/her dynamic ID.

Based on this design, the initial configuration of AP and SenSearch node is as follows:

1. Each AP has a public/private key pair and the public key is signed by a CA (Certificate Authority).
2. Each AP has Internet connection, and therefore can update its database regularly.
3. Each SenSearch node is pre-programmed with the CA's public key.
4. An AP shares two symmetric keys with each SenSearch node: AP_Node_SKey is used for encryption and AP_Node_MKey is used for generating MAC (Message Authentication Code).
5. Each SenSearch node shares separate keys (a pair of symmetric keys) with some (but perhaps not all) other SenSearch nodes.

4.3 AP-Node Authentication and Initial Setup

Initial (mutual) authentication between an AP and a SenSearch node is done in three steps as illustrated in Figure 2. Step one consists of a beacon that AP sends out at regular intervals advertising itself as an AP. This beacon includes a certificate containing the public key of the AP signed by the CA.

```
[I am AP] Beacon | AP Public Key Certificate
```

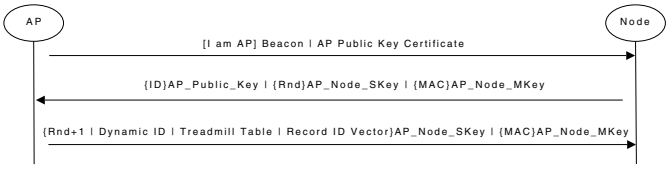

Fig. 2. Initial authentication and setup between a SenSearch node and an AP

When a SenSearch node receives this beacon, it first verifies the certificate, and and then replies with its real ID. The real ID of a SenSearch node identifies the owner of the node that is stored in the database at the AP. This database is confidential, and only authorized search and rescue authorities can access it. To keep the real ID confidential, the SenSearch node encrypts it using the public key of the AP. The SenSearch node also generates a random number and encrypts it with the symmetric key it shares with the AP. Finally, the SenSearch node calculate a MAC (Message Authentication Code) of the whole packet using the MAC symmetric key it shares with the AP.

```
Node Reply | {ID}AP_Public_Key |
{Rnd}AP_Node_SKey | {MAC}AP_Node_MKey
```

On receiving this reply, the AP first decrypts the node ID, and then retrieves the two symmetric keys it shares with that node from its database. The AP then checks the correctness of MAC that the node sent. A correct MAC authenticates the node to the AP, since only AP and the node know the two shared symmetric keys. After a successful authentication, the AP goes to the third step.

Since, asymmetric key encryption is significantly more computationally intensive than symmetric key encryption, step two can be altered as follows:

```
Node Reply | {ID|Rnd}AP_Node_SKey |
{MAC}AP_Node_MKey
```

Here the node ID is encrypted using the symmetric key shared between the node and the AP. However, when AP receives this reply, it doesn't know which symmetric key it should use for decryption as it doesn't know the node ID. Since an AP is power and computation rich, it can simply go through all the keys in its database to decrypt the ID and random number and verify the MAC. In the third step, the AP sends a dynamic ID, a *treadmill table*, and a *record ID* vector to the SenSearch node.

```
AP Reply | {Rnd+1|Dynamic ID|Treadmill Table|
Record ID vector}AP_Node_SKey | {MAC}AP_Node_MKey
```

Since the transmission media is radio, all nodes that are close to the AP will hear this messages. A node determines that this message is meant for it by verifying the MAC using the MAC symmetric key it shares with the AP. Once a SenSearch node validates the MAC, it verifies Rnd+1 and decrypts the dynamic ID. A

successful verification of Rnd+1 authenticates the AP to the node. Dynamic ID is different from the real ID of the node. The node uses this dynamic ID in all communication henceforth with other nodes in the wilderness area. Thus, exposure of real ID of a hiker is limited to only the initial setup, and that too in encrypted form.

