SUMP: A Secure Unicast Messaging Protocol for Wireless Ad Hoc Sensor Networks

Jeff Janies, Chin-Tser Huang, Nathan L. Johnson
Department of Computer Science and Engineering
University of South Carolina
{janies, huangct, johnso66}@cse.sc.edu

Abstract— **Most wireless ad hoc sensor networks are susceptible to routing level attacks, in which an adversary masquerades as a legitimate node to convince neighboring nodes that it is the "logical" next hop or is on a "better" path for forwarding packets, and arbitrarily drops the packets forwarded by neighboring nodes. In this paper, we propose a secure unicast messaging protocol (SUMP) for wireless ad hoc sensor networks to mitigate the threat of routing level attacks. SUMP groups nodes into levels based on hop count to provide hop-by-hop group authentication using Merkel hash trees. This method allows for varied levels of security in accordance with a node's hop count from the base station and secure, directed unicast communications from the base station to individual nodes. Unlike other sensor network security protocols that require the storage of parent node information, a sensor node running SUMP does not store parent node information, therefore preventing an adversary from gaining information of other nodes from a compromised node.**

I. Introduction

A typical wireless ad hoc sensor network consists of a number of sensor nodes and a base station. Each sensor node is a small, cheap device that is programmed to collect certain types of data, for example temperature, humidity, and light. The base station is an aggregation point for collected data and is viewed as the human interface into the network. The sensor nodes are typically constrained by limited battery power, small memory, and low computational ability. On the other hand, the base station is assumed to have greater memory and processing power, and does not have the constraints of the sensor nodes.

With their low cost and scalability, wireless ad hoc sensor networks have found a variety of applications. For instance, in the military, such networks are used in target tracking, perimeter monitoring, and battlefield survey. Commercial applications of these networks include inventory control and building systems monitoring. However, the constraints on the sensor nodes, plus the open nature of communications in a wireless ad hoc sensor network, make securing networks of this type a challenging task.

A manifest vulnerability of many wireless ad hoc sensor networks is a class of attacks called routing level attacks. In a routing level attack, an adversary masquerades as a legitimate node to convince neighboring nodes to forward their packets to it, and then arbitrarily drops the packets forwarded by neighboring nodes. This class of attacks can potentially strand many well-behaved nodes and cause large holes in the sensing environment. Furthermore, by the compromise of individual nodes the adversary is able to develop hit lists of other nodes in the network, since most ad hoc sensor network protocols require a node to store the ID of its parent node. Two examples of routing level attacks, namely black hole attacks and wormhole attacks [5], are discussed below.

A black hole attack is launched when a node (either compromised or an adversary masquerading as a legitimate node) convinces neighboring nodes that it is the "logical" next hop for forwarding packets. The malicious node then arbitrarily drops the packets forwarded by neighboring nodes. In the case of multi-hop networks, the impact of such an attack is a hole in the sensing area as nodes begin to forward packets astray from their intended destination. This attack also has the potential to strand large areas of the network that are geographically distant from the malicious node by pulling messages from their intended paths.

A wormhole attack attempts to convince nodes to use a malicious path through legitimate means. In some wireless ad hoc sensor network protocols, for instance SPINS [12], when a message is sent from the base station, an intermediate node will forward the message with its identification information attached to the message. A receiving node records the identification information, regards the forwarding node as its parent, and proceeds to accept messages only from this parent. This provides an efficient method to prevent arbitrary rebroadcast of messages. In a wormhole attack an adversary with fast forwarding capabilities can quickly forward a message to nodes in the network. Fast forwarding is accomplished through the collusion of multiple units with fast out-of-band communication or a strongly powered device with a larger range of communication positioned between the base station and its target nodes. Once the target nodes are convinced that the adversary node is on a "better" path to the base station, all communications of the target nodes are attracted to the adversary node. An adversary can selectively forward information in an effort to disrupt the sensing environment.

Despite the susceptibility of wireless ad hoc sensor networks to routing level attacks, research in securing this type of network has primarily focused on the development of secure, lightweight protocols that satisfy only confidential communication and integrity of messages with little regard to the threat of routing level attacks. A popular approach is dividing the network into groups, or clusters, as discussed in [2], [3], [4], [7], [11], and [14]. These clusters are generally formed according to locality and require the addition of group and pairwise keys to be used by associated nodes. Furthermore, these approaches require a differentiation of authority where one node, known as the cluster head, has an additional responsibility: it functions as a local aggregation point for all data readings from nodes in the cluster. A node reports its readings to the cluster head, and the cluster head is responsible for forwarding those readings to the base station. Thus, the amount of data transfers is reduced significantly, but communications among cluster heads are vulnerable to both black hole and wormhole attacks.

