Li, Bai, Batten, Lynn and Doss, Robin 2009, Lightweight authentication for recovery in wireless sensor networks*, in MSN 2009 : Proceedings of the 5th International Conference on Mobile Ad-hoc and Sensor Networks*, IEEE Computer Society, Piscataway, N. J., pp. 465-471.

2009 Fifth International Conference on Mobile Ad-hoc and Sensor Networks

Lightweight Authentication for Recovery in Wireless Sensor Networks

B. Li, L.M. Batten, Senior Member, IEEE and R. Doss, Member, IEEE
School of Information Technology
Deakin University
Australia
Email: {libai, lmbatten, rchell}@deakin.edu.au

***Abstract*—Wireless sensor networks (WSNs) suffer from a wide range of security attacks due to their limited processing and energy capabilities. Their use in numerous mission critical applications, however, requires that fast recovery from such attacks be achieved. Much research has been completed on detection of security attacks, while very little attention has been paid to recovery from an attack. In this paper, we propose a novel, lightweight authentication protocol that can secure network and node recovery operations such as re-clustering and reprogramming.**

Our protocol is based on hash functions and we compare the performance of two well-known lightweight hash functions, SHA-1 and Rabin. We demonstrate that our authentication protocol can be implemented efficiently on a sensor network test-bed with TelosB motes. Further, our experimental results show that our protocol is efficient both in terms of computational overhead and execution times which makes it suitable for low resourced sensor devices.

***Keywords*-Wireless sensor network, hash function, authentication.**

I. INTRODUCTION

Wireless sensor networks are deployed in many mission critical applications and for monitoring of critical areas of national defence.

With their development, various novel security attacks have appeared. The aims of these attacks are usually to take over nodes in the network, destroy nodes or to disrupt data flow. Since the base station can easily detect if only a few nodes are sending anomalous data (there are many protocols for this in the literature- see for example [13]), attacks are usually based on compromise of a large part of the network. Detection and recovery from these attacks have become major challenges in protecting sensor networks.

In this paper, we focus on recovery after an attack has been detected. We assume that the network is low resourced but has the capability to self-organize (cluster), detect an attack on the nodes, determine which nodes have been compromised (with high probability) and make a decision to reprogram compromised nodes.

Numerous papers have been written on clustering, re-clustering, attack detection and reprogramming [4], [13], [11], [7], however, this is not the focus of our paper.

Manuscript submitted March 31, 2009. This work was supported by the National Security Science and Technology Branch of the Australian Government under grant **07-0030**.

In a recovery operation, it is critical that messages from the base station requesting re-clustering or reprogramming be authenticated, as otherwise, a denial of service attack can be launched by anyone intending to cause havoc in the WSN. Our focus is on ensuring that authentication is effective and efficient. In the protocols presented here, we are assuming only that authentication has been implemented, and ignore other means of securing data such as encryption.

In particular, the contributions of this paper are:

- An efficient authenticated reprogramming request protocol from the base station to a compromised node.
- An efficient authenticated reprogramming request protocol from an aggregator node to the base station on behalf of a compromised node.
- An efficient authenticated re-clustering protocol from the base station to the network.
- A comparison of two lightweight well-known hash functions used in the authentication protocols and implemented with TelosB motes

The rest of the paper is outlined as follows. After an overview on related work in Section II, we present the system assumptions in Section III. Three authentication protocols are then presented in Section IV. These protocols are based on the use of a hash function and so, in Section V the two hash functions used are compared and tested in settings varying between 8 and 64-bit input. In Section VI, we examine attack scenarios and the security of our proposed scheme against them. The performance of the scheme is discussed in Section VII and includes comparison with other authentication schemes used in WSN recovery. Section VIII concludes the paper.

II. RELATED WORK

A number of authentication methods have been proposed in WSNs. In this section, we give an overview of this work.

Perrig et. al. [17] proposed the µTESLA protocol to securely broadcast messages in a WSN. This protocol uses a one-way hash chain (OHC) to authenticate broadcast messages. To tolerate packet losses, µTESLA has been extended by introducing multi-level OHCs [14]. A higher-level OHC is used to bootstrap low-level OHCs. These methods require time synchronization which is not practical in a low-resource WSN. Hu et. al. [8] proposed a secure on-demand routing protocol for ad hoc networks, in which an OHC is used to thwart malicious routing request

978-0-7695-3935-5/09 $26.00 © 2009 IEEE
DOI 10.1109/MSN.2009.47

floods. OHCs were used in INSSENS to limit broadcast floods for control routing updates in WSNs [4]. In using OHCs, problems unique to unicast messages must be addressed, for example, maintaining OHCs when many packets are lost, and generating and storing OHCs in a highly resource-constrained node.