Treadmill Table. When two hikers (nodes) meet in a wilderness area, they exchange their witness records. To ensure that a node transmits its witness records only to registered nodes and accepts witness records only from registered nodes, we need a mutual authentication mechanism between two nodes. In the absence of any third party that is connected with the two nodes at the time of contact and considering that asymmetric key decryption or signature creation are infeasible, this mutual authentication is feasible only if the nodes share a secret, e.g. a node may share a different secret key with every other node in the wilderness area. Given that there is only limited amount of memory in a SenSearch node and the number of hikers in a wilderness area is potentially very large, we need a mechanism by which a node shares such symmetric keys only with those nodes that it is likely to come in contact with during a hike. Treadmill table facilitates such a sharing. A separate treadmill table is constructed by an AP for each node n. It contains the symmetric keys that n shares with all those nodes that it is most likely to come in contact with. In addition, weightage is assigned to each node to indicate how important it is for n to exchange witness records with those nodes.

A treadmill table for a node n consists of two parts. The first part is a key table. The key table contains the symmetric keys that n shares with all those nodes that it is likely to come in contact with. In our implementation, a node shares two different keys, an encryption key (SKey) and a MAC key (MKey) with every other node that it is likely to come in contact with. These keys are used for implementing authentication, confidentiality, and message integrity. The size of a key table depends on the amount of memory available in the node. Figure 3 illustrates an example of key tables of two nodes, 8 and 10. This shows that node 8 will use key F for encryption and key 8 for generating or verifying MAC when it comes in contact with node 5, while node 10 will use key 5 for encryption and key 1 for generating or verifying MAC when it comes in contact with node 5. Notice that the keys that node 8 uses when it comes in contact with node 10 are same as the keys that node 10 uses when it comes in contact with node 8.

The second part of a treadmill is a list of weights, one for every node that n might come in contact with. This weight determines the importance of exchanging records when n comes in contact with other nodes. For example, if two nodes are moving in the same direction, they will keep detecting each other's presence frequently and may end up exchanging lots of witness records that are of little value. On the other hand, witness records exchanged during a contact between two nodes that started their hikes three hours apart and are moving in opposite directions are very useful. Weights provides a unique perspective to each node about how it should value other nodes' data and decide how many records it will accept from that node. Figure 4 illustrates an example of such a list in a

	0	1	2	3	4	5	6	7	9	10
Node 8	3	5	1	8	B	F	1	0	9	0
Key Table	A	1	5	7	2	8	4	B	A	2

	0	1	2	3	4	5	6	7	8	9
Node 10	A	9	3	2	1	5	B	F	0	C
Key Table	7	5	9	D	9	1	5	6	2	1

Fig. 3. Key table of nodes 8 and 10

treadmill table. The first row in this list shows the dynamic IDs of nodes that have departed for hike in the order of their departure times. For example, node 3 departed first, followed by node 62, followed by node 98, and so on. The blank entry indicates the departure order of node n. The curve indicates the weights. So, node n values encounters with 34, 55, 64 and 25 very highly, while it doesn't consider encounters with nodes 3, 2, 39 and 70 as important.

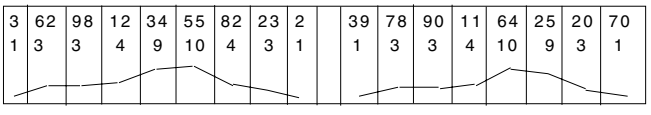

Fig. 4. Weight table of node n

An important utility of treadmill table is that it naturally promotes sharing of keys among a dynamic set of nodes, where old nodes (hikers that started their hikes much earlier) are retired (i.e. newer nodes do not share a key with them) and newer nodes are integrated into the system. It eliminates a need for pre-assigning keys to all nodes at once.

Record IDs. In SenSearch, each witness record is uniquely identified by a *record id*. Whenever a node n generates a new witness record, it assigns a unique ID to this record. So, we need an efficient mechanism that each node may use to generate globally unique record IDs. A straight-forward approach to do this is to use <node ID, local sequence number> as record IDs, where local sequence number is a locally unique number generated by the node. However, SenSearch also requires an efficient search operation: given a record ID, a node should be able to efficiently determine whether or not it has that record in its memory. This operation is needed to minimize the number of records exchanged during a node-node encounter. The straight-forward approach to generate globally unique IDs requires a node to perform a complete search over all records stored in its

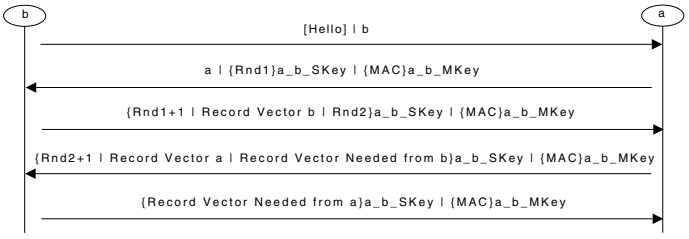

Fig. 5. Node-node authentication and record exchanges

memory to determine if a record with a particular record ID is available. This can be quite inefficient.