Securing by groups is not the only approach. Perrig et al. [12] propose a method based solely on membership to the network. All nodes are treated as members to a single group. There are two major components in this approach: The Secure Network Encryption Protocol (SNEP) and µTesla. SNEP provides security for unicast packets from the base station to a given node and requires loose time synchronization between the two. State information is maintained by each node and the base station. µTesla provides security for broadcasts, through the use of delayed key distribution and one-way hash functions. Both components provide integrity but are also vulnerable to routing level attacks.

In this paper, we discuss the design and implementation of a novel protocol named Secure Unicast Messaging Protocol (SUMP). There are three goals in the design of SUMP: mitigate the threats of routing level attacks, limit the amount of knowledge that an adversary gains by compromising a node, and provide a level-wise grouping of nodes in contrast to the locality-wise grouping approach used in other works. Mitigating routing level attacks in the network provides survivability and reliability in hostile environments. By limiting the amount of information stored on a node and handling routing decisions at the base station, SUMP has the advantage of providing the current global view of the network to the base station. The level-wise grouping approach allows for varied degree of security according to the "need" of a node. For example, we note that nodes that are closer to the base station are more likely to be mediating for other nodes that are farther from the base station. Therefore, it represents a greater risk if the nodes closer to the base station are compromised or masqueraded, and greater care is needed for securing the communications of these nodes.

II. Environmental Assumptions

SUMP makes the following seven environmental assumptions:

1) There are no compromised nodes in the network during the initialization of the network.
2) A collision avoidance scheme exists in the initialization phase of the network.
3) The base station cannot be compromised.
4) The base station is not resource constrained.
5) A node's individual key and ID information are modifiable before deployment.
6) The base station is aware of all nodes in the network prior to deployment.
7) Nodes are fixed in location.

The first two assumptions are only restrictive for a fixed amount of time. As in [14], it is assumed that the minimum amount of time required to compromise a node is T_{min} and the amount of time required to initialize the network is T_{init}. Our approach fixes the initialization time prior to deployment of nodes such that $T_{min} > T_{init}$. Therefore, it is assumed that there are no compromises during the initialization of the network. Since initialization is fixed in time, collision could cause loss of data. With a simple yet efficient collision avoidance algorithm this loss can be mitigated. The second assumption is strict and is provided for ease of discussion. In implementation it is assumed that nodes have unique delays between transmissions. This assumption, though less strict than the second assumption, aids in collision avoidance and results in negligible overhead.

The third and fourth assumptions are key assumptions. In SUMP the base station maintains global information about the network, makes routing decisions, flags compromised nodes, and assesses connectivity of all nodes. The base station is assumed to be uncompromisable, since it is the human interface into the network and presumably not accessible by an adversary. However, this assumption does not limit the possibility of an adversary intercepting or modifying incoming and outgoing communications to and from the base station.

The fifth and sixth assumptions only apply in so far as the base station being able to uniquely identify the nodes that are potentially present in the network and record their key information. Location information is not provided, and no knowledge of connectivity is known prior to deployment. Similar assumptions are found in [6], [7], [12], and [14].

The seventh assumption limits the scope of the paper to wireless ad hoc sensor networks consisting of non-mobile nodes. It is assumed that nodes will be deployed in areas that are not easily accessible and left for the duration of sensing. Therefore, routes in the network are static.

III. The Secure Unicast Messaging Protocol

Similar to other wireless ad hoc sensor network protocols, SUMP classifies network entities into two types: the base station and sensor nodes. The base station maintains network information and is responsible for making decisions regarding message propagation. The sensor nodes gather readings and maintain a limited amount of information about the network. Due to the constraints on power, memory, and computational ability, sensor nodes provide only the most rudimentary of error checking. A sensor node does not keep any information about parent or child relationships with other nodes, which prevents an adversary from exploiting such a relationship to mount routing level attacks. Since the base station is responsible for routing decisions, there is little that an adversary can do to affect the route of a message at the node level. The base station is assumed to be uncompromisable according to Assumption 3, therefore an adversary cannot gain routing information from the base station. Furthermore, the route specified by the base station must be followed explicitly in order for the message to be authenticated correctly by the destination, as discussed in Section III.B.