General cryptographic authentication methods based on public-key cryptosystems tend to be unreasonable for use in WSNs because of the very low calculation capability and small memory [10]. Nevertheless, some researchers have suggested using such security mechanisms in sensor networks and the TinySec development of Karlof, Sastri and Wagner in 2004 [9] was developed to tackle the problems head on. It aims at providing all of access control, message authentication, data confidentiality and avoidance of replay attacks. However, TinySec does not specify any key pre-distribution method but assigns a global key to the system, stored always in the same location. "Thus, an attacker seeking the TinySec key need only target a well-known address or area of memory." [6].

In all schemes using authentication, some kind of key is needed and the question of key storage is critical to maintaining a WSN. In virtually all known schemes for WSNs, keys are stored on nodes and thus require a tamper resistant location or an assumption that an attacker is unable to retrieve the keys. An exception to this is given, for example in the paper [5] where the authors use public key cryptography for authentication in which the private key is stored on a PC. A hash chain is produced by the PC and the nodes must verify all messages from the PC itself. This is a time-consuming undertaking and needs each node to be able to communicate with the PC at all times. In addition, it is important that messages be received sequentially so that they can be reconstructed. In an attack scenario, this is not an assumption that can be made.

Our aim is to provide authentication for recovery message transmissions in a WSN in an efficient and effective manner with the goal of extending the life of the WSN for as long as possible. An examination of the above work leads us to implementation of a hash function method in a protocol which also provides protection against message content changes, message authorship changes and replay attacks. As we include remote location of the WSN without possibility of key update, we make the standard assumption that nodes contain a tamper-proof module to store secrets. We describe these attack scenarios in Section VI.

III. PRELIMINARIES

WSNs can be built in a number of ways depending on the desired application [15]. Often, several types of nodes are present, classified in terms of the role they play, such as gathering data, analyzing data, deploying applications etc.

An intuitive analysis of the sensor network activities of a simple network leads to mapping tasks to roles as follows:
Member node – senses and transmits data
Aggregator node – controls member nodes and senses, collects, aggregates, analyses and transmits data
Base station – controls the system and collects, analyses, transmits and stores data.

Since all sensor nodes in the network are essentially deployed to collaboratively sense target events, all nodes must assume the functions of a member node. Aggregator nodes, in addition, take on the responsibility of coordinating the sensing activities in their neighbouring region (also known as a sensing zone) and aggregate and forward the information to the base station. The task of coordination is not a simple one and it is also not a short term job. In order to provide instantaneous sensing and reporting capability (dependent upon sensing applications) each aggregator node may need to systematically rotate its responsibilities transparently among neighbouring nodes without much communication overhead.

For the purposes of an attack situation in which nodes can be lost or compromised, detection and recovery can only take place efficiently if the network can function as normally as possible. We propose to therefore retain connectivity and maximize flexibility in the network. This is achieved by allowing each node to play the role of either a member or aggregator node as appropriate under the conditions arising, and at the same time, for the entire network to efficiently re-organize itself in order to remain connected.

Our platform is TinyOS Version 2.1 implemented on Crossbow's TelosB sensor nodes [21].

A. Assumptions

Before proceeding further we present out system assumptions. These are:

- We assume the existence of a globally unique ID for each sensor node. The base station keeps track of all IDs.
- We assume that every node in the WSN is in transmission range of the base station, while it is not necessarily the case that each node can transmit to the BS. Because some nodes may not be in range of other nodes, all reprogramming and re-clustering commands will therefore be run through the base station.
- We assume that each sensor node has a secret value which can be used for authentication. This secret is known only to the base station and the node and is allocated when the WSN is set up. We also assume that this secret is stored in a tamper-resistant section of the node. (Note that while tamper resistance might be a viable defence for physical node compromise for some networks, we do not see it as a general purpose solution. Thus, we assume that an attacker may be capable of retrieving the entire set of codes residing in the node. However, in order to implement authentication, it is important to have some item residing on the node which cannot be retrieved or identified as being a 'secret'.)
- We assume that the base station is trustworthy in the sense that it is never compromised and always behaves correctly. (Most, but not all routing protocols depend on

nodes to trust messages from base stations.)