So, to generate globally unique record IDs and ensure efficient search operation, we pre-allocate a list of unique IDs to each node. This list is provided to the node during initial setup. When a node generates a witness record, it assigns a record ID chosen from this list. Since record IDs are used by nodes to determine which records to transmit, they are not encrypted. So, we need to ensure that an adversary cannot associate a range of record IDs to a particular dynamic node ID. To this end, the list of unique record IDs allocated to a node is selected randomly, i.e. it is not a consecutive sequence and it does not follow any pattern. To ensure that record IDs are unique, the record ID pool is chosen to be large enough, so that the ID lists allocated to different nodes are disjoint from one another. Record IDs are transmitted as a bit vector. For example, suppose there are nine record IDs (0-8) available, and record IDs 0, 4 and 5 are assigned to node A. During initial setup, an AP will transmit vector 100011000 to convey the list of record IDs assigned to A.

4.4 Node-Node Authentication and Record Exchanges

Mutual authentication between two nodes when they encounter each other is done in five steps as illustrated in Figure 5. During this process, each node authenticates the other node and lets that node know what witness records it needs. The first step consists of each node sending a Hello beacon periodically with following format

```
Hello | Dynamic ID
```

A node with dynamic ID a that hears a beacon from another node with dynamic ID b checks if the Dynamic ID b is in its treadmill table. If yes, it generates a random number Rnd1, and sends the following reply:

```
Reply Hello | a | b | {Rnd1}a_b_SKey |
{MAC}a_b_MKey
```

On receiving this message, node b verifies the MAC, and then decrypts the packet, and replies with

```
Exchange 1 | b | a | {Rnd1+1 | Record Vector b
| Rnd2}a_b_SKey | {MAC}a_b_MKey
```

Record vector b is a bit vector indicating the record ids of all witness records that b has. Note that these records include all records that b has generated as well as all records that b has received from other nodes in earlier encounters. On receiving the above message, node a verifies Rnd1+1 and the MAC. A successful verification authenticates node b to node a. Based on Record Vector b, node a calculates what records it needs from b. There are several criteria used for deciding what records are needed, including how much memory is available at a, how many witness records generated by different nodes are currently available at a, how much power is available, and so on[9]. This is represented by Record Vector Needed from b. It then replies with

```
Exchange 2 | a | b | {Rnd2+1 | Record Vector a
| Record Vector Needed from b}a_b_SKey |
{MAC}a_b_MKey
```

On receiving this message, node b verifies Rnd2+1 and the MAC. A successful verification authenticates node a to node b. Node b then calculates what records it needs from a, and sends the following message

```
Exchange 3 | b | a |  {Record Vector Needed
from a}a_b_SKey | {MAC}a_b_MKey
```

Finally, a and b send the witness records that they need from each other. This record exchange is interleaved, so that a first sends a witness record to b, then b sends a witness record to a, then a sends a witness record to b, and so on.

5 Implementation and Performance

We have implemented and experimented with the SenSearch security framework on MicaZ motes, running Mantis OS 1.0 beta. As mentioned earlier, MicaZ is equipped with a 8-MHz, 8-bit Atmel ATmega128 CPU, 4 KB of RAM, and 128 KB of flash memory.

5.1 Performance: Cryptographic Operations

To use RSA, we have ported BigDigits [7] to Mantis OS. BigDigits library uses 32 bits integer as a unit. To reduce the amount of memory needed to run this library, we removed all functions and variables that we didn't need for our prototype implementation. Despite this, there still wasn't enough memory for 1024-bit RSA public key operations. So, we modified the library by carefully replacing temporary arrays with global arrays. With this modification, we were finally able to run 1024-bit RSA public key operations. However, we were able to run only 512-bit private key operations. Luckily, we don't require the sensor nodes to run private key operations in our security framework. We measured the time taken

Table 1. RSA performance in Mantis OS (Times are in milliseconds)

Key size (bits)	**Encryption**	**Decryption**	**Signature generation**	**Signature verification**
1024	330.9	NA	NA	330.5
512	302	35441	35226	102

to complete various RSA operations (with e = 3) in our implementation (See Table 1). Times reported in this table are an average over ten different runs, each with a different key value.