The SUMP protocol consists of two phases of operation: *initialization* and *messaging*. Each phase supports different primitives and local storage requirements. The purpose of the initialization phase is to provide the base station with knowledge of individual node connectivity, density of distribution (with regards to connectivity, not locality), and establishment of paths. The messaging phase is the normal mode of operation used for data collection and dissemination.

The base station and nodes maintain information about the network that is specific to their individual view of the network structure. The base station's view of the network is global, and it maintains two primary structures: the *node structure* and the *group structure*. The node structure contains a list of all paths to each node, each node's distance in hops from the base station (called *hop count*), ID, and individual key. The group structure maintains information about all groups in the network in a linked list of group elements. A group element contains information about the group including a listing of all nodes' IDs, the distance from the base station to the group (called the *level*), and methods used for group membership authentication. A sensor node only maintains its own key information and group membership information.

The X-bow Motes™ are used for a prototype implementation of SUMP. With the current generation of Motes, certain implementation parameters must be observed. These parameters do not restrict the adversary or reduce the generality in any way, but are added for ease of development and explanation. First, all IDs, authentication values, and hash values used are 16-bit integers. Second, the length of the symmetric key of a node is 32 bits. Third, due to the limited packet length allowed by TinyOS [15], SUMP supports up to 10 hops of authenticated routing and approximately 1024 nodes. Fourth, there is a fixed time for initialization which starts when a node receives the hello message. The value used for this time is known prior to deployment. Furthermore, the space used to store this value is not considered when referring to overall memory requirement since this space is reclaimed after initialization. Finally, a *Message Type* byte is used to distinguish between message types: hello messages, hello reply messages, base station to node (outbound) messages, hop count change messages, root change messages, key change messages, and change hop count and key messages. Each of these message types represents a unique message type for operation in the current implementation. For the sake of brevity, only the first 4 messages are discussed in this paper.

Since it is already shown by Perrig et al. [12] that RC5 and MD5 are appropriate for use in networks of this design, this implementation does not include an implementation of RC5 and MD5. Instead, RC5-CBC encryption (with a 32-bit block size) is assumed. MD5 is used to generate hash values and checksums. For outbound messages a checksum is calculated from the message and the ID of the destination node. Therefore, the destination node is the only node that can verify the integrity of the message. Both outbound messages and the use of their checksums are described in subsequent sections. Checksum values are limited to 8 bits, and since the length of hash values generated by MD5 is much larger than the value used, it is assumed that only a subset of the resulting bits are used.

A. Initialization

The initialization method in SUMP uses breadth first search that is similar to the method discussed by Zhu et al. in [14]. However, in SUMP the base station is the only entity that initiates the search. The initialization phase in SUMP is divided into two steps: path establishment and verification. In the path establishment step, hop count and paths from the base station to a node are discovered. A node either waits to receive a hello message or forwards hello replies. In the verification step, the base station updates nodes that received an incorrect hop count due to discrepancies in the communication range. Such discrepancies arise when one node has a larger communication range than another. For example, node A has the ability to hear the communications from node B, but A's communication range is less than that of B's. When node A attempts to reply to node B, B is out of range of node A. Therefore, node A is unaware that node B is unable to receive communications from A. Thus, A and B have different views of their connectivity.

The base station initiates the path establishment step by issuing a hello message containing a count of 0. The structure of this hello message is shown in Fig. 1. The count corresponds to the current hop count from the base station. Nodes do not respond to any communications until the hello message is received. Once a node receives the hello message the node will record the hop count into its memory, increment the hop count in the hello message, and forward the message. The node then replies to the base station with a message containing the hop count recorded and the ID of the node.

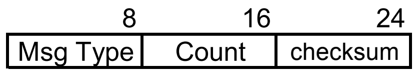

Figure 1 Hello Message

Following the forwarding of the hello message, the node enters a reply forwarding state in which it will listen for other hello replies, whose structure is shown in Fig. 2. When it receives a hello reply from another node it places the message in a message buffer, concatenates its own ID to the end of the message, and retransmits the message. If the node receives a hello reply message that contains its own ID, it will not respond. This avoids the formation of infinite routing loops that deplete resources. A sequence diagram of path establishment is shown in Fig. 3.