- We assume that an essentially infinite timer is available to each of the nodes and the base station. (In practice, timers on nodes may overflow and re-use previous times which invalidates our protocol.)
- We assume that the same hash function code is programmed into each node at set-up. This code will be used for authentication.

Figure 1 illustrates the topology of the envisioned WSN. It comprises a large number of low-resource sensor nodes that are connected to a base station (BS) in order to analyze the sensed data. By partitioning the WSN topology into a set of clusters, a hierarchical network topology is obtained that is power-efficient, scalable, and resilient to security attacks [1].

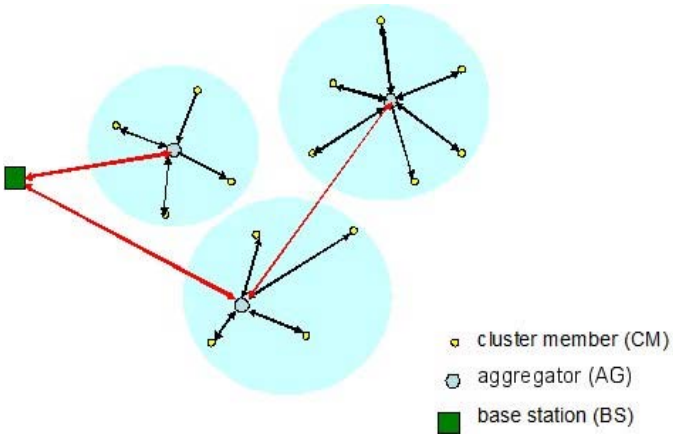

Figure 1: Clustered topology of wireless sensor network

Each cluster comprises one aggregator (AG) and in general several cluster members (CMs). Each CM is always connected to a single associated AG to exchange data and control packets.

In transmitting critical messages between nodes and the base station, in particular, commands to re-cluster or to reprogram nodes in response to an attack, message authentication is critical as a denial of service attack can easily be implemented without it.

IV. THE PROTOCOL

In this section, we describe each of the protocols of Section 1 in detail. In each case, a node with ID n contains secret S_n, R represents a random value but is in fact the local time obtained from the LocalTime.get() command in TinyOS, M represents a message to reprogram or re-cluster or a request that another node be reprogrammed. All messages transmitted include node ID of both sender and receiver, including that of the BS.

A simple hash function check achieves authentication of a message. We use two well-known functions, SHA-1 and Rabin for the purposes of comparison. SHA-1 is a preferred hash in government authentication protocols [3; Section 3.9.12], [16] and has been employed by several researchers in TinyOS applications ([10], [12]). Our second choice is the Rabin encryption system [19] adapted for use as a hash function as proposed by Shamir in [20]. While Shamir suggests an adaptation of Rabin's scheme to what he calls SQUASH, based on improved methods of computing the hash output, our implementation is constrained by TinyOS requirements and so we use smaller values than those proposed to ensure the security of SQUASH. Such smaller values do not warrant use of the improved SQUASH computations, and so we use Rabin's scheme as proposed in [19]. Shamir points out [20] that, used as a hash function, the Rabin encryption scheme can be implemented securely on many fewer bits than needed for encryption. The drawback of Rabin's scheme is that up to four input messages can result in the same ciphertext. Since we use the scheme only for authentication based on a small set of standard message inputs, this is not a problem for us.

We take data input of 8 bits, 32 bits and 64 bits for SHA-1 and 64 bits for Rabin.

A. *Reprogramming Protocol from the Base Station to a Compromised Node*

Because the BS monitors network communications, it can detect an attack. It would deal with it in the early stages by authenticated reprogramming and re-clustering. The objective is to prolong the life of the WSN for as long as possible; compromise of over 50% of the nodes would be deemed failure.

The BS has determined that node with ID n needs reprogramming. If a set of nodes is to be reprogrammed, each node must receive a separate message as each contains the secret known only by that node and by the base station. H represents a hash function. We assume that n, M and R are the appropriate size for input to H.