There are two important observations we make from these measurements. First, the times for public key operations that must be performed on sensor nodes in our framework appear large when compared to other SenSearch operations such as a message exchange. However, these times are short enough to be easily completed during a contact between a sensor node and an AP, which typically spans more than 10 seconds. Second, when compared with other RSA implementations on sensor nodes, performance of public key operations in our implementation is quite competitive. For example, performance for the same operations has been reported as 14.5 seconds in TinyPK, which uses a slower CPU (4 MHz ATmega128L)[14]. In other implementations, performance of RSA public key operation in MicaZ has been reported as 0.79 seconds [13] and 0.43 seconds [8]. Thus our 1024-bit public key operation performance of about 330 ms is better than all these earlier implementations on sensor nodes. We attribute this performance improvement to our optimization of removing unneeded variables and functions, and using global arrays. Furthermore, we note that a fair comparison between different implementations is quite hard because of differences in hardware platforms and operating systems used.

For symmetric key operations, we have implemented RC5 with 12 rounds on Mantis OS. With a 128 bit key, it takes 87.5 milliseconds to encrypt or decrypt 1750 bytes of data, which was the size of the treadmill table in our experiments. It takes 56 milliseconds to create a MAC of these 1750 bytes. Once again, performance of our implementation of RC5 on MantisOS is quite competitive with other RC5 implementations on sensor nodes. For example, performance of encryption/decryption operation using RC5 in TinySec on Mica2 nodes has been reported as 0.9 milliseconds for 64-bit blocks. Our performance also compares well with performance of RC5 implementations on sensor nodes by other research groups [11,12]. However, we again note that a fair comparison is quite hard because of differences in hardware platforms and operating systems used.

5.2 Performance: AP-Node Authentication and Initial Setup

We measured the time spent in AP-node authentication and initial setup. As discussed in Section 4.3, this is a three-step process. The time interval we measured is from the time an AP sends a beacon until the time a SenSearch node receives and decrypts its dynamic ID, record ID vector, and treadmill table. In

our experiments, we assumed that a hiker will encounter upto 50 other hikers during a typical hike. Thus the treadmill table of a node contains 50 entries. Each entry contains dynamic node ID (2 bytes), weight (1 byte), and two symmetric keys (16 bytes each). Thus the size of a treadmill table in our experiments is 1750 bytes. We also assumed that there are 256 record IDs available. Thus the size of ID vectors is 32 bytes.

We measured time for node authentication and initial setup for two cases, which differ from one another in how real ID is encrypted in step 2. In the first case, real ID is encrypted using public key, while in the second case, it is encrypted using the symmetric key. We used a laptop as an AP and a MicaZ node as a senSearch node. Time for AP-Node authentication and initial setup was about 575 milliseconds in the first case, and about 285 milliseconds in the second case. Given a beaconing period of 3 seconds, and AP-node authentication and initial setup requiring about half a second, we conclude AP-node authentication and initial setup can easily be completed with in the first four seconds of an encounter between an AP and a SenSearch node. Since a typical encounter between an AP and a SenSearch node lasts for more than 10 seconds, we conclude that our security framework for AP-node authentication and initial setup is easily feasible.

5.3 Performance: Node-Node Authentication

We measured the time it takes to perform node to node authentication. As discussed in Section 5, this is a five-step process. The time interval we measured is from the time a node a sends its beacon message until the time it receives and verifies the vector of records needed by the other node. In particular, this time does not include the time spent in exchanging witness records. Time for node-node authentication was measured to be about 40 milliseconds. Given an encounter time of about 5 to 7 seconds, and a beaconing period of 3 seconds, there are 2 to 4 seconds available for node-node authentication and witness record exchanges in the worst case. Of these 2-4 seconds, node-node authentication takes up only about 40 milliseconds, which we think is quite reasonable.