8	16	24	32
Msg Type	Count	checksum	
ID1		ID2	
ID3		ID4	
ID5		ID6	
ID7		ID8	
ID9		ID10	

Figure 2 Hello Reply Message

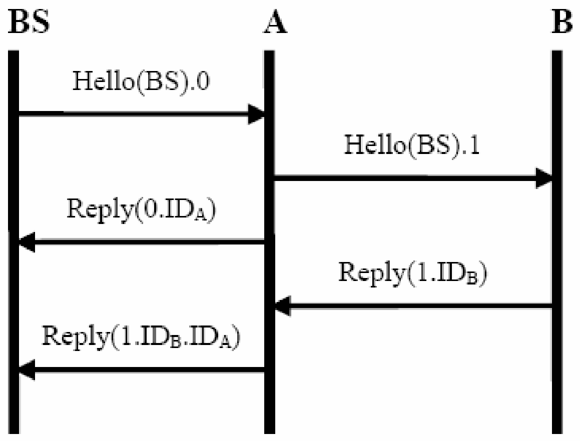

Figure 3 Message sequence diagram of path establishment

When the base station receives a reply from a node it finds the path traversed by this reply based on the ID list found in the reply message. The path derived from the first reply message received from a node is stored as the primary path to the node, and all paths derived from replies received after the first reply are stored as alternate paths. These alternate paths are used to reduce packet loss and enhance the survivability of the network as dynamic events occur.

The path establishment step persists for a preset interval of time. After the expiration of this period, all nodes and the base station enter the verification step. In the verification step the base station compares the node's recorded hop count with the length of the first path received in the path establishment step. If a discrepancy exists the base station rectifies this by sending a hop count change request encrypted with the individual key of that node. The hop count change request includes the ID of the destination node and the hop count value determined by the base station, as seen in Fig. 4. This ensures that the hop count is representative of a symmetric path between the base station and the destination node. In order to assure that these requests are delivered to their destination every node that overhears a hop count change forwards the message. If a node receives a hop count change request intended for it, the node updates its hop count accordingly.

8	16	24
Msg Type	ID	Count

Figure 4 Hop Count Change Request

Once verification of appropriate hop count is completed, the base station groups nodes according to hop count. All nodes of the same hop count are members of the group with the corresponding level value. For example, the group representing level one is comprised of all nodes with a hop count of one. Once this grouping is completed, the base station computes the group's key information and distributes it to all nodes in the group.

B. Messaging

Hop counts of nodes can be used to generate key material and determine membership status in the network. The network in Fig. 5 can be represented in a general tree structure where the level of a node in the tree corresponds to its hop count, as shown in Fig. 6. Nodes of the same hop count are treated as a group. Therefore, nodes are grouped topologically by connection and not geographically by location.

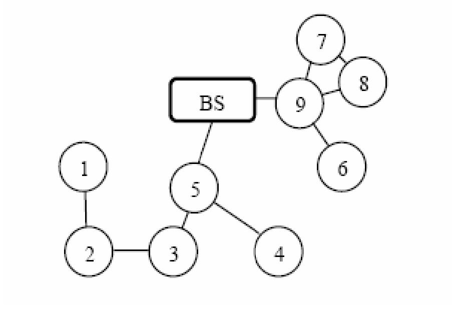

Figure 5 A sample connected wireless ad hoc sensor network

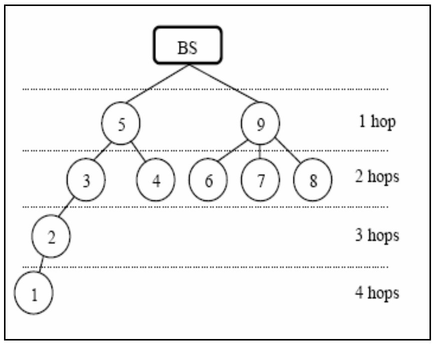

Figure 6 Tree structure for authenticated levels

This method lends itself to routing by level, in which only one node per hop forwards a received message. When a message is transmitted from the base station to a node, the primary path to the node is used to ensure that the message is not arbitrarily rebroadcast throughout the network, thus conserving network resources while directing communications towards their ultimate destination. However, in order to prevent malicious redirection of packets, a group authentication method is required.

Several group authentication methods are available for membership testing, but due to the constraints of the sensor nodes, many of them are not feasible. Benaloh et al. [1] propose the use of one way accumulators for this method of authentication, where one key value is stored by a node. In this method messages are prepended with an authentication value. This method has the advantage of small size, but is computationally demanding to compute and verify membership. Liu et al. [8] propose a key chain commitment that requires the node to store several key values, increasing the memory requirement on the nodes.