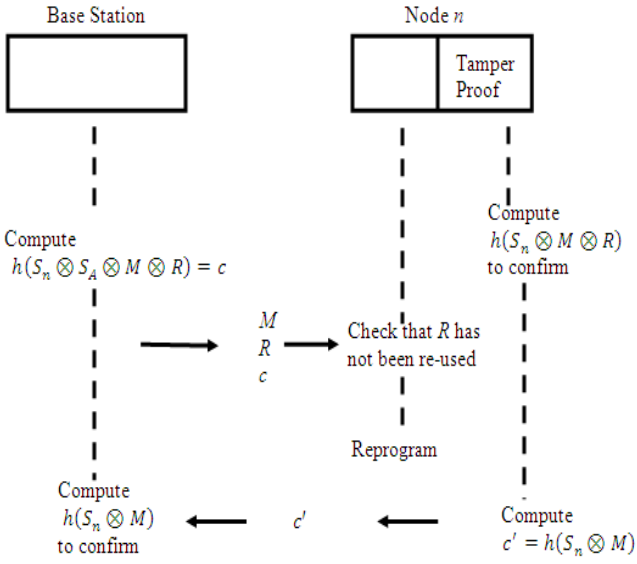

Figure 2: Authenticating a reprogramming request

(a) BS XORs the secret S_n, the reprogramming message M and the local time R to obtain m.
(b) BS computes H(m) = c.
(c) BS transmits c, M and R which are received by n.
(d) Node n re-computes H of the XOR of n with S_n, M and R and checks that it is c.
(e) If the check is 'true' AND the time R has not been used in a reprogramming request earlier, the node reprograms.
(f) The node updates its secret to S_n= m.
(g)The node informs the BS that it has successfully reprogrammed and the BS then updates the node's secret in its table.

B. *Reprogramming Protocol from an Aggregator Node to the Base Station on Behalf of a Compromised Node.*

In this protocol, an aggregator node A has determined that one of the nodes in its cluster must be reprogrammed. Since reprogramming is intensive from the initiator side, only the BS can implement it. Thus, A requests reprogramming from the BS.
(a) A retrieves the ID n of the node to be reprogrammed.
(b)A XORs n, its own secret S_A, the request for reprogramming message M and the local time R to obtain m.
(c) A computes H(m) = c.
(d) A transmits c, n, M and R to the BS.
(e) The BS retrieves S_A, re-computes the hash and checks if it is c.
(f) If the check is 'true' AND the time R has not been used in a reprogramming request earlier, BS initiates protocol A.

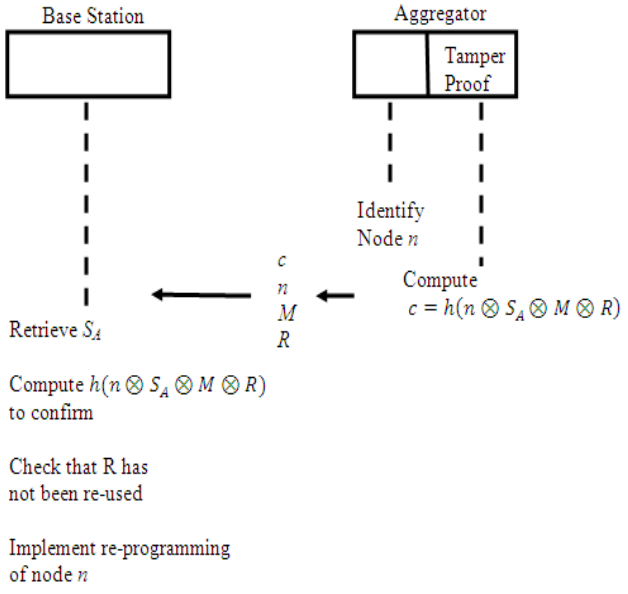

Figure 3: Authenticating a reprogramming request from an aggregator

C. *Re-clustering Protocol from the Base Station to the Network.*

Re-clustering may need to be implemented after an attack on a WSN in case several nodes are no longer trust-worthy. The BS makes this decision, based on information gathered from the network. Once it has been made, it sends a re-clustering message to all aggregator nodes to begin the procedure. In this protocol, we describe the authenticated message to re-cluster which must be sent to each node separately, identifying it by its ID and its secret.
(a) The BS retrieves the secrets $S_1, \ldots S_c$ of each node and for each node ID n, XORs its secret S_n, the re-clustering message M and the local time R to obtain m_n.
(b) The BS then computes $H(m_n)$ for each n and transmits it with M and R to the corresponding node n.
(c) Each node XORs its secret S_n, M and R to obtain m_n.
(d)Each node computes $H(m_n)$ and compares with the messages received from the BS. If there is a match AND if no such message with time R has been used in a re-clustering request earlier, n accepts this as a valid re-clustering message.
(e) Once all nodes have received and verified such a message, re-clustering commences.