5.4 Security Overhead

Finally, we measured the performance our third (optional) security feature described in Section 4.2. This feature requires generating a session key and encrypting each witness record before it is transmitted. We generate this session key in the last step (Exchange 3 message) of node-node authentication procedure. The sender node in this step first generates random key and sends it to the other node as follows:

```
Exchange 3 | b | a |  {Record Vector Needed
from a | Session key}a_b_SKey | {MAC}a_b_MKey
```

To send a witness record, a node encrypts the record using this session key and includes a MAC. This encryption and MAC generation takes about 5 milliseconds

for each witness record. While this time is relatively low, it should be noted that this encryption and MAC generation has to be done for every witness record transmitted during a node-node encounter. If the number of records to be transmitted is high, this overhead can become significant.

5.5 Power Consumption

As discussed in [9], the major source of power consumption in SenSearch is GPS module (acquiring GPS signal and receiving the location coordinates). This is followed by time spent in transmitting data, receiving data, and being idle. Our framework has no effect on GPS module. However, it does require nodes to send additional bytes in terms of MACs and random numbers during an AP-node encounter and a node-node encounter. In particular, 40 bytes out of 180 bytes exchanged during a node-node authentication account for MACs and random numbers. Assuming that a typical node-node encounter results in an exchange of about 50 witness records (1200 bytes), this implies that our framework results in the transmission of only an additional 2.8% bytes. So the overhead imposed by our framework on power consumption during a node-node encounter is quite low.

On the other hand, in an AP-node authentication and initial setup, the overhead is relatively large. This is because it involves transmission of treadmill table in which two rows correspond to symmetric keys. However, note that this initial setup is done only once when a hiker starts his/her hike. In fact, this initial setup is typically performed in a visitor center or at a location where ample source of power is available.

6 Conclusion

We have described the design, implementation and evaluation of a security and privacy framework for SenSearch, which is a search and rescue system for locating people in emergency situation in wilderness areas. There are two important challenges in building a security and privacy framework for SenSearch. First, the framework has to be implemented on resource-constrained devices, and second, there is only a limited time period during which most security operations have to be performed. Our framework carefully employs both symmetric and asymmetric key cryptography. Performance measurements show that our framework is feasible with in the constraints of SenSearch, and provide the required support for security and privacy.

References

1. 802.11-based tracking system, `http://www.pangonetworks.com/locator.htm`
2. Personal locator beacons with GPS receiver and satellite transmitter, `http://www.aeromedix.com/`
3. Personal tracking using GPS and GSM system, `http://www.ulocate.com/trimtrac.html`

4. Rf based kid tracking system, http://www.ion-kids.com/
5. Jaskowski, W., Jedrzejek, K., Nyczkowski, B., Skowronek, S.: Lifetch life saving system. In: CSIDC (2004)
6. Huang, J.-H., Amjad, S., Mishra, S.: CenWits: A Sensor-Based Loosely Coupled Search and Rescue System Using Witnesses. In: SenSys 2005 (2005)
7. Open source big number operation and RSA library, http://www.dimgt.com.au/bigdigits.html
8. Gura, N., Patel, A., Wander, A., Eberle, H., Shantz, S.: Comparing elliptic curve cryptography and RSA on 8-bit CPU. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 119–132. Springer, Heidelberg (2004)
9. Huang, J., Jiang, L., Kamthe, A., Freeman, I., Ledbetter, J., Mishra, S., Han, R., Cerpa, A.: SenSearch: GPS and witness assisted tracking system for delay tolerant sensor networks. TR 1045-08, Computer Science, CU-Boulder (2008)
10. Liu, A., Ning, P.: TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In: IPSN 2008 (2008)
11. Luo, X., Zheng, K., Pan, Y., Wu, Z.: Encryption algorithm comparison for wireless sensor networks. In: Proceedings of the International Conference on Systems, Man and Cybernetics (2004)
12. Vitaletti, A., Palombizio, G.: Rijndael for sensor networks:Is speed the main issue. In: WCNC 2006 (2006)
13. Wang, H., Li, Q.: Efficient implementations of public key cryptosystems on mote sensors. In: Ning, P., Qing, S., Li, N. (eds.) ICICS 2006. LNCS, vol. 4307, pp. 519–528. Springer, Heidelberg (2006)
14. Watro, R., Kong, D., Cuti, S., Gardiner, C., Lynn, C., Kruus, P.: TinyPK: Securing sensor networks with public key cryptography. In: SASN 2004 (2004)