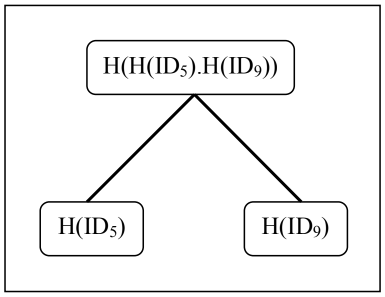

Figure 7 Merkle Hash Tree of level 1

Merkle hash trees, as discussed in [9], exhibit a low computational cost and storage requirement but still provide secure group membership authentication. A Merkle hash tree uses a secure one-way hash function to generate a binary tree in which the members of the group are represented as the leaf values. The tree is formed by concatenating the sibling values and hashing the result to form a parent element of the tree. As is seen in Fig. 6, the network is divided into levels based on hop count, and each level represents a group. The group that represents level 1 consists of nodes 5 and 9. Level 1's group is represented as a Merkle hash tree in Fig. 7. In this representation, the hash values of nodes 5 and 9 form the leaves of the tree. The remainder of the tree is formed according to the following three rules. First, the tree is a balanced binary tree. Second, if an element is a leaf of the tree, then its value is the hash of a node's ID. Third, if an element is not a leaf of the tree, then its value is the result of hashing the concatenation of its two children elements' values.

As a result of constructing a tree with the above rules the root value is a representative number of the entire group membership and with a node's ID is used to authenticate a message. The base station maintains a representation of the entire tree. The root value, height of the tree, and node's ID are the only values stored by the node. The tree cannot be produced unless all node IDs are known. Since nodes do not store the IDs of other nodes in the network, an adversary cannot capture a node and reconstruct the membership tree from the root value.

These trees provide a method in which messages are securely authenticated by the destination with the addition of *h* authentication values, where *h* is the height of the tree. The authentication values are the values needed by the node to reproduce the root value from already known information, i.e. the sibling values of elements in the path from the node's hashed ID to the root of the tree.

Since the path to every node is known and the nodes are grouped according to hop count, Merkle hash trees provide a method by which a message is directed hop-by-hop from the base station towards the destination. The base station communicates with the node by encrypting messages according to the established primary path. The base station concatenates the ID of the destination node to the message and encrypts the result with the key it shares with the destination node. Then the authentication values are attached to the beginning of the message as per the Merkle hash tree algorithm. The base station then uses the stored primary path to the node to encrypt the message further. For an intermediate node between the base station and the destination node the base station encrypts the entire message with the shared key of the intermediary node and concatenates the authentication values of intermediate node with the resulting message. This results in the encapsulation of the original message, *M*, in a message to the intermediary node. For example, if the path from the base station to a node is *{5, 3, 2}*, the message produced is:

$$E_{K5}\left(\begin{array}{l}\{authValues5\}.\\ E_{K3}\left(\{authValues3\}.E_{K2}\left(\{authValues2\}.ID_2.M\right)\right)\end{array}\right)$$

Upon receiving a message of this structure, a node uses its key to decrypt the message and attempts to authenticate the message to determine if the message is to be forwarded by the node. If the node successfully authenticates the message using the authentication value in the message, it checks if the decrypted message begins with its ID. If so, the decrypted message is for the node and will not be retransmitted; otherwise the node forwards the decrypted message to the next hop. In the case of the current implementation of SUMP the structure of outbound messages is as shown in Fig. 8.

8	16	24	32
Msg Type	Seq#	Auth 1	
Auth 2		Auth 3	
Auth 4		Auth 5	
Auth 6		Auth 7	
Auth 8		Auth 9	
Auth 10		Dest ID	
Message			
checksum			

Figure 8 Messaging Phase Outbound Message

Regardless of the number of actual authentications needed, the message size is assumed to be fixed, and therefore, unneeded or previously processed authentication values are represented by a zero value. An individual node only uses as many non-zero *auth values* as necessary to authenticate a message for that hop. For instance, if a given level has a Merkle hash tree of height 2, then only the first two non-zero auth values are used in authentication. Once used the auth values are replaced by zeros. If the message is authenticated, the remaining auth values, message field, and checksum are decrypted. The resulting packet is forwarded to the next hop. The sequence number is added to a node's transmitted list only if the message is authenticated. This mitigates premature processing of a message which can result in a breakdown of communication.