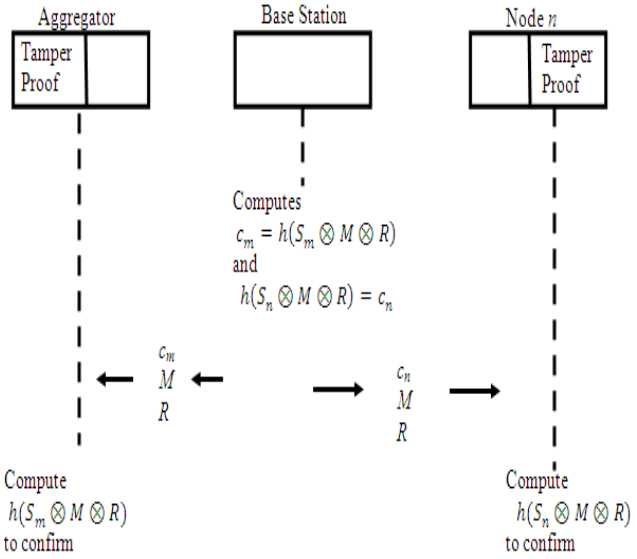

Figure 4: The BS authenticates re-clustering messages to nodes

When running this protocol, the timing must be managed. A Timer.startOneShot(2000) command is set to fire in 2000 ms from the time of invocation at the base station. This cancels any previously running timer and will only fire once, then stop. This gives enough time for the node in the network to receive and verify the hash message before the base station starts another timer. The protocol is complete when the last node verifies the hash message. Timing synchronization is not required. We require only that the timer value (independently scheduled at node) be set to a value (we use 2000msec) that allows for the hash computation to complete.

We tested all three protocols on TinyOS Version 2.1 with the results presented in the following tables. Table 1 shows the time, ROM and RAM used by the reprogramming

protocol for four applications of hash function. Although the execution time does not vary, Rabin uses only 83 to 85% of the RAM and ROM used by SHA-1 on 64 bits.

Primitives	Execution Time (sec)	RAM (bytes)	ROM (bytes)
SHA-1 (8 bits)	6.0	1230	19802
SHA-1 (32 bits)	6.0	1262	20178
SHA-1 (64 bits)	6.0	1334	21034
Rabin (64 bits)	6.0	1100	17972

Table 1. Performance of reprogramming protocol from the base station to a compromised node

The same distinctions are apparent when running protocol II. Again, while execution time does not vary, ROM and RAM for Rabin are only 84 to 86% that of SHA-1 on 64 bits.

Primitives	Execution Time (sec)	RAM (bytes)	ROM (bytes)
SHA-1(8 bits)	10.0	1262	20556
SHA-1 (32 bits)	10.0	1296	20948
SHA-1 (64 bits)	10.0	1368	21796
Rabin (64 bits)	10.0	1142	18830

Table 2. Performance of reprogramming protocol from an aggregator node to the base station on behalf of a compromised node

Protocols I and II were run 100 times with no substantial difference in results. Tables 1 and 2 average these results.

For protocol III, because the distribution of messages needs to be associated with timers, the implementation is significantly more complex than that for the previous protocols; therefore we ran this protocol only five times. Table 3 presents the average results of running protocol III five times with a set of 5 TelosB motes. In each case, time starts when the BS issues its first message and stops when the last node verifies the hash message.

Primitives	Execution Time (sec)	RAM (bytes)	ROM (bytes)
SHA-1 (8 bits)	30.0	1350	22978
SHA-1 (32 bits)	30.0	1382	23968
SHA-1 (64 bits)	30.0	1454	26340
Rabin (64 bits)	30.0	1220	21686

Table 3. Performance of re-clustering protocol from the base station to the network

Once again, execution time does not vary. We believe that this is because the actual execution time of the hash function is significantly smaller than that of the rest of the protocol and therefore has negligible influence. (We show the hash times in the next section.) It may be that in a large network with thousands of nodes, the execution time of the hash function begins to play a role.