C. Dynamic Node Events

Dynamic node events, such as node joins and node deaths, are open problems. A *reseeding*, or addition of more nodes, represents an interesting challenge to SUMP. It is assumed that the communication channel is free of malicious activity only during the initialization phase, and therefore the initialization method discussed previously is not effective for adding new nodes to an established network after the expiration of the initialization phase. Although an initialization key to encrypt all initialization messages would alleviate this concern, the secure distribution of such a key is resource consuming. Therefore, this problem remains open. Node death gives rise to several questions. How is node death detected? What happens to nodes that are unable to receive communications from the base station due to the death of a node? Since a node's death may alter the hop count of one or more nodes, does the base station have to regroup all nodes that had paths containing the dead node? These questions, among others, present an interesting challenge to this protocol. Since the base station maintains a list of alternate paths to a node, communication can be re-established easily, but the group membership may no longer be representative of the network. Thus, the node death problem remains open.

IV. Evaluation

For the purpose of analysis, SUMP is mainly compared to SNEP (the unicast protocol presented in [12]). SNEP is known for its efficiency and security in wireless ad hoc sensor networks. The purpose of this exclusive comparison is twofold. First, SNEP is well documented in its weaknesses and strengths. The authors provide extensive analysis of its potential and overhead in this specific network environment. Second, SNEP and SUMP are designed exclusively for unicast messaging and therefore, the comparisons made are fair and accurate to the protocols' design.

A. *Storage Requirements*

The storage requirements of SUMP are comparable to SNEP. As previously mentioned, the ID and Merkle hash tree root values are 16 bits in the current implementation, and the symmetric key shared with the base station is 32 bits. The tree height is stored as a 16-bit integer, and the hop count is an 8-bit integer, resulting in a total storage requirement of 88 bits. Although SNEP requires nodes to store only a symmetric key and counter, it also requires synchronization between the two communicating parties and knowledge of the parent, or next hop from the node to the base station. Therefore, if it is assumed that the counter is stored in a 16-bit integer and the key size is equivalent, SNEP requires only 48 bits of storage, but with only 40 bits of additional storage SUMP does not require synchronization or knowledge of the parent.

B. *Communication Overhead*

Communication overhead is comprised of two main components: size and frequency. For the purposes of this analysis, size only refers to the additional bytes added to a message to provide security. Obviously SUMP incurs much overhead in outbound communications due to the addition of the authentication values, which alone require 20 bytes. In comparison, although SNEP requires only 8 bytes of overhead, it does not provide any directional information. Due to the utilization of directional information SUMP performs better then SNEP with regards to frequency of outbound communication. In this context frequency refers to the number of times that a packet is transmitted beyond what is required for the message to be received by the destination node. In the case of outbound communication, SUMP has little additional overhead with regard to frequency since the message is directed. SNEP requires all nodes to rebroadcast every message in order to ensure that a message is received. However, in SNEP the deliverability of the message is dependent on proper forwarding by all intermediate nodes. Therefore, breakdowns in communication require re-establishment of paths.

C. *Resistance to Routing Level Attacks*

SNEP requires nodes to maintain parent information and thus is vulnerable to routing level attacks. SUMP does not share this vulnerability. Since no information about a node's parents or children is stored by the node, these forms of attacks are limited in impact. In the case of the black hole attack there is no disruption. In the case of communications from the base station to a given node, if a path is compromised the base station can use an alternate path. Therefore, SUMP effectively eliminates these routing threats.

D. *Weaknesses*

We note that SUMP has two identifiable weaknesses. First, outbound messages are limited to 32 bits of data and thus the number of messages is potentially increased. This results in greater consumption of power to transmit the same amount of information. Second, SUMP limits the sizes of groups. Due to the fixed message length, only a limited number of authentication values can be included in a message.

The limit on the size of outbound messages does not hinder this method as greatly as would be expected. Due to the limited amount of information stored on a node, individual updates require very little space. However, the overhead of the security implementation does deplete resources. This depletion of resources is mitigated by the fact that unicast messages follow specific paths and are completely ignored by nodes that are unrelated to the communication.

This implementation of SUMP does not allow for all possible network configurations. It can be seen that with the mote's limited packet size and 16-bit authentication values, only ten authentication values can be stored in each message. Additionally the payload for these messages is limited to 32 bits. With the introduction of the message type byte in the message and the limited storage requirements this size is sufficient for maintenance operations such as key update, level update, sleep/active update, and group root value update.