In all cases, Rabin uses less memory, both RAM and ROM, than SHA-1. This is particularly useful in applications requiring small memory usage. The security of the algorithms is likely to be comparable as both have been available for some time and are well understood by the cryptographic community. While Rabin on 64 bits is not acceptably secure as an encryption mechanism, it is sufficiently strong as a hash function. Therefore, we recommend the use of Rabin to generate hashes in lightweight environments.

V. THE HASH FUNCTION IMPLEMENTATION

It is worthwhile to compare, independently of the protocols, the performance of our two hash functions. While SHA-1 is a standard choice, it is clear from Table 4 below that Rabin is faster with smaller RAM and ROM.

SHA-1 and Rabin are implemented as follows on TinyOS Version 2.1. For comparison, we take data input of 8 bits, 32 bits and 64 bits for SHA-1 and 64 bits for Rabin. As shown in Table 4, for SHA-1 the code consumes 126 bytes of RAM, 3892 bytes for 8 bits, 3896 bytes for 32 bits and 3900 bytes for 64 bits of ROM, and takes approximately 7 ms to produce a 160-bit hash of different sizes of messages. Rabin produces a consecutive 32-bit hash window for a 64-bit calculated hash. The code consumes 46 bytes of RAM and 1870 bytes of ROM, and takes approximately 2 ms to hash a message of 64-bits.

Primitives	Execution Time (msec)	RAM (bytes)	ROM (bytes)
SHA-1 (8 bits)	7	126	3892
SHA-1 (32 bits)	7	126	3896
SHA-1 (64 bits)	7	126	3900
Rabin (64 bits)	2	46	1870

Table 4. Comparison of the Rabin and SHA-1 hash functions

Lee, Choi and Kim [12] have introduced a Hash component for TinyOS, designed to be used as an independent hash module with TinySec. According to the authors, this hash module accepts any hash function, and can be implemented on 8, 16 and 32 bit words.

We compared time and memory for our implementation of SHA-1 against theirs and also against an implementation in [10]. While we found that our results, from Table 4, are better than those in [12], 2005, and [10], 2008, for 8 or 32 bit input, we note that their papers do not specify the TinyOS version used and we believe that the difference is due to their use of an earlier version of TinyOS or different coding structure or techniques.

VI. ATTACK SCENARIOS

The standard attack scenarios in a WSN setting are known as the passive and active attack [18] models for communication compromises. In a passive attack, the intruder is able to capture and interpret data, thus extracting information; our protocols are not designed to protect against a passive attack.

In an active attack, both the integrity and the availability of the communication are threatened. An intruder implementing an active attack may destroy the integrity of a communication in two ways: by changing the content of the communication and by changing the authorship of the communication. An intruder implementing an active attack may also

- capture a communication and delete it altogether
- resend it at a later time, or
- resend several communications many times in a flooding attack.

An additional active attack which we will consider here is a situation where a compromised node which has received a reprogramming message sends an acknowledgement without actually reprogramming. In other words, it lies about having reprogrammed.

Here we discuss how each of these attacks is managed in the network

Passive Attack Scenarios

We assume here that a set of intruders or compromised nodes collaborate to implement a passive attack by capturing communications. Our architecture allows this to happen and allows such colluders to read data as it is all sent in the clear. The colluders may hope to capture WSN secrets S_n shared by the BS and node n. These are never sent in the clear, but only in hashed communications; if colluders capture the hashes, they cannot determine the secrets. Even if the colluders include compromised nodes in the WSN, these share no secrets with other nodes and therefore are unable to capture secrets of nodes not in the collusion.

Active Attack Scenarios

We identified several types of active attack earlier in this section. Here, we consider each of them. Note that as our protocol is designed only to provide recovery message and source authentication in this paper; and we cannot necessarily guarantee protection against all attacks mentioned here.

(i) Message Content Changes

In this attack a set C of colluders of intruders and compromised WSN nodes attempt to change the content of a captured recovery message M broadcast from the BS. Such a message was intended for node n. C changes M to M' and produces c' and R'. However, in order to compute a valid hash, C needs to have the valid secret S_n of the target node. Thus our protocol protects against such an attack: $h(m) \neq h(S_n \otimes M' \otimes R')$.

(ii). Message Authorship changes

Assume that a set of colluders C comprising intruders and compromised nodes wishes to send a recovery message to a node n and pretends that such a message originates from the BS. In this case, as in scenarios (i), C needs access to the secret S_n shared by the BS and node n. It does not have this information. A similar problem arises in protocol III where the message originates from an aggregator and is directed to the BS. Thus, our architecture protects against a message authorship attack.

(iii). Message Deletion

Assume that a set C of intruders and compromised nodes colludes to delete recovery messages in the WSN. A key assumption (see the Introduction) is that the network, via the BS, has the ability to detect compromised nodes. This is usually done by monitoring network message flow. Thus the BS can detect interruption of communications within the network, but our architecture for authentication does not have the capability of recovering deleted messages. In dealing with such a disruption the BS would attempt to recover compromises nodes by, for instance, reprogramming.

(iv) Message Replay

A set C of intruders and compromised nodes attempts to replay a valid recovery message M, R and $h(m) =$ issued by an aggregator or the BS. If C sends M, R and $h(m) = c$ captured earlier, the node (or BS) receiving the information checks if it has previously received such a message with the same value for R. If it has, it drops the corresponding packets. If C computes the current time R' and sends this along with the captured M and $h(m) = c$, the recipient will compute $h(Sn \otimes M \otimes R') \neq c$. Thus our protocol protects against replay attacks.

(v) Flooding

Assume a collusion C of intruders and compromised nodes captures and resends network messages in a flooding attack. Because the BS monitors network communications, it can detect such an attack. It would attempt to deal with it by reprogramming and reclosing but there is no guarantee of success. Our authentication architecture alone cannot protect against such an attack.

(vi) A compromised node lies

In this attack, we assume that a compromised node has received a reprogramming request from the BS. It attempts to send an acknowledgement to the BS confirming that it has reprogrammed and re-instated its original code. In order to do this, it must compute a hash of its secret along with the current time and an acknowledgement M: $h(Sn \otimes M \otimes R')$. However, we assume that the attacker does not have access to the tamper-proof section of the node and so cannot compute this.

VII. COMPARATIVE PERFORMANCE

In this section, we compare our results with those of other researchers who have developed authentication protocols for a similar situation.

In [2], Benenson, Gedicke and Raivio establish an authentication protocol for a WSN based on elliptic curve cryptography. This paper is the only other paper in the literature proposing a WSN authentication protocol without a key-discovery phase: a user in a fixed location broadcasts identity and a certificate. A node responds with a nonce. The user replies with a hash.

1 hash and 2 verifications are needed while in our protocol only 1 hash and 1 verification are needed. The authors of [2] rely on the existence of a third party trusted certificate authority while we assume only that the BS is trusted. In

summary, we avoid the need for a trusted third party and reduce the number of verifications.

Benenson, Gedicke and Raivio implemented their protocol on five TelosB motes using TinyOS. The hash function SHA-1 on 64 bits was used in [2] to hash the identities of the sender and receiver along with a random number. We see in Table 5 that our protocol is more than 10 times faster and requires almost 50% less in terms of memory requirements.

Primitives	Execution Time (sec)	RAM (bytes)	ROM (bytes)
Benenson, Gedicke, Raivio [2]	440	2000	45500
Our protocol III with SHA-1	30	1454	26340
Our protocol III with Rabin	30	1220	21686

Table 5. Comparison of two implementations of protocol C

They point out that the execution time is disappointing due to the long execution times of the elliptic curve cryptography routines. Note also, that in their protocol, motes must be capable of executing both symmetric and public key cryptographic protocols and must be able to securely store secret keys.

VIII. CONCLUSIONS

In this paper, we focus on recovery after an attack has been detected and provide lightweight protocols for re-clustering and for reprogramming nodes in a WSN. We provide a message authentication protocol for limited resource sensor networks enabling the network to securely implement recovery strategies efficiently and effectively. The protocol is based on hash functions and we compare the performance of two well-known lightweight hash functions, SHA-1 and Rabin. We demonstrate that our authentication protocol can be implemented efficiently with TelosB motes in comparison with existing protocols.