V. Concluding Remarks

In this paper, we introduce SUMP, a secure unicast messaging protocol for wireless ad hoc sensor networks. Security and efficiency are the two emphases in the design of SUMP. Through the use of Merkle hash trees, SUMP provides security for these networks with survivability. As shown in our experiments, SUMP is applicable even on very constrained devices such as the X-bow Mote.

The strengths of SUMP are manifest in the following three regards. First, it is demonstrated that SUMP is not susceptible to black hole and wormhole attacks that would otherwise allow an adversary to disrupt communication in the network. Second, it is shown that very little storage is required by SUMP for sensor nodes to securely route outbound messages. Third, communication overhead is alleviated by avoiding arbitrary rebroadcast of messages.

Future expansions of this work include the introduction of a secure broadcast mechanism that capitalizes on the global knowledge maintained by the base station, the development of both node-to-node and node-to-base station (inbound) communication methods, and the extension of a sleep state to save power. It is desirable to develop a lightweight secure broadcast protocol in addition to the proposed SUMP, and we believe that the global knowledge maintained by the base station in SUMP can be exploited to reduce the computational cost. Moreover, the group establishment could allow for sub-network routing to route messages among nodes in the same group and allow for efficient inbound messaging.

Acknowledgment

This work is supported in part by an AFRL/DARPA grant (FA8750-04-2-0260). The authors thank Dr. John Zachary and our colleagues at NIST for their support.

References

[1] J. Benaloh and M. de Mare. "One Way Accumulators: A Decentralized Alternative to Digital Signatures". In *Proceedings of Advances in Cryptology (EUROCRYPT'93)*, Vol. 765, *Lecture Notes in Computer Science, Springer*, 1993.

[2] A. Cerpa and D. Estrin. "ASCENT: Adaptive Self-Configuring Sensor Networks Topologies". Available at http://citeseer.ist.psu.edu/559481.html.

[3] W. Du, J. Deng, Y. Han and P. Varshney. "A Pairwise Key Pre-distribution Scheme for Wireless Sensor Networks". In *Proc. of the 10th ACM Conference on Computer and Communication Security (CCS 03)*, Washington DC. 2003.

[4] M. Goodrich, R. Tamassia, and J. Hasic. "An Efficient Dynamic and Distributed Cryptographic Accumulator". In *Proceedings of the Information Security Conference (ISC '02)*, Vol. 2433, *Lecture Notes in Computer Science*, Springer, 2002.

[5] C. Karlof and D. Wagner. "Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures". In *Proc. of First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

[6] D. Liu and P. Ning. "Multi-level μTesla: A Broadcast Authentication System for Distributed Sensor Networks". Submitted for review, 2003.

[7] D. Liu and P. Ning. "Establishing Pairwise Keys in Distributed Sensor Networks". In *Proc. of the 10th ACM Conference on Computer and Communication Security (CCS 03)*, Washington DC. 2003.

[8] D. Liu and P. Ning. "Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor Networks". In *Proc. of 10th Annual Network and Distributed System Security Symposium (NDSS 03)*, San Diego, CA, 2003.

[9] R. Merkle. "Protocols for Public Key Cryptosystems". In *Proceeding of the IEEE Symposium on Security and Privacy*, 1980.

[10] MICA2™ Specifications Data Sheet. *Document 6020-0042-04*, *Available at http://www.xbow.com/Products/Product_pdf_files/ Wireless_pdf/6020-0042-04_B_MICA2.pdf* Rev. B, May2003.

[11] T. Park and K. Shin. "LiSP: A Lightweight Security Protocol for Wireless Sensor Networks". *In Proc. Of the ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, August 2004.

[12] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. Tygar. "SPINS: Security Protocols for Sensor Networks". In *Proc. of Seventh Annual ACM International Conference of Mobile Computing and Neworks (MOBICOM 2001)*, Rome Italy, July 2001.

[13] F. Stajano and R. Anderson. "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks". *AT&T Software Symposium*, 1999.

[14] S. Zhu, S. Setia, and S. Jajodia. "LEAP: efficient security mechanisms for large-scale distributed sensor networks". In *Proc. of the 10th ACM Conference on Computer and Communication Security (CCS 03)*, Washington DC. 2003.

[15] TinyOS, available at: http://www.tinyos.net.