REFERENCES

[1] Basagni, S., Petrioli, C. and Petroccia, R. 2007. "Fail-Safe Hierarchical Organization for Wireless Sensor Networks". Proceedings of MILCOM 2007. IEEE, 1-7.

[2] Benenson, Z., Gedicke, N. and Raivio, O. 2005. "Realizing robust user authentication in sensor networks". In Workshop on Real-World Wireless Sensor Networks, 2005, Stockholm, Sweden. 5 pages. Available at http://www.mobnets.rwth-aachen.de/pub/realizing-authentication.pdf.

[3] Defence Signals Directorate Information and Communications Technology Security Companion, September 2007.

[4] Deng, J., Han, R. and Mishra, S., 2005, "Defending against path-based DoS attacks in wireless sensor networks", In Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (Alexandria, VA, USA, November 07 - 07, 2005). SASN '05. ACM Press, New York, NY, 89-96.

[5] Dutta, P., Hui, J., Chu, D. And Culler, D. 2005. "Towards secure network programming and recovery in wireless sensor networks". Technical Report UCB/EECS-2005-7.Available at http://www.eecs.berkeley.edu/Pubs/TechRpts/2005/EECS-2005-7.html.

[6] Hartung, C., Balasalle, J and Han, R. 2005. "NodeCompromise in Sensor Networks: The Need for Secure Systems", Technical Report CU-CS-990-05, University of Colorado at Boulder, 9 pages.

[7] Jeong J. and Culler D., 2004 "Incremental network programming for wireless sensors". In Proceedings of the First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks IEEE SECON (2004), 25-33 .

[8] Hu Y., Perrig A. and Johnson D., 2002 "Ariadne: a secure on-demand routing protocol for ad hoc networks". In *8'th Annual International Conference on Mobile Computing and Networking (MobiCom' 02)*, 12-23 .

[9] Karlof, C., Sastry, N. and Wagner, D. 2004. "TinySec: A link layer security architecture for wireless sensor networks". SenSys '04. ACM, 162 - 175.B.

[10] Kausar, F., Hussain, S., Yang, L.T. and Masood, A. 2008. "Scalable and efficient key management for heterogeneous sensor networks". J. Supercomputing, Vol. 45, 44-65.

[11] Krishnan, R. and Starobinski, D. 2004."Efficient Clustering Algorithms for Self-Organizing Wireless Sensor Networks," *Journal of AD - Hoc Networks*, 36-59.

[12] Lee, H., Choi, Y. and Kim, H. 2005. "Implementation of TinyHash based on hash algorithm for sensor network". Proceedings of World Academy of Science, Engineering and Technology, Vol. 10, 135-139.

[13] Li, B. and Batten, L.M. 2009. "Using mobile agents to recover from node and database compromise in path-based SDOS attacks in Wireless Sensor Networks". *Journal of Network and Computer Applications.* Accepted February 2008. In press.

[14] Liu, D. and Ning, P. 2003, "Efficient Distribution of Key Chain Commitments for Broadcast Authentication in Distributed Sensor networks", In *10'th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, 263-276.

[15] Lindsey, S. and Raghavendra, C. S. 2002, "PEGASIS: Power Efficient Gathering in Sensor Information Systems," In *Proceedings of IEEE Aerospace Conference*, Big Sky, MT, USA, 9-16 March 2002, vol 3, pp. 3-1125-3-1130.

[16] National Institute of Standards and Technology, 2002. 'Secure hash standard', FIPS Publication 180-2. 32 pages.

[17] Perrig, A., Szewczyk, R., Tygar, J., Wen, V. and Culler, D. 2002, "Spins:Security Protocols for Sensor Networks", Wireless Networks Journal (WINET), 8(5): 521-534.

[18] Oppliger, R. 1996. Authentication Systems for Secure Networks. Artech House, London, Boston.

[19] Rabin, M. 1979. "Digitalized Signatures and Public-Key Functions as Intractable as Factorization" MIT Laboratory for Computer Science, January 1979, 16 pages.

[20] Shamir, A. 2008. "SQUASH – A new MAC with provable security properties for highly constrained devices such as RFID tags". In FSE 2008. LNCS 5086, 144-157.

[21] Crossbow http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf.